NEAR REAL-TIME WEB-BASED MAPPING OF LIVE VIDEO STREAMS FROM UNMANNED AERIAL VEHICLES

İNSANSIZ HAVA ARAÇLARINDAN AKAN CANLI VİDEO GÖRÜNTÜLERİNİN YARI GERÇEK ZAMANLI OLARAK WEB TABANLI HARİTALANDIRILMASI

EFE PİRİHAN

ASST. PROF. DR. MURAT DURMAZ

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Geomatics Engineering

June 2022

To my family.

ABSTRACT

NEAR REAL-TIME WEB-BASED MAPPING OF LIVE VIDEO STREAMS FROM UNMANNED AERIAL VEHICLES

Efe Pirihan

Master of Science, Geomatics Engineering

Supervisor: Asst. Prof. Dr. Murat DURMAZ

June 2022, 136 pages

Unmanned Aerial Vehicles (UAVs) can be controlled autonomously or manually via a ground station and reduce the dependency of mission success on human capability. They can perform tasks longer than a pilot's ability to carry out while avoiding the risk of life loss. UAVs are used for the purposes of target determination, reconnaissance and observation, clash and logistics in the fields of military operations. For most of these use-cases the image/video stream taken by the digital camera payload on the UAV is very important for the success of the mission. Near-real time display of these images especially by georeferencing information brings additional advantages such as being able to monitor the operation area instantly and examine the current view which can greatly improve the situational awareness of the decision maker. They also can create a communication channel between people in distress and rescue teams in case of natural hazard. In most of the scenarios, images from multiple UAVs shall be consumed by multiple decision makers simultaneously. This creates a bottleneck on the infrastructure especially in the communication channels. In this study, a software pipeline that can display the images/streams coming from multiple UAVs on web-based 3D virtual globes as georeferenced video frames is proposed. In this way, it is aimed to transmit the images coming from operation area with the associated georeferencing information in near

real-time and without the need for powerful servers by establishing a virtual network based on WebRTC technology. The proposed network topology and pipeline is implemented using CesiumJs as virtual globe and network and bandwidth utilization is tested. In addition direct georeferencing techniques and visualization of the georeferenced video is demonstrated. Besides, an application was developed that can push image and orientation information from an android phone to the same pipeline. According to the results, if high quality position and orientation information of video streams from UAVs are provided, the proposed pipeline can provide near real-time mapping of video streams on virtual globes.

Keywords: UAV, Cesium, WebRTC, Direct Georeferencing, Geovisualization

ÖZET

İNSANSIZ HAVA ARAÇLARINDAN AKAN CANLI VİDEO GÖRÜNTÜLERİNİN YARI GERÇEK ZAMANLI OLARAK WEB TABANLI HARİTALANDIRILMASI

Efe Pirihan

Yüksek Lisans, Geomatik Mühendisliği

Danışman: Asst. Prof. Dr. Murat DURMAZ

Haziran 2022, 136 sayfa

İnsansız Hava Araçları (İHA'lar), bir yer istasyonu aracılığıyla otonom veya manuel olarak kontrol edilebilmektedir ve görev başarısının insan kabileyetine olan bağımlılığını azaltmaktadır. IHA'lar can kaybını azaltırken bir pilotun gerçekleştirebileceğinden daha uzun görevleri yerine getirebilmektedir. İHA'lar, askeri harekat alanlarında hedef belirleme, keşif ve gözlem, çatışma ve lojistik amaçlı kullanılmaktadır. Bu kullanım alanlarının birçoğu için İHA üzerindeki dijital kamera tarafından çekilen görüntü/video akışının sağlanması, görevin başarıyla sonuçlanması için önemlidir. Bu görüntülerin özellikle koordinatdırılarak yarı gerçek zamanlı olarak görüntülenmesi, operasyon alanını anlık olarak izleyebilme ve mevcut görünümü inceleyebilme gibi ek avantajlar sağlayarak karar vericilerin durumsal farkındalığını arttırmaktadır. Ayrıca İHA'lar doğal afet veya kaza durumunda tehlike altındaki kişiler ve kurtarma ekipleri arasında bir iletişim kanalı oluşturabilmektedir. Çoğu senaryoda, birden fazla İHA'dan gelen görüntüler, aynı anda birden fazla karar verici tarafından görüntülenmektedir. Bu da altyapıda özellikle iletişim kanallarında darboğaz yaratmaktadır. Bu çalışmada çok sayıda İHA'dan gelen görüntüleri web tabanlı 3 boyutlu sanal küre üzerinde aynı anda görüntüleyebilen bir yazılım hattı

önerilmiştir. Bu sayede istemci makineler arasında WebRTC teknolojisine dayalı sanal bir ağ kurularak, operasyon sahasından gelen görüntünün koordinat bilgisi ile birlikt yarı-gerçek zamanlı ve güçlü sunuculara ihtiyaç duymadan iletilmesi amaçlanmaktadır. Önerilen ağ topolojisi ve yazılım CesiumJs kullanılarak geliştirilmiş, ağ ve bant genişliği kullanımı test edilmiştir. Bunlara ek olarak doğrudan coğrafi referanslama yöntemleri ve coğrafi referanslanmış videonun görselleştirilmesi üzerine çalışılmıştır. Bunun yanında Android telefondan görüntü ve oryantasyon bilgisini alarak aynı yazılım hattına verileri gönderen bir uygulama geliştirilmiştir. Sonuçlara göre, İHA'ların pozisyon ve oryantasyon bilgileri yüksek doğrulukla sağlandığında, önerilen yazılım hattıyla bu görüntülerin sanal küre üzerinde yarı gerçek zamanlı gösterimi yapılabileceği görülmüştür.

Keywords: İHA, WebRTC, Cesium, Coğrafi Referanslandırma, Coğrafi Görselleştirme

ACKNOWLEDGEMENTS

First of all, I would like to express my great gratitude to my supervisor, Asst. Prof. Dr. Murat DURMAZ, for his guidance, support and encouragement throughout the study.

I would like to thank my thesis committee members, Assoc. Prof. Dr. Sultan KOCAMAN GÖKÇEOĞLU, Prof. Dr. Ali Özgün OK, Assoc. Prof. Dr. Saygın ABDİKAN and Assoc. Prof. Dr. Fatih NAR for their valuable suggestions, comments, and contributions.

I also would like to thank my wife Belmen PİRİHAN, my father Ramazan PİRİHAN and my mother Selma PİRİHAN for their unconditional love and support.

CONTENTS

Page

TABLES

Page

FIGURES

Page

ABBREVIATIONS

UAV	:	**U**nmanned **A**erial **V**ehicle
WebRTC	:	**W**eb **R**eal **T**ime **C**ommunication
DASH	:	**D**ynamic **A**daptive **S**etreaming **o**ver **H**TTP
RTSP	:	**R**eal **T**ime **S**treaming **P**rotocol
C/S	:	**C**lient / **S**erver
TAIC	:	**T**errain **A**ware **I**mage **C**lipping
3D	:	**3 D**imensional
2D	:	**2 D**imensional
GML	:	**G**eography **M**arkup **L**anguage
RCNN	:	**R**egion Based **C**onvolutional **N**eural **N**etworks
RFCN	:	**R**egion Based **F**ully **C**onvolutional **N**etworks
mAP	:	**M**ean **A**verage **P**recision
GPS	:	**G**lobal **P**ositioning **S**ystem
GNSS	:	**G**lobal **N**avigation **S**atellite **S**ystem
SAR	:	**S**ynthetic **A**perture **R**adar
ROS	:	**R**obot **O**perating **S**ystem
ALPR	:	**A**utomatic **L**icense **P**late **R**ecognition
FPS	:	**F**rame **P**er **S**econd
AIS	:	**A**utomatic **I**dentification **S**ystem
TCP	:	**T**ransmission **C**ontrol **P**rotocol
UDP	:	**U**ser **D**atagram **P**rotocol
IP	:	**I**nternet **P**rotocol
HTTP	:	**H**ypertext **T**ransfer **P**rotocol
B/S	:	**B**rowser / **S**erver
API	:	**A**pplication **P**rogramming **I**nterface
CPU	:	**C**entral **P**rocessing **U**nit

GPU	:	**G**raphics **P**rocessing **U**nit
MQTT	:	**M**essage **Q**ueuing **T**elemetry **T**ransport
MPEG	:	**M**oving **P**icture **E**xperts **G**roup
XML	:	**E**xtensible **M**arkup **L**anguage
NAT	:	**N**etwork **A**ddress **T**ranslation
ICE	:	**I**nteractive **C**onnectivity **E**stablishment
STUN	:	**S**ession **T**raversal **U**tilities for **N**AT
TURN	:	**T**raversal **U**sing **R**elays around **N**AT
SCTP	:	**S**tream **C**ontrol **T**ransmission **P**rotocol
MCU	:	**M**ultimedia **C**ontrol **U**nit
SFU	:	**S**elective **F**orwarding **U**nit
IOT	:	**I**nternet **o**f **T**hings
ENU	:	**E**ast **N**orth **U**p
NED	:	**N**orth **E**ast **D**own

Chapter 1

INTRODUCTION

UAVs are controlled by radio communication or by mechanisms that allow them to proceed autonomously on a certain route. Drones are generally equipped with optoelectronic equipment used for imaging and surveillance. The most important feature of these vehicles is that the desired area or object can be observed without the need for an additional infrastructure. Another important advantage is that it can be made ready for flight in a very short time. UAVs can be equipped with proper equipment to monitor large areas, so they can be used in critical zones such as state borders that need to be protected. With night vision and thermal cameras, they can provide images in different illumination conditions. They can provide real time images in case of emergency. Thanks to their small size, they can move between obstacles such as buildings, trees and pass-through passages such as windows and doors [23]. Drones are critical when people are unable to perform dangerous/risky task in a timely and efficient manner or in many situations from rush hour delivery services to scanning inaccessible areas. It offers new opportunities in different applications including civil infrastructure in terms of lower risk and lower cost [24]. Drones can be used by the military to follow a target, to provide direct fire support, to spy on the operation area, by police to monitor traffic jams, to track events, by the energy and chemical industry to monitor gas emission levels and by companies for mapping and fast visualization of the area [23]. The use of drones is rapidly increasing in many civilian application areas

including real-time monitoring, wireless coverage, environmental protection, search and rescue operations, surveying, remote sensing, remote controlling, goods delivery, agriculture and civil infrastructure inspection [12].

The widespread use of drones brings along needs such as real-time image transmission and display of the drone footprint on a virtual globe. There are many protocols that offer real-time image transmission capability. Websocket, Dash, RTSP and WebRTC are some commonly used protocols for data transmission. Although Websocket is mostly used to transfer data in text format, there are also studies on video transmission over this protocol. It has been studied in [25] that images in Jpeg format can be converted to base64 and transferred over websocket to be displayed on the browser, without the need for extra plugins. In [26], it has been studied on transferring images from drones over websocket in emergency situations. Another protocol used for video transmission is DASH. It basically works on the principle that the video is divided into segments of equal size and served. It is stated in another study [27] that this protocol reduces the Quality of Experience due to the switching delay in systems whose network conditions change frequently. Transferring images from drones using adaptive streaming in low-capacity networks has been studied in [3]. RTSP is a non-connection-oriented protocol developed at the application level. It works with the traditional client-server model. In [28], the development of a real-time remote video monitoring system using RTSP has been studied. WebRTC is a streaming protocol that provides peer-to-peer communication, which has been widely used in recent years. Performance of WebRTC in networks with limited capacity is examined in the study [29]. WebRTC and RTSP were compared in the study [30] and it was seen that WebRTC gives better results. WebRTC offers infrastructure not only for video but also for the transfer of audio and text data. In this context, sharing of files between client applications is studied in [31], and sharing player information in online games is studied in [10]. In [32], WebRTC datachannel was used to eliminate the delay brought by TCP's handshake mechanism while transmitting Dash data. Establishing communication over WebRTC in emergency situations is also studied [33]. In [11], a flying communication server system working with WebRTC was developed in order to provide communication in such cases.

WebRTC was used to transfer drone images in [34]. The images obtained from a drone camera and the data obtained from the sensors are sent over WebRTC in a single stream [35]. Except for WebRTC, the protocols examined within the scope of the study work with the classical client/server model. It is concluded that when the number of active users in the system increases, powerful servers will be needed. In summary, WebRTC technology has some advantages over websocets such as; peer-to-peer communication, no need for extra plug-in installations, being supported in all modern browsers, good documentation, working in low-bandwidth networks, transferring data in text format in addition to video and audio formats. When near real-time video stream and its associated georeferencing information is available it can be rendered on web based virtual globes for ubiquitous use.

With the rapid development of web and computer graphics technologies, the development of web mapping applications are widespread. Real-time sensor and orthophoto data can be visualized on these mapping applications. In this context, one of the most widely used API is Leaflet. Its advantages such as being opensource and leightweight are mentioned in [36]. An Audio Visual Web Cartography application is developed using Leaflet [13]. The visualization of the natural environment in the pre-industrial period on Leaflet is studied in [37]. In [38], real-time monitoring of air pollution has been demonstrated. Openlayers is another web mapping application. It offers capabilities such as management and visualization of spatial data, real-time observation of rain intensity [39], observation of marine areas [40] and visualization of agricultural data [29]. Another frequently used web mapping tool is Cesium. CityGML objects are displayed and buffer zone and intersection analyzes were performed on these objects in [41]. In another study a sensor data management system was developed on Cesium to monitor cycling and fitness level of users [15]. In [42], images obtained from time-varying/multidimensional climate data are shown on Cesium.

Web based virtual-globes can provide an important base displaying real-time video streams from UAVs. In this context, the video stream shall have a synchronized telemetry data that contain georeferencing information such as position and orientation of the digital camera on the platform. WebRTC tehnology combined with a Cesium based virtual globe can be combined together to both disseminate the image and georeferencing information to web

based clients. Georeferencing of an image can be categorized under two methods: indirect and direct georeferencing. Indirect georeferencing needs accurate ground control points that can be located on images. Methods such as bundle block adjustment are included under indirect georeferencing. However, the determination of ground control points is a costly process and it is not very possible to establish ground control points for every region. In direct georeferencing the image is georeferenced using position and orientation information of the digital camera without the need for ground control points. The calculation of the amount of landslide using structure from motion, which is one of the direct georeferencing methods and can work both with and without ground control points, is studied in [43]. The Terrain Aware Image Clipping algorithm developed to reduce the amount of images used in direct georeferencing is described in [18]. Real-time mapping of aerial photographs is also studied [19]. In the scope of this study, direct georeferencing method was preferred due to the difficulties and constraints related to obtaining ground control points. A web based technology for dissemination and mapping of georeferenced video streams from multiple UAVs to multiple clients has not yet been studied in the literature as to the authors knowledge at the time of this study.

Data processing can be categorized as batch processing, near real-time and real-time. In batch processing, large amounts of data are collected, processed and then served in batches. It is less time sensitive than near real-time and real-time data processing and can take days or even weeks. When the response time, responsiveness and latency of the system are important, near real-time and real-time data processing methods are used. The difference between near real-time and real-time data processing depends on the precision of the amount of delay. While the amount of delay can be clearly specified in real time systems, this amount of delay can vary in near real-time systems [44]. The developed system is defined as near real-time because of the changes in the delays experienced in the distribution of data depending on the number of users and network conditions.

1.1. Objectives and Methodology

Mapping of near real-time aerial images from the drones by many clients via WebRTC on a web-based virtual globe is an important technology for many emergency and defence applications. The main purpose of this study is to propose a software pipeline where the images obtained by drones from the operation area in situations such as natural hazards, accidents or defence operations are quickly shared among the people who manage/monitor the operation without the need for a strong server infrastructure. In addition to this, another aim of the study is to display the image obtained and shared among the clients on the virtual globe with additional georeferencing information.

A software architecture is proposed in this study that manage, disseminate and display near real-time video streams with associated telemetry information. Mainly Javascript and Java programming languages are used to develop signaling server, video and telemetry management and serving, and Cesium based web client development. In order to test the system, a prerecorded drone video is utilized in a series of test scenarios where proposed WebRTC based solution is compared to conventional websocket implementation. A Cesiumjs-based web application is developed and the frame by frame georeferencing of the flight video is done and displayed on the virtual globe at the clients in order to distribute the computation load. With this study, it has been seen that not only drone videos, but also videos transferred from Android phones can be displayed on the virtual globe which may be considered as a geo-cast from individuals.

1.2. Contributions

This thesis contributes to the related literature in the following aspects:

- A software pipeline is proposed to visualize live georeferenced video streams from drones across multiple client applications without the need for powerful servers.

- Performance comparison was made by providing live video over Websocket and WebRTC.
- The approximate direct georeferencing of real-time aerial photographs on Cesium is demonstrated.
- By using android sensors, images obtained from these devices were transferred to cesium via WebRTC and visualized on Cesium which may be an interesting contribution to geo-casting from individuals.

1.3. Organization

Chapter 1 briefly summarizes the literature on real-time data transfer, georeferencing and web mapping concepts used in the thesis and presents our motivation, contributions and the scope of the thesis. Chapter 2 provides the usage areas of UAVs, DASH, SRTP, Websocket and WebRTC technologies. Chapter 3 provides details about the proposed method. Chapter 4 provides the applications developed, the results of the tests performed, and the discussion of the results. Finally, a summary of the findings obtained as a result of the study and future developments are given in Chapter 5.

Chapter 2

RELATED WORK

The system proposed in this study combines UAVs, real-time data transfer, georeferencing and web mapping technologies. In this section, studies on the use of drones in emergency situations, data transfer protocols, georeferencing methods and open source web mapping libraries are explained.

2.1. Drone Assisted Object Detection and Surveillance

In this section, use of drones in natural hazard, studies on monitoring highway and marine areas and object classification on the aerial images taken from drones are listed.

2.1.1. Object Detection

Object detection applications through aerial photographs are becoming more and more common. In natural hazards, drones can quickly scan large areas affected by disasters and speed up search and rescue activities and saving more human lifes. With the developing deep learning methods and sensor technology, various analyzes can be made on the drone images and location information of important assets and even people can be extracted.

In the literature, there is not enough data set based on human and motion detection required to train and create object models to detect survivors in a automatic way with deep learning methods. A data set is proposed to be used in search and rescue activities and results with 7% improvement are obtained than the standard Okutama dataset [1]. It was aimed to extract the coordinates of the detected objects from the image by direct video analysis and object detection, and to send this information to central server first and then to the search and rescue teams. In this way, it is aimed to quickly determine the locations of the victims in the disaster area and to intervene quickly by the search and rescue teams. The video images coming from the camera to drone are processed with SSD object detection algorithm and the bounding box of the objects are calculated using non-maximum suppression method. The provided data set and standard Okutama data set are divided into training and test data sets, object detection is made using Faster RCNN, RFCN and SSD on these data sets. When the mean average precision (mAP) metrics of the obtained results calculated, it was seen that results %7 better results are obtained with the single shot multi-box detector (SSD) method.

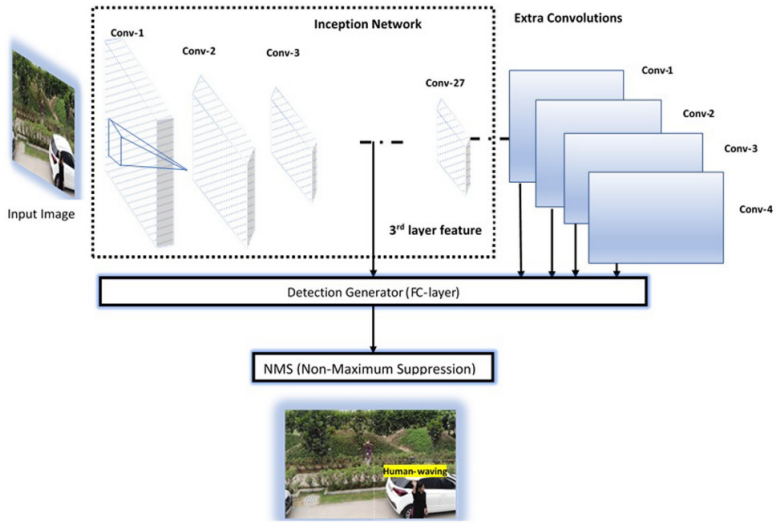

Figure 2.1 System architecture for detecting human actions [1]

Terrorist attacks are also situations that require urgent intervention, as in natural hazards. In these cases it is possible to track targets without endangering human life via drones. A system that enables the observation of the target from a distance by means of a Quad Copter equipped with a Wi-Vi sensor, infrared camera and GPS has been designed in cases where

it is not possible for people to reach or is dangerous [45]. The main purpose is to detect survivors or enemies behind walls or obstacles. Thermal cameras generate data by detecting body temperature. The data obtained here is used together with the data obtained from the Wi-Vi sensor. The results of the study shows that while the accuracy rate of measurements at a distance of less than 5 meters is very high, the measurements at distance of 10 meters and above give completely wrong results.

Observing and analyzing events such as concerts, celebrations and meetings has become an important research topic. Early detection and prevention of crowd events is a challenging task for police and crisis management teams. Crowd detection and crowd counting methods can play an effective role in preventing possible incidents by providing useful information from the field such as crowd density. However, the important thing here is that these methods have to be done in real time in order to intervene quickly. While real-time images can be obtained from ground cameras fixed at certain points, drone is a good alternative as they can quickly deliver videos from the field. Detecting the crowd through drone videos with the CNN-based state-of-the-art approach are compared [46]. The developed algorithm was tested on two different drone videos and the result are compared with the results obtained by the Cascaded-MTL method.

Real-time monitoring of parking lots helps reduce vehicle queues and time spent searching for empty parking spaces. In many parking areas, this information is provided by sensors placed on the ground. However, this method requires the placement of these sensors for each parking area and continuous maintenance. In addition, this method becomes very costly for very large parking areas. To solve this problem determining the spaces in parking areas and detecting license plates in real time is studied [47] where drones with integrated cameras are used. In the first stage of the study, the parking areas were mapped using drone coordinates and Traveling Salesman algorithm is used to visit all parking areas as soon as possible to obtain images. The gimbal angle and drone height are dynamically adjusted during flight according to the conditions of the location where the video is taken. Empty and occupied parking lots were determined by using a neural network algorithm on the obtained images. In addition, vehicles were checked for compliance with the rules using

the ALPR algorithm. The outputs of this algorithm are saved in the cloud database and a web application is provided to the user to query this database. In the test conducted with the DJI Matrix 100 drone, the detection of the empty parking spaces was performed with and accuracy of 95.83%, while accuracy of 85.83% was obtained from the license plate identification algorithm.

2.1.2. Surveillance

In many civil and military operations, it is necessary to monitor the site of interest. For instance, the police patrol around the city to ensure security and to ensure that the rules are followed. Natural hazards occur all over the world at any time. After a disastrous geological or meteorological hazard event, the current situation should be examined before the search and rescue teams are sent to the field. An area prone to geological or meteorological hazard is often difficult to cross due to obstacles such as steep slopes or streams. In such cases, images can be obtained from the field via drones and the operation can be accelerated without endangering human life.

Within the scope of the study [48], a drone surveillance system with the ability to be used in many military and civilian areas has been developed. A high-resolution camera with zoom in, zoom out and night vision capabilities is integrated on the drone and this camera is connected to the Raspberry Pi on-board computer. A wireless adapter was also added to the drone and the mini computer was connected to this network. In this way, it was ensured that the laptop in the ground control center was on the same network and the images obtained from the camera are sent over this network. This network also allowed the ground control center to access and send commands to the on-boards computer using Secure Shell. Applications such as Mission Planner and APM Planner, which are open-source software, are used on the computers in the ground control center. With these applications, information such as live location and altitude of the drone can be displayed, and automatic flight routes are created. A webcam server running in the background has been created by using Motion software, which examines the camera signals of the Raspberry Pi computer and detects whether there

is any change in the image. The images are transferred to the single LCD monitor in the ground control center. The system has been tested in manual mode and in automatic mode by creating flight path, and it has been seen that 100 fps images can be taken from the camera.

Highways cover very large areas and are therefore difficult to observe. In order to obtain data from these areas, sensors such as cameras shall placed in fixed locations and intensive labor force needed. Personnel are sent to places where these sensors cannot observe, and measurements and observations are made manually. Monitoring of traffic by drones and applications that can be developed based on the data obtained from drones are examined [49]. In this study, advantages such as the ability of drones to move faster than vehicles moving on highways, not having to move on the highway surface, eliminating the need to send personnel to the field due to their automatic flight capability and providing data from the field quickly are listed as benefits. In the study five use-cases of drones for calculation of highway density, calculation of traffic flow, observation of the intersections and queue length at intersections, calculation of empty parking lots are examined methodologically.

Maritime accidents cause loss of life and serious property damage. According to the report published by EMSA in 1018, 20,616 marine accidents occurred between 2011 and 2017 and 7,495 deaths/injuries occurred because of these accidents. There is a need to take measures to prevent greater losses and create situational awareness. The need to take quick actions in emergency has made it important to monitor the sea area with unmanned aerial vehicles. Accurately specifying the positions of ships is very important to reduce the number of possible accidents. Therefore, direct georeferencing and orthophotos obtained in real time is one of the main needs. Modern mapping techniques are highly advanced in terms of accuracy and time efficiency. In addition, the technology has become able to detect objects automatically, and many studies have been carried out for the detection of ships by using space-based sensors such as SAR. The use of AIS has been studied, but the fact that this method can only detect ships carrying AIS receivers has been recorded as a limitation. For this reason, there is a need for a system that can extract coordinates on real-time images. A system that can process, map, and display the sea images taken and served by drone in near real time has been developed [2]. System architecture is given in Figure 2.2. The

system consists of 3 separate subsystems: UAV module, real time image processing module and visualization module. While the drone module sends the data from the drone to the ground control center, the image processing module creates orthophotos from the aerial images. The visualization module enables these images to be displayed. Within the scope of this study, a drone containing an optical camera and GPS, which can wirelessly transmit the images obtained from sensors, has been selected. An interface has been developed for the drone to send data using Python-based Flask framework. Since it is not possible to determine the ground control points every time, a system designed to georeferencing images coming from drone in real time directly. The orientation parameters used to georeferencing the image was calculated by averaging the sensor data before and after the image acquired. While detecting the ships, the YOLOv3 method was used because it makes the calculations efficiently. When the image processing module completes the task, it uploads the images obtained to the visualization module. Mago3d, an open-source software, was used as the visualization module. Orthophotos are displayed through this module and capabilities such as listing the ship's details in case a ship is detected and displaying the flight route have been provided. The system was tested in two different regions, and it took an average of 3.26 seconds to obtain the image from the drone and display it on the visualization module. The coordinates calculated in these images obtained from a height of 500 meters are found with a difference of approximately 8.25 meters from the actual coordinates.

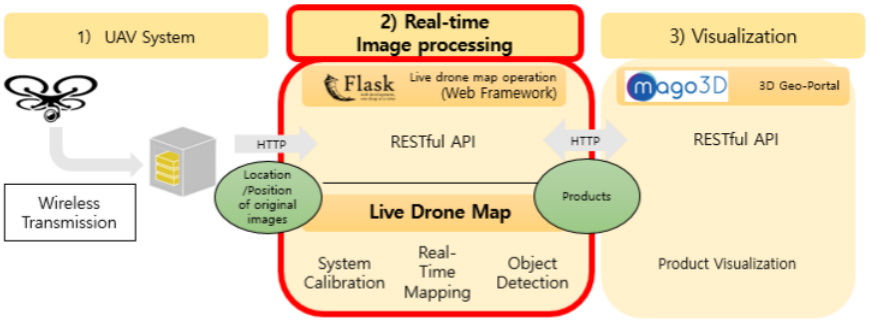

Figure 2.2 The architecture of marine surveillance system [2]

2.2. Data Streaming

With the discovery of new technologies, it was inevitable that the role of communication media in communication between people would increase. Among these, video streaming has become the most frequently used communication method. Although this way of communication is a good alternative to face-to-face meetings, it has some problems such as bandwidth, network congestion, reliability and cost [50]. Many protocols and infrastructures such as Websocket, DASH, RTSP, WebRTC have been developed to provide data streaming and to solve such problems. In this section, information about the most used data streaming applications is given.

2.2.1. WebSocket

WebSocket is a TCP/IP based protocol that provides real-time and bidirectional communication between a web server and a client. When the client application sends a request to the websocket server, if the server contains a service running on the websocket protocol, the communication between the client and the server is converted from HTTP protocol to websocket protocol and continues to serve on the same port as HTTP [25]. Websocket uses browser/server (B/S) architecture and pushing mechanism instead of classical client/server architecture and polling mechanism. With the push-based architecture, the client does not need to constantly check whether there is a change in the server [26]. Websocket can be used in many areas such as receiving data from sensors and chat applications. For instance real-time display of the sensor data coming from the laser on the client application and changing the orientation of the laser according to the angle values entered on the application screen can be achieved by bidirectional communication over the websocket.

Although websocket is mostly used to transmit data in text format, there are also studies on real-time video transmission over websocket in the literature. It has been studied in [26] that the images transferred over the websocket are displayed in the web browser

without the need for an extra download process or special hardware. The proposed system consists of 4 components: Server, Visualization Engine, Daemon and Client. Visualization engine processes the data it receives from databases such as geology, bioinformatics and simulation, then transforms it into an image in jpeg format by using CPU and GPU. Daemon works as a middle-ware between client and server, and hosts the websocket service.It sends the information from the client to the visualization engine and sends the outputs of the visualization engine to the client. The client application converts the images it receives via the WebSocket API to Base64 and displays it on the Canvas provided by HTML5.

Display of images obtained from the drone over websocket for use in emergency situations has been studied in [27]. With this system, which can be integrated into the EUROCOM Iot platform, it is aimed to support the operation management. A web application has been developed for those who follow the search and rescue operation from the center, and a mobile application for those who follow it from field. It is thought that with the developed system, rapid response to accidents, rapid transmission of small medical vehicles, and search and rescue activities for missing people can be supported. The north interface of the EUROCOM provides services such as access to sensor data, data management and discovery. Within the scope of this study, the north interface has been extended and custom services have been added to the access video and telemetry data. Mission definition service, which is one of these developed services, receives the GPS coordinates from which an alarm is generated and sends them to the drone assignment service, allowing the task o be assigned to a plot. Telemetry streaming service provides receiving drone telemetry data via MQTT. Another service provides real-time video transmission over websocket. The authors did not provide information on how the application was tested. Also, the advantages/disadvantages of using MQTT and websocket are not listed.

2.2.2. DASH

Dynamic Adaptive Streaming over HTTP is a standard developed by the Moving Picture Experts Group (MPEG) to provide a smooth transmission of multimedia data to client

applications under variable bandwidth conditions. DASH splits video content into segments of equal length and creates copies of each segment at different resolutions. Each segment stored on the server can be accessed via the XML-based MPD file. In systems using this standard, the resolution of the video is determined by the client application by measuring the bandwidth and stream processing speed. However, in systems whose network conditions change very frequently, Quality of Experience may be adversely affected due to view-switching delay. In addition, the buffer filling time can be reduced by controlling the buffer through the client application and transmitting the segments in parallel [51]. It is used by applications like Apple, Netflix, Youtube and Microsoft [3]. In the study [28], RTP/RTSP, HTTP Progressive and DASH methods were compared, and it was seen that better results were obtained with the DASH-based solution.

As mentioned in the previous sections, drones are frequently used for disaster response and surveillance. Today's UAVs usually stream videos with a fixed bit-rate. However, adaptive video streaming is required to solve the problems that may occur due to uncertain link capacity in real-time image acquisition scenarios. For example, in military operations or border patrols where real-time imagery is required, the success of the operation depends on the link capacity. In some scenarios, communication between the drone and the ground station may only be provided over low-capacity satellite links. In these cases, the system should operate in accordance with the low link capacity. In systems using Dynamic Adaptive Streaming over HTTP, the client application needs to make many controls and request videos with a resolution that the system can handle. In order to overcome these problems video transmission from drones is ensured by making content-based compression and video rate adaptation [3]. The system changes the video bit-rate by using the link capacity, the buffer capacity of the client and the location of the UAV depending on the time. In addition, compressed features are sent to the ground station using the compressed histogram of gradients (CHoG) method. The ground station checks whether the video frame is relevant by comparing the target objects with the compressed objects. If the frame is associated, it added to the transmit queue. The overall system design is shown in Figure 2.3. When the results of the study were examined, it was seen that the proposed method was more successful than

conventional throughput-adaptive methods.

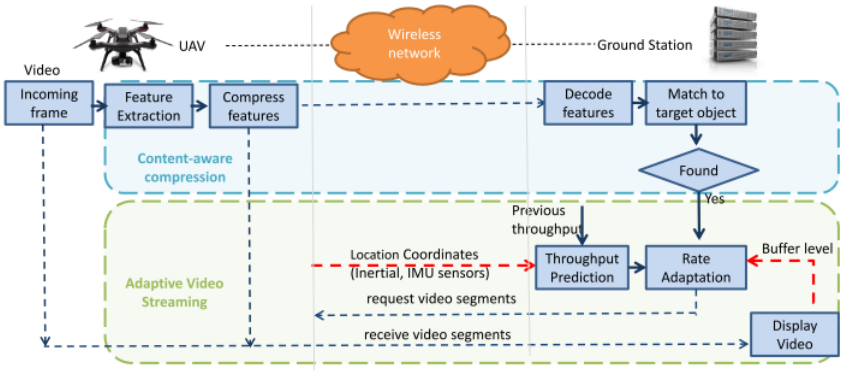

Figure 2.3 System implementation [3]

2.2.3. RTSP

RTSP (Real-time Streaming protocol) is an application-level and non-connection-oriented protocol for transmission of real-time video and audio streams. Besides transferring the media stream, RTSP also allows clients to control this stream in real time. With RTSP, live video can be directly taken from sources such as encoder or camera and played on standard media players such as VLC that support this protocol [50] .The Setup, Play and Teardown methods offered by RTSP are sufficient for the clients to create sessions over this protocol, play the media and close the streaming session. Figure 2.4 shows the requests to play streams over RTSP [4].

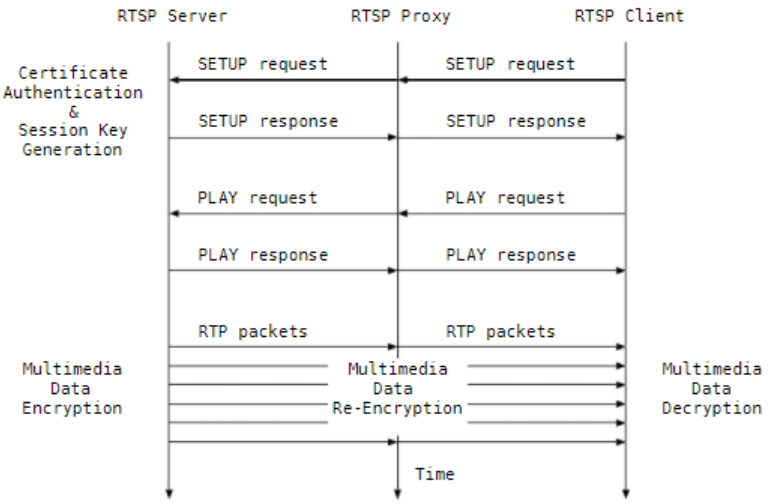

Figure 2.4 RTSP Operations [4]

There are many studies in the literature on playing videos over RTSP. A TCP/IP based remote video monitoring system was developed in [52]. The system consists of tree components: video acquisition sending terminal, network and client PC. Terminal is a development board with embedded processor and embedded Linux operating system. Images taken from the camera in YUV420 format are converted to H264 format with MFC hardware encoder and presented over RTSP protocol with Live555 library. Client computers can play these images with the VLC application that supports the RTSP protocol. With this system, remote streams can be displayed without distance limit. In addition, since the RTSP protocol has its own security mechanism, there is no need to take extra precautions. Besides, multimedia mail architecture developed for Internet telephony using SIP and RTSP is described in the study [53]. The performance of the RTSP protocol in learning systems has been studied in

2.2.4. WebRTC

WebRTC is a set of standards and protocols that provide peer-to-peer real-time data streaming between web browsers. It offers various APIs for audio, video and file streaming via HTML5 and JavaScript. Since all APIs it provides are based on peer-to-peer communication, data is transferred directly between clients and does not require an intermediate server [54].

WebRTC does not need an extra plugin to be installed on browsers, everything necessary comes embedded in browsers. Provides interoperability between different web browsers such as Google Chrome, Mozilla Firefox and Opera [7]. It allows software developers to develop high-performance applications by writing just a few lines of code [54].

WebRTC Architecture

Unlike the traditional server client architecture, WebRTC is based on peer-to-peer communication. In order to ensure communication, the information of the clients must be transferred between each other. This information flow is provided by a signaling server. Therefore, applications running on this framework need to develop a back-end signaling server. However, in cases where the clients are behind NAT, only the signaling server is not sufficient to ensure communication. WebRTC uses the ICE framework to solve this problem, and STRUN and TURN servers are needed to provide ICE support. Once the connection is made through the APIs used in the front-end and the servers running in the back-end, all communication is done directly between the clients [6].

Signalling Server The signaling server (Figure 2.5) provides the coordination between client applications and the transfer of information necessary for communication. While WebRTC only offers capabilities for media streaming, it leaves the development of the signaling server completely flexible to the developers. Thanks to this flexibility, developers can use software such as Kurento or develop this server themselves, depending on the application needs. Unlike WebRTC, the signaling server works with a client/server model, so it should be centralized. These servers keep the records of the online clients in the system and when one of the clients wants to communicate, they provide the necessary information to be shared between the clients to establish the connection. In addition, data such as control messages, error messages, network information are transmitted over these servers [7].

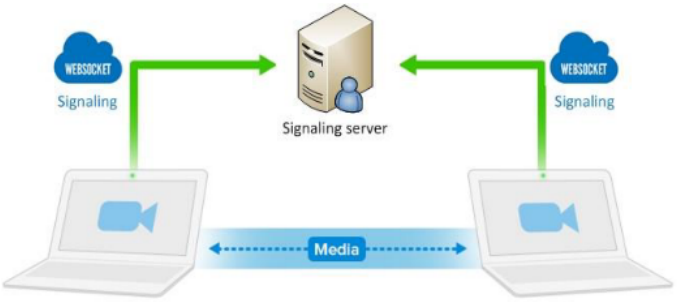

Figure 2.5 WebRTC signalling overview [5]

NAT Traversal Network Address Translation (NAT) is a mechanism of mapping private addresses to the public address and changing address information within IP packets while it is in transit through the device for routing. All computers connected to the Internet via a router will have the same external IP address, but a local IP address is also defined to distinguish them from each other. When a packet arrives at the router, the NAT table is checked and the packet is forwarded to the destination computer. The main reason for using NAT is the limited number of IPv4 addresses. Due to the increasing number of Internet users, this problem has been overcome by enabling many users to use the same IP. Another reason for using NAT is to provide security by separating public and private networks. There are basically four different NAT implementations. In Full Cone Nat, requests from the same internal IP and port are mapped to the same external IP and port. Therefore, any client can send requests to this machine using external IP and port. In Restricted Cone Nat, same internal host and port are mapped to the same external host and port like Full Cone Nat. However, in order for an external client to send request to internal host, the internal host must have requested this external IP previously. Port Restricted Cone Nat works with the same logic as Restricted Cone NAT except that it takes the port into account. A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port [54].

Even if the clients have forwarded their connection information to each other through the signaling server, the connection cannot be established because these clients are usually behind one or more firewalls or NAT. WebRTC uses Interactive Connectivity Establishment

to bypass NAT and firewall. Incoming ICE agents embedded in WebRTC try to find the best path to establish communication between two peers. ICE first tries to establish a connection directly via the IP address obtained from the signaling server. However, this local IP address allows connection to be established when both clients are on the same local network. However, if the two clients are not on the same local network, it tries to obtain the external IP address and establish a connection using the Session Traversal Utilities for NAT (STUN) server [7].

STUN provides a number of tools for providing NAT traversal. With the mechanism it provides, it is possible to learn the public IP and public port that NAT maps local IP and local port. Google has STUN servers available to the public. Besides, STUN works successfully on full cone, restricted cone and port restricted cone NATs. Clients running behind such NATs can communicate with each other using STUN servers. Providing peer negotiation via STUN server is illustrated in Figure 2.6. If even one of the clients is running behind Symetric NAT, STUN server will not be enough. To solve this problem, TURN servers that implement the Traversal Using Relay Around NAT protocol are used. TURN servers are capable of working with different types of NATs and converting UDP stream to TCP to bypass firewall rules. However, these servers transform P2P communication into classical client/server communication. Data flow between clients is provided through the TURN server. Therefore, the end-to-end delay is increased compared to previous solutions. Besides, TURN servers are not offered free of charge and require powerful CPUs to process all incoming packets [6].

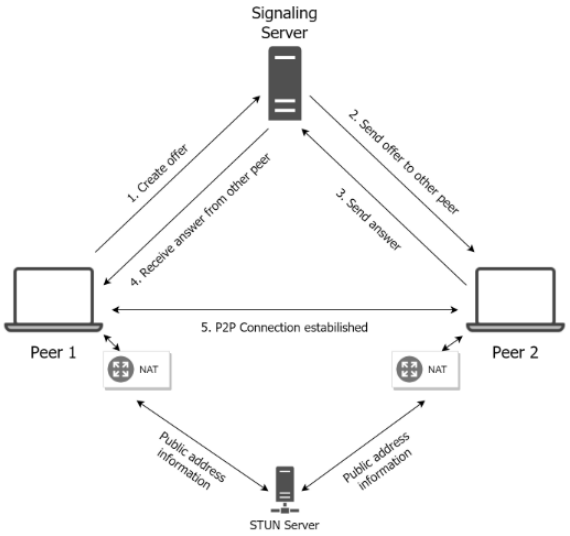

Figure 2.6 Peer discovery/negotiation through the signaling server with a STUN server [6]

Protocols An overview of the protocols WebRTC has implemented for media transfer is shown in Figure 2.7. Basically, TCP and UDP, which are 2 transport layer protocols, are used to provide data flow between the browser and the server. TCP is known as a secure protocol with its capabilities such as re-transmission of packets when packets are lost, forwarding of packets to the clients in the order they were sent, and error corrections. HTTP/HTTPS protocols work based on TCP. In addition, WebRTC prioritizes fast transmission of data over reliability. Therefore, it provides audio, video and file transfer between two browsers over UDP. With UDP, it is not controlled whether the packets are transmitted to the clients and whether they are transmitted in the correct order. WebRTC uses extra security protocols in addition to UDP for encryption, security, flow control, and congestion control. WebRTC uses DTLS to encrypt and secure unsecured UDP packets.

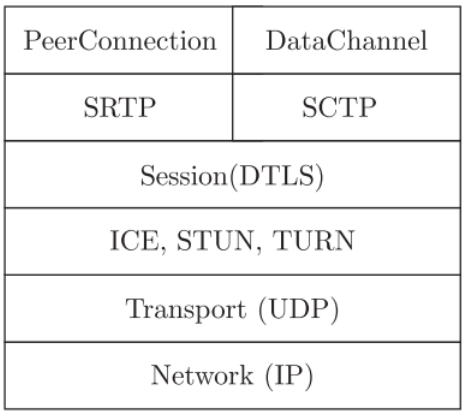

Figure 2.7 WebRTC protocol stack [7]

WebRTC uses SRTP and SCTP protocols to transmit data between clients. After the P2P communication is established, the transfer of the audio and video stream is provided over the SRTP protocol. SRTP contains all the necessary information to transport and render the video. In addition, it adds information such as timestamp and sequence number to the data, allowing the synchronization between audio and video data to be controlled. It uses the SCTP protocol on top of the UDP and DTLS protocols to transmit data over the WebRTC data channel. SCTP adds capabilities such as flow control and congestion control to the UDP protocol [7].

Application Programming Interface

WebRTC technology is a set of APIs provided over interconnected interfaces. Through these APIs, it provides access to local media devices such as web camera and microphone, communicates with other clients over the internet and sends the data received from media devices in real time, in addition to video and audio streams, it can send data in text format over a data channel. It basically uses MediaStream, PeerConnection and DataChannel interfaces to do these operations.

The MediaStream API allows managing local and remote streams. It also provides methods for recording, playing data or transferring data to another peers. It can access the camera

and microphone of the local machine through the getUserMedia() method. The stream taken with this API can be viewed using the HTML5 video tag [54].

PeerConnection is responsible for managing peer to peer communication between different browsers. Communication with the remote peer is provided via PeerConnection API. Codec handling and SDP oferring required to establish communication, sending of data packets, bandwidth management are provided by this interface.

DataChannel is a network channel that provides bidirectional transmission of arbitrary data between clients. Data channel can operate over TCP or UDP, the choice of which is entirely up to the developer. Retransmission behavior, reliability and delivery options can be determined according to the needs of the application. If these parameters are set to unordered and unreliable delivery, they show the same properties as UDP.

Topology

Topology is defined as the structure obtained by connecting the nodes in a network with each other with different links and different configurations. WebRTC leaves the creation of the topology to the software developers. They can develop topology in different ways according to the needs of the application. With WebRTC, one-to-one, one-to-many, many-to-many data transfer can be achieved. One-to-one communication is the simplest topology created to share data between clients. In addition, one-to-many or star topology is one of the topology types frequently used in the field of networking. IP TV or conferences can be given to the usage scenario of this topology. In the classical one-to-many topology, the video from one of the clients is sent to a central server and transmitted to other clients via this server. In this network, when one of the clients fails, data flow to other clients continues. However, when the Central node fails, data flow to all clients stops. In WebRTC star topology, there is data flow from one client to all other clients. In this case, when the number of clients in the system increases, the load on the source client also increases. In many-to-many topology, each client communicates with other clients. Each node sends data to other nodes in the system and receives data from all other nodes. Therefore, when new users are included in the system,

the total number of connections increases rapidly. This topology is very difficult to maintain and the traffic load is very high [7]. Multimedia Control Unit was developed to solve these problems. In that topology, each client sends its own data to the control unit, and this control unit combines all the data and sends it to other clients. Since the control unit has to process all the data, it needs powerful servers. The Selective Forwarding Unit works similarly to the MCU. However, SFU sends data directly to other clients without processing it. Therefore, it does not need CPU as much as MCU, but high bandwidth is required. Today, there are many media server software that can be used as middleware in SFU and MCU topologies. For example, SFU topology can be created with Jitsi, Janus, Mediasoup software, while MCU topology can be created with Licode and Kurento software [8]. Figure 2.8 shows the mesh, MCU and SFU based topologies.

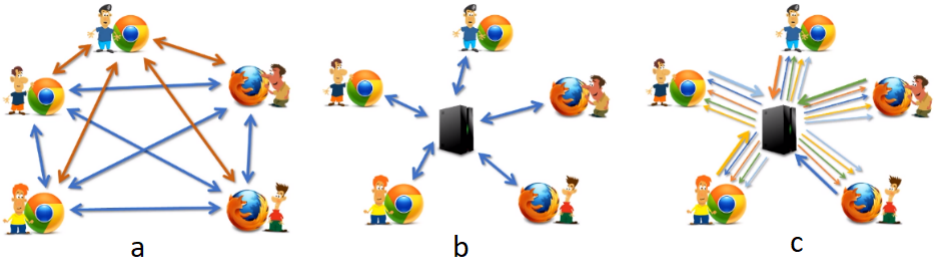

Figure 2.8 a - Mesh Topology, b - MCU, c - SFU [8]

In addition to these well-known network topology types, different topologies can be developed according to the needs of the application. For example, in the tree streaming topology, the data can be distributed from the server to the clients in the tree structure. Each client receives data from its parent and transmits it to the child node. In this topology, the nodes at the bottom get the data last, when one of the nodes fails, all nodes below this node are affected. Mesh topology can be used to solve such problems. In this system, each client connects with more than one client, and when one of its connections is failed, it continues to receive data over other connections [55].

WebRTC Based Studies

In the literature, there are many studies on examining the performance of WebRTC and transferring data through this framework. WebRTC performance is examined in environments with limited resources [10]. The aim of the study is to examine the relationship between video quality and energy efficiency when video streamed over WebRTC on Raspberry platform. Besides, another aim is to experimentally test the resource consumption of WebRTC by changing default parameters and reduce the resource consumption while maintaining the video quality. Within the scope of the study, Raspberry computer was chosen because it can encode high quality video and has low energy consumption. Adafruit INA219 DC current sensor connected to the Arduino board is used to detect the power consumption of the computer. At the end of the study, the change in energy consumption was determined by making changes on the default values used by WebRTC.It has been observed that WebRTC tries to increase the video quality by keeping the energy consumption in the 2nd plan in its default settings, and that the energy consumption can be reduced by about 30% with manually applied settings.

WebRTC is compared with RTSP one of the most used protocols for video streaming [50]. Within the scope of the study, two different android-based applications were developed. While one of these applications works over the RTSP protocol, the other application uses the WebRTC infrastructure. The main difference between these protocols is that when RTSP is used in case of video streaming is required between a group of clients, each client sends the video to the server and these streams send to other clients by server. However, when using WebRTC, each client has to send the video to other clients. When the communication establishment times of the two applications are compared, it is seen that the average value of 2.304 seconds for the RTSP protocol is 1.835 seconds for WebRTC. When the stream reception times are compared, it is seen that the average value of RTSP is 2.161 seconds and average value of WebRTC is 1.709 seconds.

In addition to performance reviews, new protocols have been developed that expand the existing capabilities of WebRTC. For example Peer-to-Peer Straightforward Protocol

developed based on HTML5 and WebRTC infrastructure is introduced [9]. This protocol contains some rules to increase the quality of service in real-time image transmission systems. Contrary to the classical client/server architecture, it is aimed to increase the system performance by using resources of the clients. Although the system is useful for small scenarios in its current structure, it offers the opportunity to be used in wider scenarios by integrating with the client/server model. The overview of the system architacture is given in Figure 2.9. The system consists of streaming server, Splitter + signaling server and clients. The video stream first comes to the streaming server. This video stream is then sent to the splitter + signaling server over the HTTP protocol. On this server, the video is split into chunks and shared between clients. Each client sends incoming data to other clients in the system via WebRTC data channel. In this way, the load on the server is reduced by using the capacities of the clients.

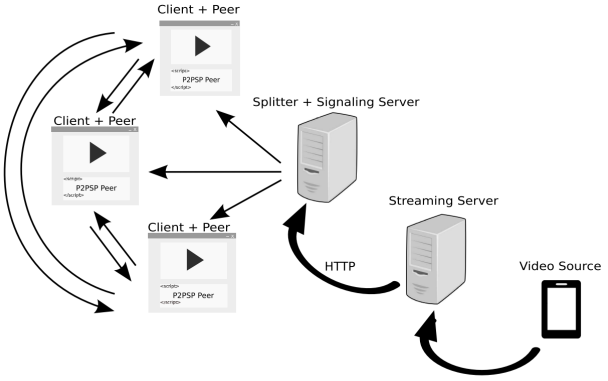

Figure 2.9 Peer-to-Peer Straightforward Protocol [9]

There are many applications developed using WebRTC datachannel in the literature. File sharing between clients is one of the most frequently researched topics. In today's methods, one of the users uploads the file to the server and the other downloads this file from the server. In this case, the full load is on the server and stronger servers and larger bandwidths are needed. A method that can share peer-to-peer files based on WebRTC and HTML5 has been developed to reduce this need [31]. In the study, websocket technology is used for the signaling server. The file received via the HTML5 file API is divided so that the size of each chunk does not exceed 16K. In addition, the size of the data is reduced by converting

it to Base64. The obtained data is transferred to other clients via WebRTC data channel. According to the WebRTC topology created within the scope of the study, a connection is established between all clients. Therefore, n * (n – 1) / 2 peer-to-peer WebRTC connections are established in the system. Each user keeps the incoming chunk in the local cache and broadcasts the data to all clients except the source. When the data transfer processes are finished, each client will have every piece of data. Then, these parts are reassembled, and the file is recreated. In this way, the user providing the data will consume resources only transferring one file and the file sharing speed will increase.

Data sharing among multiple clients using WebRTC on a virtual world is developed in [10]. The study focused on a kind of multi-layer online game. In such games, players create an avatar, control these avatars, and interact with other players. In order to achieve this, each player has to constantly send the information of the character they manage to the others. A central server collects information from all players and sends this information to other players in the system. Therefore, as the number of players increases, the load on the server also increases. To solve these problems, a method has been developed that allows player information to be transferred directly between browsers, using WebRTC, without the need for a powerful server. Signaling service is provided via a websocket. The server developed in this context applies the ring topology (Figure 2.10). The results of the study have been tested by opening 10 tabs on a web browser and creating a ring topology between these tabs. 1000 messages were sent over the first node at intervals of 500 milliseconds. Each node records the time it receives the data and sends it to the next node. In this way, it can be measured how long after the data from the starting node reaches the last node. As a result of the measurements, it was observed that the nodes close to the first node received the data in 2-3 milliseconds, while the farthest nodes received the data after 7-8 milliseconds.

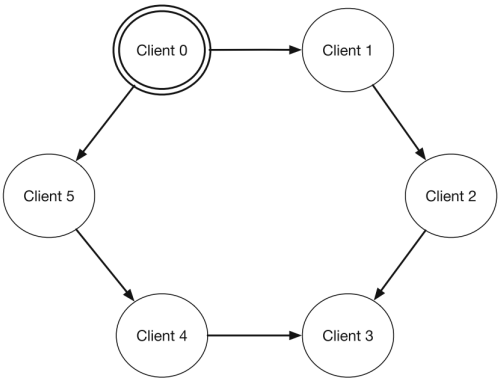

Figure 2.10 Ring Topology [10]

The transfer of DASH data over WebRTC datachannel is also studied [32]. DASH is usually transmitted over technologies such as HTTP and WebSocket. However, the rules applied by the TCP protocol used in these technologies extend the time required for the flow to start. Two Linux containers are connected to each other over a 1mbps link over WebRTC, and a test was conducted by sending 19KB chunks over the data channel. The results obtained were compared with the results obtained when streaming over HTTP using Apache server and DASH.js, it was seen that WebRTC had lower latency and higher bandwidth utilization.

There are studies on providing communication over WebRTC in emergency situations. For example in[33], a system has been developed that enables medical instructions to be taken quickly in cases such as accidents and natural hazards by using IoT, cloud computing and real-time communication technologies over the web. With this system, people who will apply first aid can get information from the emergency doctors from nearby hospitals on how to apply the first aid. The system includes applications that can work on IOS and android-based platforms that allow first aiders to talk with doctors. Video communication between doctors and emergency response teams is provided via WebRTC. In tests repeated 30 times for each step, 96 % connection success and average 0.5 second delay for video calls with 2 people, 85% connection success and 1.57 seconds delay for video calls with 6 people, connection success from control room to webcam was calculated as 93%.

The Flying Communication Server (Figure 2.11) has been developed, which enables people to share images of the disaster area, send text messages and make video communication via their mobile devices in cases where the public network is damaged due to the disaster [11]. With this server, it is aimed to support emergency teams, police, fire-fighting teams, and emergency medical services even if there is no internet connection. Besides, it is planned to provide 5 connections to the system at the same time, considering that only search and resuce team leader will access the system. Real-time messaging and video communication capability is provided via WebRTC server. For these capabilities to be used by teams, an application that works on smart phones has been developed. While the 5fps value can be maintained as the distance increases, it has been observed that the image stuttering starts to increase when the frame rate is set to more than 10 fps. When the number of simultaneous connections is examined, it is seen that the value of 10 fps is preserved up to 3 connections and it decreases to about 5 fps with 5 connections. In another experiment, 3 search and rescue team leaders were allowed to communicate via messaging and video while keep taking images from the drone. Although there are delays in the image obtained from the drone, it is thought that these values are not very important in practical use.

Figure 2.11 Flying communication server [11]

With the rapid development in drone technology and the use of this technology in different areas, many studies have been carried out on its use with WebRTC. A study was conducted combining drone and WebRTC technologies to observer urban and industrial areas [12]. With

this study, it is aimed to create a prototype to be used in other studies. Structurally, the system consists of 3 sub-systems, namely the air station, the ground station and the network that connects these two. The ground system is responsible for collecting data from the air station and sending command and control messages to control the air station. The air station is responsible for sending the images from the camera and the digital and analog data collected from sensors to the ground station in real time. WebRTC system combines the data collected from camera and other sensors into a single stream. This combined single stream data is fragmented again at the ground station and the sensor data and video data are separated. GPS service, WebRTC video service and sensor service in the system is activated when a connection request is received from the ground station. After the connection is established, the data flow is made directly between the two stations and there is no need for an external server. In the ground station, Leaflet library was used to show the location information and incoming sensor data from air station, and OpenStreetMap was used as the base map. In the prototype, the drone locations is shown with a blue triangle, while the air pollution at this location is shown with rings in different colors (Figure 2.12). To test the accuracy of the location, the values obtained from the drone compared with the known locations and difference measured. It was observed that the obtained results shifted an average of 0.2 meters from the real positions.

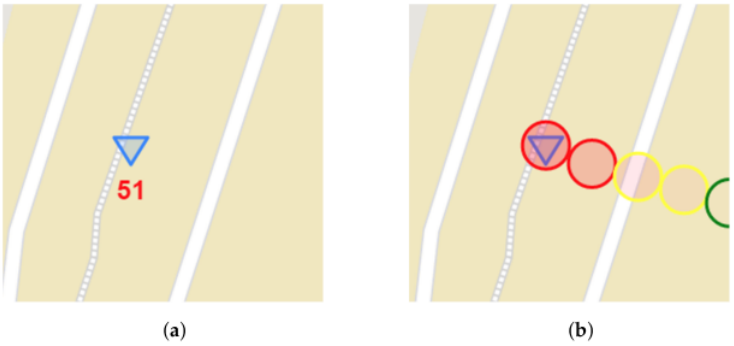

Figure 2.12 a- position marker, b - air quality marker [12]

A fliying IOT system was developed in the study [34] to monitor parking lots. The system includes a ground station and a Turnigy SK540 quad copter with integrated air quality sensor and camera controlled by Pixhawk Autopilot. Similar to this study a single board computer

placed on the drone system to combine the data from the sensor and camera and sent it to the ground system via WebRTC. Google Chrome browser is used in both the ground and air station as WebRTC sender and receiver. WebRTC server, signalling server and stun server are written in javascript and all use the node.js runtime environment. The video, telemetry and sensor data obtained from the drone are combined and sent to the ground system over the datachannel using the same WebRTC session.

2.3. Web Mapping Applications

With the rapid development of web technology and computer graphics technologies, the development of web mapping applications has also accelerated. These applications not only provided the display of geo-spatial data, but also enabled users to obtain information from this data in real time. There are many applications such as Leaflet, Cesium and Openlayers that are frequently used in this field [13]. Such applications are used in many areas such as monitoring sea areas, determining the routes of unmanned aerial vehicles [56], evaluating location information [57], displaying 3-dimensional cadastral data [58]. In this section some virtual globe applications will be introduced.

2.3.1. Leaflet

Leaflet is one of the most commonly used Javascript APIs for creating mobile-friendly interactive maps. Leaflet was developed with usability, performance and simplicity in mind. It has a well-documented user manual and a simple and readable code base. In addition, its capabilities can be extended with different plugins and can work well on most desktop and mobile platforms as stated in [36]. It has capabilities such as WMS support, video display and geometry drawing. Besides, unlike other libraries, Leaflet clients can parse SVG files themselves. Although the library is designed to work with OpenStreetMap, it can work with other map services and render raster maps in different coordinate systems [37]. Its popularity is increasing due to its open source and leightweight. In addition, Mapbox.js is embedded in Leaflet, which provides features such as simple loading and manipulation of custom tileset. It

is frequently used for mobile application development due to its small size and touch control support [59] .

In addition to mobile applications, there are also web applications developed using leaflet. An Audio Visual Web Cartography application is developed using leaflet.js [13]. Leaflet is compatible with HTML5 and CSS3, besides it is very easy to use and allows integration of multimedia data such as audio files. Within the scope of the study, a single page application was developed, and Open Street Map was used as the base map. In the next step, using the bindPopup method, an iframe with an audio file is opened when the marker is clicked. In this way, data such as language translation files and music recordings were matched with the points determined on the map. The developed system shown in Figure 2.13.

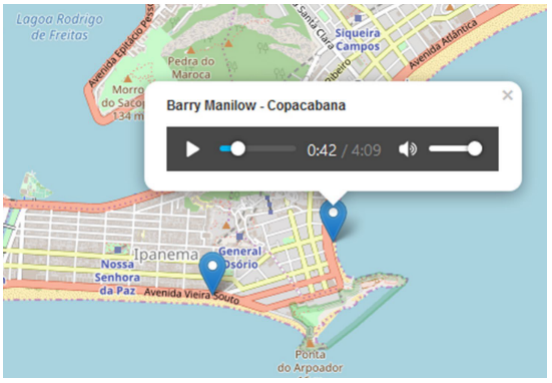

Figure 2.13 Audio Visual Web Cartography [13]

The visualization of the natural environment in the pre-industrial period on leaflet was studied in [37]. It is intended to transform old topographic maps into interactive maps using GeoJSON and Leaflet. The authors determined ground control points on old topographic images and created georeferenced maps with the QGIS software. Then, the objects on the created map are selected and classified for vectorization. Obtained objects are converted into Shape files in point, polyline and polygon formats. Shape files are converted to geojson format with the help of QGIS and visualized on Leaflet.

Studies have also been carried out on the visualization of sensor data on Leaflet. In [38] the authors studied on real-time air quality monitoring. After the data from the sensors are stored

and processed in the Cloud, they are transferred to Leaflet and visualized. In another study [12], location information is shown in real time on Leaflet with the GPS sensor placed on the drone. In addition, the data from the air quality sensors are visualized on Leaflet using geometric primitives such as triangle and square according to the level of danger.

2.3.2. Openlayers

Openlayers is an open source map software developed in Javascript language. With its high performance, it allows the management and visualization of spatial data. Besides, it supports many map service providers and has a flexible structure. With this library, servers such as Google Maps, WMS, ArcGIS Server, MapServer and Microsoft Bing can be used as map resources. Openlayers supports OLAP operations and can display the resulting data in real time without the need to refresh the page. It can render and visualize data in GeoJSON format on client-side [60].

Openlayers can be used in many fields such as weather, logistics management, tourism management, emergency transportation and sea ship management. In [39], an Openlayers-based application was developed to display the rain density and amount in real time. Real-time monitoring of rain is especially important for flood-prone areas. Obtaining information such as the amount of rain that has fallen in the last few hours and its current intensity can be effective in preventing disasters. With this study, it is aimed to detect possible floods in advance and to prevent possible casualities. This web-based application, developed using Openlayers, needs information such as the instantaneous rain density in a region and the amount of precipitation in the last 24 hours. With the developed system, rain gauge stations on Openlayers are marked with pin points, and when these stations are clicked, information about this region can be displayed. With Openlayers, sensor data can be visualized, as well as a comprehensive monitoring system can be created by using these data together with vector and raster data. The monitoring of marine areas on Openlayers using different data sources is studied [40]. Similarly, the system that enables the display of agricultural data on openlayers has been developed within [60].

2.3.3. Cesium

Cesium is a high-performance, user-friendly and open source web-based application library developed with javascript programming language. It provides several capabilities such as 3D object display and animation, map visualization using various map servers as a source, drawing of various geometry types, displaying point clouds, playing videos on terrain as shown in Figure 2.14. With the APIs it provides, it enables the development of many applications such as monitoring the drone traffic, repeating the flight on the virtual globe after the flight, creating a flight simulator, and displaying the animation of meteorological data. It is used in many areas for reasons such as being open source, having an active community, and containing extensive examples with applications such as sandcastle [14]. Although Cesium is a widely used application, there are not many studies on Cesium in the literature. Some of the studies carried out are mentioned under this section.

Figure 2.14 Cesium Capabilities [14]

Time series data is used in many fields such as engineering and economics. The importance of analyzing and visualizing this data is increasing. A method is developed for spatial location search and display of spatial-temporal remote sensing data using Cesium framework [61]. In case of natural hazards such as earthquakes, displaying information such as where the event affected and when it happened on a virtual globe is very important in terms of analyzing these data and for planning. KML format is used to visualize the information about the earthquake. In addition to this capability, a function has been added that integrates with

China's TIANDUTU Web GIS API, making bus stop, and administrative division searches via keyword similarity, and visualizing them through the Cesium entity API. Two capabilities have been developed for adding image data such as JPG and PNG directly to Cesium and for displaying remote sensing data as tiles from the server.

Displaying data in 3D on a virtual globe has an important place in thematic cartography. For many years, such data could be displayed on Google Earth based on KML. However, due to the obsolescence of this technology, Google stopped supporting this service in 2016. Although this type of data can be displayed through applications such as OpenWebGlobe developed by Swiss researchers or Earth JavaScript API developed by Klokan firm [62], Cesium can be another alternative because it has wider capabilities, rich documentation, and an active community. However, although Cesium fully supports basic geometry types from the KML format, it does not support zoom-dependent notation and context-sensitive styles. Therefore, the most widely used format is CZML. However, this format is also not supported by other popular GIS software. a QGIS plugin to generate data in CZML format is developed in [63]. Using this plugin time-dependent animated representation of the data on Cesium is possible.

Visualization of 3-dimensional and dynamic objects on the virtual sphere allowed the data to be interpreted more easily. For this reason, 3D representation has been used frequently in engineering and GIS applications. In Study [41], the authors studied the representation of 3D CityGML objects on Cesium. Although there are many browser-based virtual globe applications such as WebGL Earth, OpenWebGlobe, these applications were not competent enough at the time of the study, Cesium has been used in the scope of work due to reasons such as having very large libraries, easy to manage 3D objects, providing performance by working directly on WebGL. Classic client/server architecture was used in the study. Client applications that support HTML5 and WebGL receive HTML, CSS, JavaScript files and city data directly from this server. Since virtual spheres support global coordinate systems and cityGML files are created according to local coordinate system, these files are converted to KML/KMZ formats with the 3DCityDB Importer Exporter application and uploaded to the

server. City models were loaded and client-side buffer zone and intersection analyzes were successfully performed on Google Chrome and Opera browsers.

A sensor data management system to monitor pedelec use and the fitness level of the user. is developed in [15]. Data from Ebike-TCU sensors, data from Garmin Smarth Watch and historical data obtained via OpenWeatherData API are displayed on Cesium. SensorThings API server was used to present sensor data and requests were sent to this server using JQuery. The data received in key-value pair format is displayed on Cesium by applying client-side CZML format conversion. Capabilities such as marking important places by placing pins on the map, displaying 3D city models, monitoring the status of the bike in real time, tracking the status of the bike historically have been added to the Cesium-based application. The user interface of the developed application is shown in Figure 2.15.

Figure 2.15 Cesium- IoT [15]

With the increase in studies on global climate change and the increase in the need for situational awareness on this issue, it has become a necessity to visualize large climate data more effectively and to access this data from anywhere and anyone with an internet connection. In [42], the authors developed a Cesium based platform to meet this need and visualize time-varying/multidimensional climate data. The platform is based on the classic server/client architecture. Video is created by performing dimension reduction, color space transformation and data compression on meteorological data such as NetCDF sent to the server. The created video contains a frame for the combination of each dimension in the

meteorological data. With the study, big meteorological data are sent to Cesium-based browser application in video form and visualized with high performance on the virtual globe.

In aerial photogrammetry, the image acquisition phase is the most important phase to obtain high quality products and is affected by weather and illumination conditions. Simulating the trajectory of the aircraft in 3D can assist the planners in making decisions during the planning phase. In study [64], the flight paths of the aerial platforms and the footprints of the images taken on the ground are calculated and displayed on Cesium by using the perspective center of the image, camera rotations, image acquisition time, interior and exterior orientation parameters of the camera.

Protected areas are very important for maintaining biological diversity and protecting human life. In the study [65], accurate determination of the boundaries of these areas was studied and a 3D GIS environment was developed. Building models, high resolution DTM and vector data visualized on Cesium.

2.4. Georeferencing

Georeferencing, also known as ground registration, is the technique of processing digital images and matching the pixel coordinates on the image to the ground coordinates in Earth reference frame. Although some authors in the field of remote sensing call it rectification, in the field of photogrametry the term rectification refers to the removing of tilt effects by projecting them on a user-defined plane. The georeferencing process basically consists of two steps: 1) determining the coordinate transformation parameters that associate the digital image coordinates with the object space coordinates, 2) image array creation by assigning the brightness/DN values to the transformed image points. In the first step, x and y ground coordinates are converted to X and Y image coordinates. An Affine transformation based relationship is shown in Equation 2.1, where x and y represent the values of the points in the ground coordinate system, and X and Y represent the image coordinates. the parameters a, b, T_{x} and T_{y} are determined during the first step [16].

$$X = ax - by + T_{\mathrm{x}}$$
$$Y = ay + bx + T_{\mathrm{y}} \tag{2.1}$$

2.4.1. Indirect Georeferencing

In most cases, georeferencing of aerial photographs is based on an indirect method of obtaining the exterior/interior orientation parameters, together with measuring the ground control points and applying bundle block adjustment [66]. By using these aerial photographs with control points, operations such as finding the orientation of the image, finding the coordinates of a point in the image on the ground can be performed. In this section, the concepts of collinearity and coplanarity, which are important terminologies of photogrammetry, and the space intersection, space resection and bundle block adjustment methods based on them will be explained.

Collinearity Condition

One of the most important relationships in analytical photogrammetry is the collinearity condition. According to this condition; exposure station, any object point on the ground surface and the positions of this point on the photo are located on a line in a 3-dimensional space (Figure 2.16) [16].

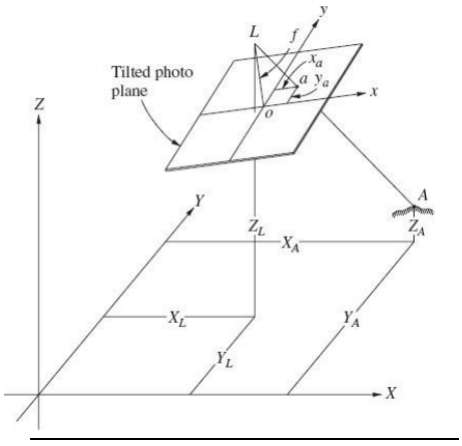

Figure 2.16 L, a, and A lie along a straight line [16]

Coplanarity Condition

According to this condition, the exposure stations of a stereopair, the positions of the common object points in the two images and the points showing their actual position on the ground are located on a common plane. As shown in Figure 2.17 exposure stations L_1 and L_2, image points a_1 and a_2, the object's real point A on the ground surface, forms a common plane.

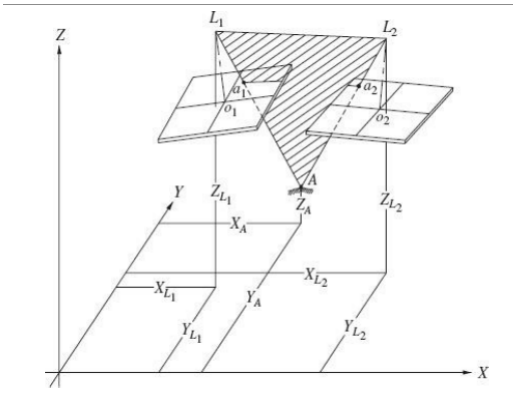

Figure 2.17 Coplanarity Condition [16]

Space Resection by Collinearity

Space resection is a method used for finding six exterior orientation parameters (ω, ϕ, κ, X_{L}, Y_{L}, Z_{L}) of a photograph. This method requires at least 3 ground control points with known XYZ object space coordinates. In addition, since the collinearity equations are linearized by Taylor's theorem, they need the initial values of the unknown orientation parameters. With this method, six exterior orientation parameters of the images can be calculated over the points with known image coordinates and ground coordinates [16].

Space Intersection by Collinearity

Exterior orientation parameters of photographs of a stereopair can be found using the space resection method. After calculating these orientation parameters, the object point coordinates

of the common features in the two photographs can be found. For both images, the lines passing through the exposure station and the image coordinate of the object are intersected on the ground surface and the ground coordinates of the object are found. For this reason, this method is called space intersection. Since the equations are made linear with Taylor's theorem, initial values are needed. Using initial approximations, corrections to these initial values can be calculated with least squares method and the solution applied iteratively until the corrections are small enough to be neglected. With the space intersection method, the ground coordinates of the points located in the overlapped areas of two images with known 6 exterior orientation parameters can be calculated [16].

Aerial Triangulation

It is the process of finding the X, Y, Z ground coordinates of a point based on photogrammetric coordinate measurements. With this method, new control points can be created for other images by using existing ground control points.This reduces the need to collect data from the field and therefore lower costs. This method is called bridging because it creates connection points between different images. It can also be used to find section corners and create a digital elevation model. One of the mostly used Aerial Triangulation methods is bundle block adjustment. This method applies space resection and space intersection methods together on image rays. After calculating the relative orientation using known camera parameters and measured coordinates, the bundles from all photos are adjusted simultaneously so that corresponding light rays intersect at positions of the pass points and control points on the ground. In the bundle block adjustment method, X,Y,Z object space coordinates and 6 orientation parameters are unknown. In addition, the x and y photo coordinates of the object points in at least two images and the X, Y, Z coordinates of the ground control points must be measured.

2.4.2. Direct Georeferencing

Coordinating aerial photographs with Direct Georeferencing method can be considered as an alternative to Aerial Triangulation method. Direct Georeferencing works with the principle of obtaining the camera exterior orientation parameters in an Earth reference system, without the need for ground control points, by using the position and orientation information of external sensors such as camera or laser scanner. It provides this by integrating the data it measured by inertial sensors and GNSS with the mapping sensor. Direct georeferencing has the advantages of mapping in real time, reducing time and labour cost, and increasing accuracy when used with Aerial Triangulation [67].

The results of the tests performed with the Applanix APX-17 UAV GNSS-Inertial system and the Sony a7R camera integrated on the MD-1000 to ortho-rectify aerial photographs with high accuracy without the need for ground control points are presented in the Study [67]. GNSS and inertial data collected with APX-15 UAV were post-processed with POSPac UAV. The obtained data was processed with Applanix Calibration and Quality Control (CalQC), which is a bundle adjustment software. As a result of the process, tie points were obtained by using exterior orientation and camera interior orientation parameters from POSPac. Then, by running the bundle adjustment algorithm with tie-points and exterior orientation parameters, IMU, camera misalignment angles, focal length and principal point offsets were found approximately. Although the study was conducted on direct georeferencing, one ground control point was used for bundle adjustment. Ortho-images were created using Inpho photogrammetric software and Exteriror orientation parameters obtained as a result of bundle adjustment. A digital surface model was created using Inpho MATHC-AT DSM software. Then, images were ortho-rectified using Inpho OrthoMaster software. Due to the processing time being approximately one hour, this study seems not suitable for real-time mapping. Besides, it has been observed that the coordinates obtained using a single ground control point are calculated with a negligible error of approximately 5 centimeters from the real points.

SFM (Structure From Motion) is one of the common methods used for direct georeferencing. Whereas traditional photogrammetric methods require the location and orientation of the camera or ground control points to triangulate the scene, SFM calculates camera orientation and scene geometry automatically by detecting matching features in overlapping images (Figure 2.18). By tracking features image to image, after the initial camera pose and its coordinate are found, it is iteratively refined using non-linear least squares minimisation. Therefore, the SFM technique requires multiple overlapping images. As a result of the operation, a 3D point cloud associated with the image-space coordinate system is obtained and these points must be associated with the real world object-space coordinate system. In most cases, this conversion can be done with a small number of ground control points. There are many studies conducted with SFM in the literature [17].

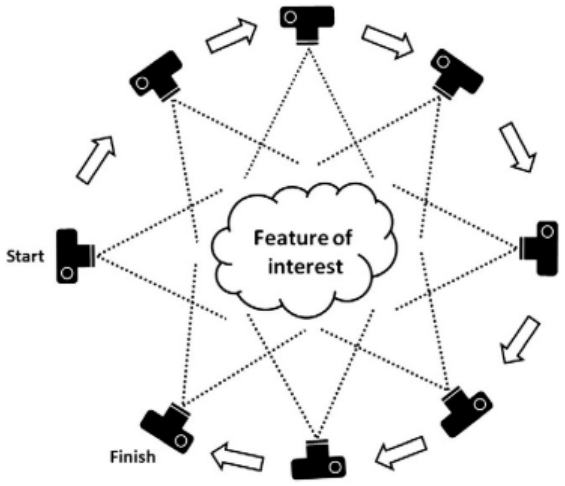

Figure 2.18 Multiple, overlapping images [17]

In [43], the authors developed a method for mapping an area where landslides occurred using UAV. In ground measurements, using the Topcon GPS Geodetic tool, 4 ground control points were obtained for use in the photographs taken and 2 for use in the tests. Then 60 aerial photographs were taken using the DJI Phantom 3 UAV. The SFM algorithm was used to obtain orthophotos from these photographs and it was observed that the algorithm could create a mesh in 30 minutes and a dense cloud in 3 hours and 29 minutes. When the model obtained is compared with the measurements made using Total Station, it was seen that the

contours are shorter and discrete. It was stated that this difference was due to the change in UAV height. In addition, while the volume was calculated as 10527.032 m^3 using 8947 points with UAV, it was calculated as 11491.708 m^3 using 260 points with the terrestrial method. It has been stated that the 964,676 m^3 volume difference is due to the differences in the number of points used. This point difference also affects the surface model. Within the scope of the study, it was not specified on which tool was used to perform the SFM algorithm.

In the Direct Georeferencing method, each image is processed by combining the X, Y, Z coordinates of the camera from GNSS and the "ω, ϕ, κ" orientation parameters of the camera taken from INS. The change of accuracy of direct georeferencing when only linear exterior orientation parameters are used by applying real time kinematic (RTK) and virtual reference system (VRS) techniques are studied [68]. Angular exterior orientation parameters are calculated with the SfM approach. A set of 182 aerial photographs were taken from a height of 85 meters and 18 ground control points are determined using static GNSS. The linear exterior orientation parameters of the images were obtained using RTK and VRS techniques on the WGS84 datum. A point cloud was produced to compare with RTK and VRS methods using Agisoft Photo Scan software. Since it would not be sufficient to compare the models with a small number of points, comparisons were made on all points in the model by using the M3C2 technique. As a first test, the accuracy of the point cloud obtained by the SFM technique was compared with the measurements made with static GNSS, and it was seen that the horizontal error was 0.022 meters and the vertical error was 0.026 meters.When the models obtained by using the linear orientation parameters calculated by using the RTK technique and the orientation parameters calculated by using AT is compared with the ground control points, the horizontal error is calculated as 0.065 meters and the vertical error is 0.061 meters. When the same processes were performed with the VRS technique, the horizontal error was calculated as 0.052 meters and the vertical error as 0.045 meters. While the most accurate results were obtained with AT, the VRS technique was found to be more successful than the RTK technique.

Although the transfer of static images and videos in real time can be provided by many applications, the process of obtaining maps from these images can generally only be done

after the flight is over. There are generally two approaches to mapping these images: photogrammetric methods that work on the texture domain and those that work on the spatial domain. Both of these approaches use some kind of image correlation and the results are used as inputs for approaches such as structure from motion. However, these methods perform intensive image analysis and consume a large amount of resources [19]. Since the processing density of the analyses increases depending on the amount of data, there is a need to reduce the number of images without reducing the information obtained. Within the scope of the study [18], the authors presented a method that enables the acquisition of a minimum number of image sets that represents the region by cropping the overlapping parts of the obtained images. It was named as Terrain Aware Image Clipping because the data set was created over the geometric intersections of the rays sent to the Earth through the image and the terrain elevation model. Because of this working principle, the algorithm needs the camera's interior orientation parameters, exterior orientation parameters for each aerial photograph, and the elevation model of the imaged region. By using these parameters, the intersection of the rays coming out of the 4 corners of the image with the elevation model is calculated approximately. By finding the horizontal centers of successive images, the ground coordinates of this point are calculated. Then, it is determined which pixels in the 2 images correspond to this point calculated on the ground surface. A line is created on the image over these pixels and this line is projected on the elevation model again. After finding the horizontal centers at the beginning and end of the two projected lines and the pixel coordinates of these points on the image, the image is cropped. Figure 2.19 shows that the images contain the information of the whole area even after cropping. The developed algorithm can be used in 2 different scenarios, Rapid Mapping and Real-time Mapping. In the Rapid Mapping scenario, the optimum data set is obtained by cropping the images from the entire data set. This algorithm works as a part of the MACS-Sar system used in disaster management. In the Real-time Mapping scenario, each image is cropped by comparing it with the previous image. In this way, while the map of the entire area is created, the amount of data sent over the network is reduced. In addition, the amount of images that need to be rendered on the map is also reduced. As a result of the tests, it has been seen that this algorithm gives much faster results than applications such as Pix4D and

Agisoft. The completion times of these algorithms are 2:40 minutes, 34:09 minutes and 20:42 minutes, respectively. It has been observed that the application gives successful results even in lightweight embedded systems, since the processing power requirement is very low. The authors stated that the developed algorithm gives much faster results compared to the classical structure from motion approach, given the position and orientation information with sufficient accuracy. However, within the scope of the study, no information was given about how the intersection with the ground surface was calculated and which technologies were used to crop and combine images.

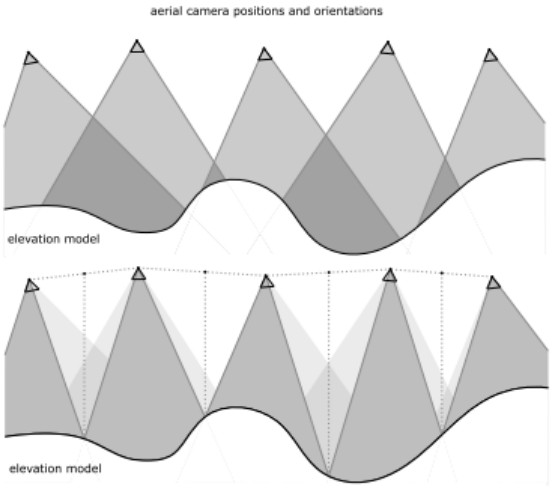

Figure 2.19 Upper: full frame, Lower: eliminated by TAC Algorithm [18]

Considering the long processing times of methods such as structure from motion, it seems that it may not possible to use them for real-time mapping. To address this problem, the authors worked on real-time mapping of images acquired during flight [19]. Within the scope of the study, the size of the image set has been reduced by using the TAC method [18] and JPEG compression algorithms. The system needs an imaging sensor, GPS module together with an IMU that can calculate real-time position and altitude. In addition, all sensors must be capable of synchronizing the position and orientation information at the time the photo was taken. The system basically crops the images taken from camera by running the TAC algorithm developed in [18]. By applying FFC on the obtained images, sensor-induced dark signal non-uniformity (FFC) and photo response noon-uniformity (PRNU) effects are

eliminated. The image is then compressed using the libjpeg library. It was observed that the number of pixels used in the mosaicking process decreased by 85% with the application of the Tac method. In addition, it was observed that the data size decreased by 98% with JPEG compression. With the two applications, the total data size has been reduced to 0.35% (Figure 2.20). Despite this significant decrease, the resulting images contain the entire scene. Projective mapping is performed on each image transmitted to the ground station via TCP. With this mapping process, the pixel coordinates of the image are converted into four-sided footprint polygon. This process is done on a single projection matrix that is calculated jointly. Although the calculation of the single footprint on each image is good enough in terms of position accuracy, the image is divided into tiles and calculations are made for each tile to increase this accuracy in the study. When the processing times in the developed method are examined, it is seen that 85% of the total time spent is jpeg compression and 11.5% is radiometric correction. In addition, it has been observed that the processing time of the TAC algorithm remains constant when the number of pixels on the image increases, while the times of other operations increase linearly. The created system has been tested on a standard aerial imaging aircraft with Wifi-based radio link and a miniaturized UAV-based camera system. The images obtained in the first scenario were sent to the ground control point 80 km away and it was seen that the image could be mapped after about 2 seconds. When the location of the created map is compared with the ground control points, it is seen that there is a difference of about 2 meters. In the second scenario, the amount of images is reduced by running the TAC algorithm on the images obtained from VTOL fixed-wing UAV after the flight. It has been seen that the created system can also be used in UAV-based systems because it needs low processing power. However, it should be taken into account that this system cannot be used in off-the-shelf UAVs due to the need for high-accuracy sensors and on-board operations.

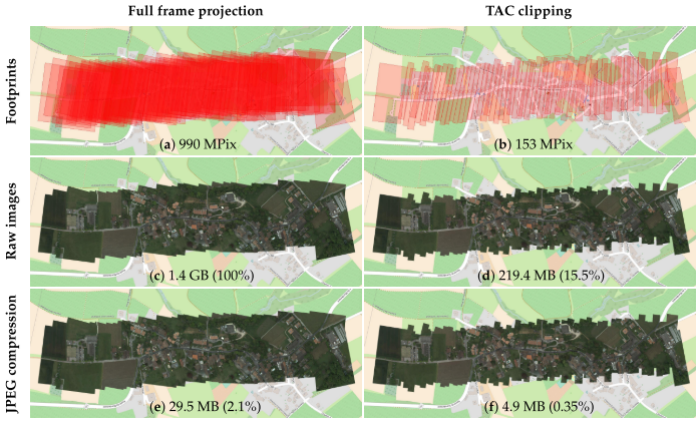

Figure 2.20 TAC algrothm and image compression results [19]

Chapter 3

METHODOLOGY

In this section, the proposed method, the steps followed and the data model created are explained in detail. The overall design of the proposed system is illustrated in Figure 3.1. The system is basically divided into 3 layers. The Data Generation Layer is responsible for producing the image with the associated telemetry data and transmitting it to the Stream Management Layer. Within the scope of the thesis, prerecorded UAV data is simulated in the Data Generation Layer and an application that transfers data in real-time from Android has been developed. Details of these studies are described in Section 3.2. Stream Management Layer is responsible to receive data from different platforms with various protocols and serving this data to Data Distribution and Visualization Layer. The Data Distribution and Visualization Layer is responsible for distributing the synchronized image from the upper layer among Cesium-based clients and displaying it on the virtual globe by real-time georeferencing. Image and telemetry data must be produced in accordance with the data model described in 3.1. The data flow between the Content Server and the first client in the system is provided over the Websocket protocol. After the connection is established, the first client acts as a data source to other clients participating in the system and distributes the data over the WebRTC protocol.

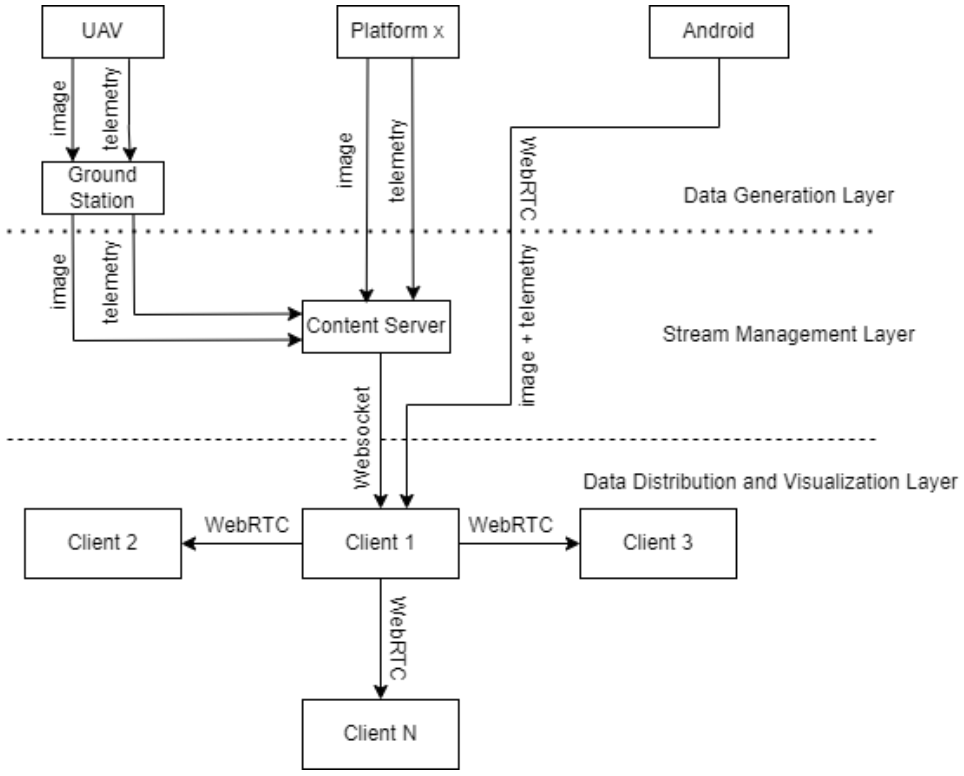

Figure 3.1 System Design

3.1. Data Model

In this section, the data format used in the transmission of image and telemetry data within the system is introduced. The system basically needs an aerial image, the yaw, pitch, roll, latitude, longitude and height information at the time this aerial photograph was taken. In order for image and telemetry data to be transmitted to client applications at the same time, the data must be delivered over the same channel. Therefore, there is a need to represent the image in text format. Both to meet this need and because Cesium's Entity classes support images in Base64 format, thus images are transferred in this format within the system. An example of the proposed data model is shown in Listing 3.1. Besides, yaw, pitch, roll, longitude and latitude data are represented in degrees and height is represented in meters. The units of these parameters are summarized in Table 3.1.

```
{
  "image" : "image/jpeg;charset=utf-8;base64..",
  "height" : 30.9,
  "lat"    : 39.914687,
  "lon"    : 32.775317,
  "pitch"  : 2.2,
  "roll"   : 3.0,
  "yaw"    : 159.5,
 }
```

Listing 3.1 Data Model example

Parameter	Unit
Aerial Image	Base64 String
Latitude	Degree
Longitude	Degree
Yaw	Degree
Pitch	Degree
Roll	Degree
Height	Meter

Table 3.1 Parameters and Units in Data Model

3.2. Data Generation Layer

In this layer, any platform that can take aerial photographs and has orientation and position information can take place. However, these platforms that provide the data are expected to synchronize the image and telemetry data and transmit it to the stream management layer. In addition, the synchronization of the data can also be done on the stream management layer, or synchronized data can be transferred directly to the Data Distribution and Visualization Layer. Data transfer from these platforms can be provided through various protocols.

3.2.1. UAV

Mavlink is a lightweight protocol that enables the communication of drone and ground stations. Video and telemetry data can be obtained over this protocol. By using the Mavlink

Java SDK, it is possible to provide data flow between the drone and the content server with new developments to the content server. In addition, there are studies on data transfer from drones with WebRTC using the onboard computer placed on the drone. Synchronized data obtained from these protocols can feed the system through the content server. In addition, it is possible to establish a connection directly between the drone and the web client by skipping the step of sending the data to the content server with WebRTC-based studies. Within the scope of this thesis, the data acquisition protocols from drone are not focused on and real-time data flow is simulated using pre-recorded drone and telemetry data. In this study telemetry and video data are taken from DJI Mavic Pro drone. The flowchart of creating this simulation is given in Figure 3.2. Since the georeferencing process is performed on each image frame in the proposed system, the video data must be divided into frames. The Command Line command used to extract the frames given in Listing 3.2.

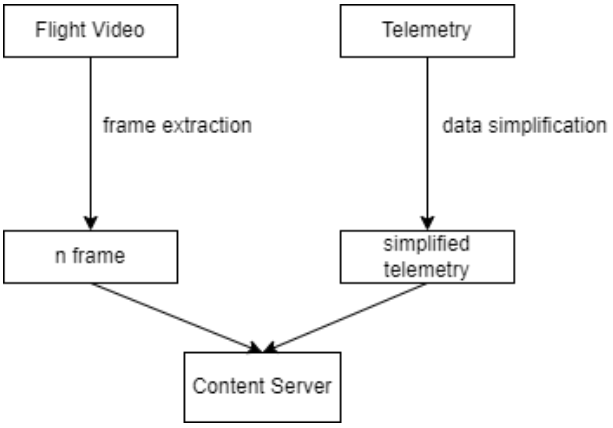

Figure 3.2 UAV Data Preparation

```
ffmpeg -i flight.mp4 -vf 'fps=25' frames/c01_%04d.jpeg
```

Listing 3.2 Ffmepg Frame Exraction

Telemetry systems provide automatic measurement via sensors and wireless transmission of these measurements from remote sources. These systems are not only in drones but also in many applications used in IOT [69]. As described in the Data Model section, telemetry data is expected to include latitude, longitude, height, yaw, pitch, roll and update time information.

In addition to these, there are many different data such as magnetometer reading, gyroscope reading, battery, flight status and velocity [70]. In order to reduce the amount of data and speed up telemetry searches, the necessary information in this data in CSV format was taken and saved in JSON format. The code block, which was developed with the Java programming language and allows only the needed columns in the CSV, is given in Listing 3.3. An example of telemetry data obtained as a result of the simplification process is given in Listing 3.4.

```
SparkSession sparkSession = SparkSession.builder().config("spark.master", "local")
.getOrCreate();

Dataset<Row> dataset = sparkSession.read.format("csv").option("header", "true")
.option("mode","PERMISSIVE").load(telemetryPath);

Column[] columns = {col("height"),col("lat"),col("lon"),col("pithc")
,col("roll")
,col("yaw"),col("updateTime")};

dataset = dataset.withColumnRenamed("OSD.height","height")
                 .withColumnRenamed("OSD.latitude","lat")
                 .withColumnRenamed("OSD.longitude","lon")
                 .withColumnRenamed("OSD.pitch","pitch")
                 .withColumnRenamed("OSD.roll","roll")
                 .withColumnRenamed("OSD.yaw","yaw")
                 .withColumnRenamed("CUSTOM.updateTime","updateTime")
                 .withColumnRenamed("OSD.height","height")
                 .withColumnRenamed("OSD.height","height");
```

Listing 3.3 Simpifying Telemetry Data

```
{
  "height" : 30.9,
  "lat"    : 39.914687,
  "lon"    : 32.775317,
  "pitch"  : 2.2,
  "roll"   : 3.0,
  "yaw"    : 159.5,
  "ts"     : 0.0,
}
```

Listing 3.4 Simplified Telemetry

3.2.2. Android

Besides drones, phones can also be used to obtain images with telemetry data. In this context, an application has been developed that obtains location and orientation information from the sensors and images from the camera on an Android phone. Location information on Android can be obtained via the GPS receiver embedded in the system. GPS data can be read through Location and LocationCallback interfaces. The declination angle can be calculated instantaneously over the GeomagneticField interface. The method developed to access this information is given in Listing 3.5.

```
1   private void createLocationCallback() {
2        locationCallback = new LocationCallback() {
3        @Override
4        public void onLocationResult(LocationResult result) {
5            super.onLocationResult(result);
6            location = result.getLastLocation();
7            geomagneticField = new GeomagneticField(
8                    Double.valueOf(location.getLatitude()).floatValue(),
9                    Double.valueOf(location.getLongitude()).floatValue(),
10                   Double.valueOf(location.getAltitude()).floatValue(),
11                   System.currentTimeMillis()
12           );
13       }
14       };
15    }
```

Listing 3.5 Getting Location from Android

Orientation information on Android can be obtained through the Sensor API and ArCore API, the details are given in the subsections. The application basically transforms the image and position information it receives over the Android system into the structure described in the data model section and sends it to the client applications. This developed application provides data transfer via WebRTC, thus eliminating the dependency on the content server. If additional controls are to be made on the content server, it is also possible to transfer data between the Android application and the content server over protocols such as websocket.

Sensor API The basic flow diagram of the application developed in this context is similar to the structure described in the previous section. Location information from the GPS sensor provided by the Android system, orientation information from the sensors and images from the camera are obtained, converted into a data model and sent to the web client via WebRTC. The structure of the proposed system is illustrated in Figure 3.3.

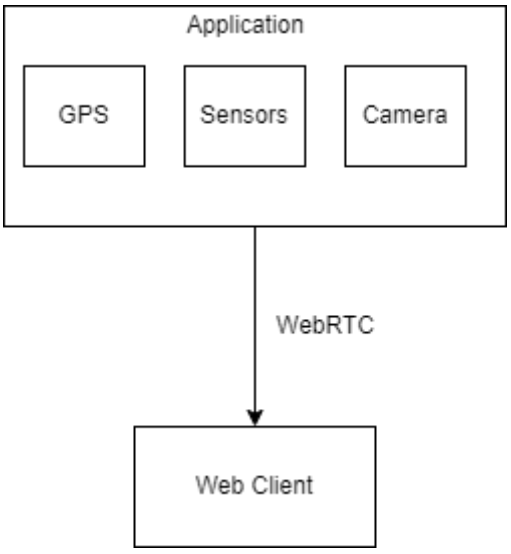

Figure 3.3 Android App - Sensor API

Many data such as orientation, rotation vector, step counter can be retrieved via the Android Sensor API. Within the scope of the thesis, the orientation of the phone was calculated using the Sensor.TYPE_ROTATION_VECTOR. However, the orientation information received via Android is represented in a different coordinate system than the coordinate system used in Cesium. By default, Cesium uses the ENU coordinate system shown in Figure 3.4. In this right-handed coordinate system, the x axis points East, the y axis points North, and the z axis points upward. Besides, as stated in [14], Cesium has determined yaw angle as rotation around -z axis, pitch angle as rotation around y axis and roll angle as rotation around x axis. Therefore, the yaw, pitch, roll angles represent rotations in the down, South, and East directions, respectively (Table 3.2). In order to switch from the ENU coordinate system to the coordinate system where the heading, pitch, roll angles are represented in Cesium, 180 degrees rotation must be applied in the x axis.

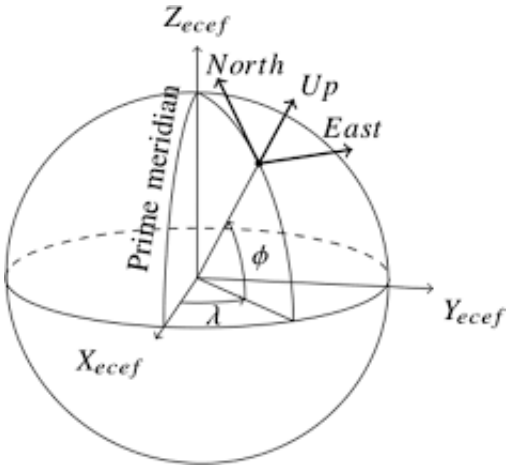

Figure 3.4 East North Up [20]

Cesium	
Yaw	Down
Pitch	South
Roll	East

Table 3.2 Cesium Directions

Like Cesium, Android uses the ENU coordinate system by default. Android device orientation is illustrated in Figure 3.5. Besides, android determined yaw angle as rotation around z axis, pitch angle as rotation around x axis and roll angle as rotation around y axis (Table 3.3). Therefore, the yaw, pitch, roll angles represent rotations in the Up, East, and North directions, respectively. In addition to these, as stated in [21], while the angles in the counter-clockwise direction should be positive in right-handed coordinate systems, the roll angles obtained from Android sensors are calculated as positive in the clockwise direction. Since this value is calculated counter-clockwise in Cesium, the negative of this angle is taken in the calculations. Besides, since the back camera of the phone is used within the scope of the thesis, the reverse of the z axis must be taken. In this context, the yaw value of the camera is calculated as the negative of the yaw value from the sensor. Besides, with the inverse of the z axis, the Android coordinate system turns into a left-handed coordinate system.The directions that yaw pitch roll angles refer to after taking the inverse of the z axis are given in Table 3.4. In the left-handed Android coordinate system, the rotations are -yaw in the Down direction, pitch in the East direction, and -roll in the North direction. In order to

match the sensor data obtained from Android with the yaw, pitch, roll (Down, South, East) directions used in Cesium, 90 degrees clokwise rotation from the z axis should be applied. The reference directions of yaw, pitch, roll angles after rotation are given in Table 3.5. In the left handed coordinate system obtained with this rotation, the rotation angles are -yaw in the Down direction, -roll in the East direction and pitch in the South direction. Since the direction of positive rotations is different in left-handed and right-handed coordinate systems, rotations are reversed when matching angles obtained from Android with Cesium axes. The relationship between the rotations obtained from Android sensors and Cesium as a result of these processes is given in Equation 3.1.

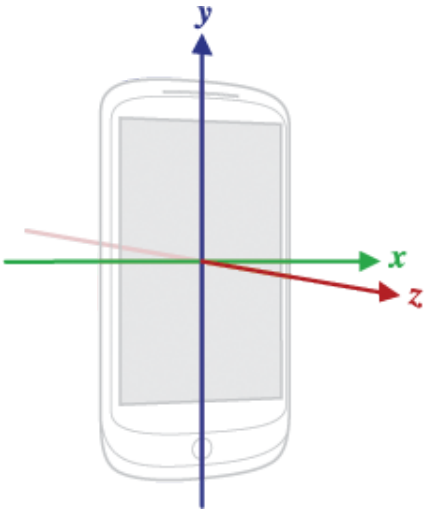

Figure 3.5 Android Device Axes [21]

Yaw	Up
Pitch	East
-Roll	North

Table 3.3 Android Device Directions

-Yaw	Down
Pitch	East
-Roll	North

Table 3.4 Android Camera Directions

-Yaw	Down
Pitch	South
-Roll	East

Table 3.5 Android Camera Directions After Rotation

$$
\begin{aligned}
Cesium_{\text{yaw}} &= Android_{\text{yaw}} \\
Cesium_{\text{pitch}} &= -Android_{\text{pitch}} \\
Cesium_{\text{roll}} &= Android_{\text{roll}}
\end{aligned}
\tag{3.1}
$$

As given in Table 3.6 Cesium initial yaw angle points to East. Besides Android sensor references magnetic north. Therefore, declination value must be included in the calculations to convert Magnetic North to True North. The method given in Listing 6 has been developed to convert the yaw angle from Android to True North using the declination angle and to make the angle values suitable for the degree range used in Cesium.

```
private double normalizeYawAngle(double value) {
   double retValue = 0;
    if (value <= 0) {
        retValue = 360 + value;
    } else {
        retValue = value;
    }
    retValue = retValue  + geoField.getDeclination() - 90;
    retValue = (retValue < 0) ? 360 + retValue : retValue;
    return retValue
}
```

Listing 3.6 Android Yaw to Cesium Yaw

East-South	0 - 90
South-West	90 - 180
West - North	180 - 270
North -East	270 - 360

Table 3.6 Cesium Yaw Intervals

After obtaining the location via GPS and orientation via Sensor API, a snapshot is taken from the camera. It will be sufficient to call setPreviewCallback() function via the Android

Camera API to get the image from the camera(Listing 3.7). The data model obtained as a result of these processes is transmitted to the content server or directly to the web clients, and the photos taken from the phone are georeferenced in real time and displayed on Cesium.

```
Camera.PreviewCallback previewCallback = new Camera.PreviewCallback() {
    public void onPreviewFrame(byte[] data, Camera camera) {
        try {
            capturedImage = new CapturedImage(data);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }};
mCamera.setPreviewCallback(previewCallback);
```

Listing 3.7 Android Image Capture

ArCore API

ARCore is a library developed by Google for developing augmented reality applications. ARCore enables the pose of the phone to be determined on the world by its motion sensing capability. In the developed application, position information is obtained as described in section 3.2.2, while image and orientation information is obtained through ARCore. The structure of the proposed system is illustrated in Figure 3.6. The formula required for converting pose to yaw, pitch and roll angles is given in Equation 3.2, 3.3 and 3.4. Since axis transformations are the same as described in the Sensor API section, they will not be mentioned under this section.

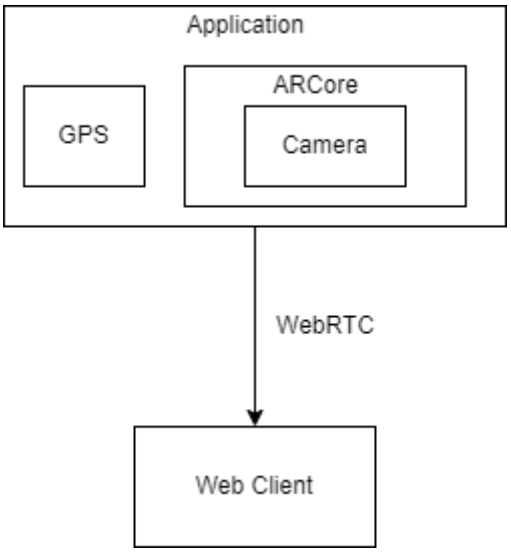

Figure 3.6 App with ARCore API

$$yaw = \arctan2(2 \times q.y \times q.w - 2 \times q.x \times q.z, 1 - 2 \times q.y \times q.y - 2 \times q.z \times q.z) \quad (3.2)$$

$$pitch = \arcsin(2 \times q.x \times q.y + q.z \times q.w) \quad (3.3)$$

$$roll = \arctan2(2 \times q.x \times q.w - 2 \times q.y \times q.z, 1 - 2 \times q.x \times q.x - 2 \times q.z \times q.z) \quad (3.4)$$

ARCore API provides interfaces for capturing images from the camera. These captured images need to be converted to Base64 to fit the Data Model. The steps followed for the conversion are given in Figure 3.7.

Figure 3.7 Image to Base64 Conversion

The data model obtained as a result of these processes is transmitted to the content server or directly to the web clients, and the photos taken from the phone are georeferenced in real time and displayed on Cesium.

3.3. Stream Management Layer

Content Server acts as a data source for client applications in the system. It is responsible for providing synchronized aerial photography and telemetry information over the Websocket protocol. Content server has the ability to be fed from many data providers. By connecting with ground stations, the data these stations receive from UAVs over protocols such as mavlink, the data transferred by connecting directly to the UAV, or the data that can be received over different platforms can be used by Content server as data source.Synchronized data can be sent directly to the content server, as well as the synchronization process can be done on the content server with additional improvements to be made. As mentioned in section 3.2.1, within the scope of this thesis, the data acquisition protocols from drone are not focused on and real-time data flow is simulated using prerecorded drone and telemetry data. Content server reads this simulated data from threads running in the background and sends it to the first client that joins the system via websocket. Content server can serve multiple flight videos at the same time. However, with the proposed architecture, Content Server only serves one sample of the videos, regardless of how many users there are in the system. If there are multiple clients requesting the same video, the data is shared directly between client applications. So Content Server creates only one websocket connection for each flight. The design of the content sever is illustrated in Figure 3.8.

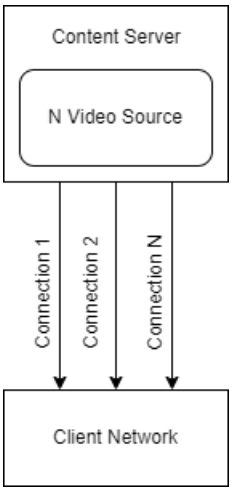

Figure 3.8 Content Server

Content Server also needs to generate data in accordance with the structure described in the Data Model section. Frames with .jpeg extension created with ffmpeg need to be converted to Base64. The code block providing this functionality is given in Listing 3.8.

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
try{
    BufferedImage image = ImageIO.read(new File(pathToFrame));
    ImageIO.write(image, "JPG", out);
}catch(Exception e){
    e.printStackTrace();
}
byte[] bytes = out.toByteArray();
String base64String = Base64.getEncoder().encodeToString(bytes);
```

Listing 3.8 JPEG to Base64

In the proposed system, websocket is used not only to transfer images and telemetry, but also to provide orchestration between WebRTC clients, details are described in section 3.4.1.

3.4. Data Distribution and Visualization Layer

In this section, the steps followed to distribute the synchronized image among client applications and visualize it on the virtual globe are explained in detail. This section will be examined under the sections of Data Distribution and Data Visualization.

3.4.1. Data Distribution

Data transfer between components is illustrated in Figure 3.9. Synchronized video and telemetry data from drones or other platforms are included in the system via the content server using websocket. The same data is distributed between client applications over WebRTC. In addition, the signaling server, which provides websocket service to orchestrate WebRTC clients, is also included in the system.

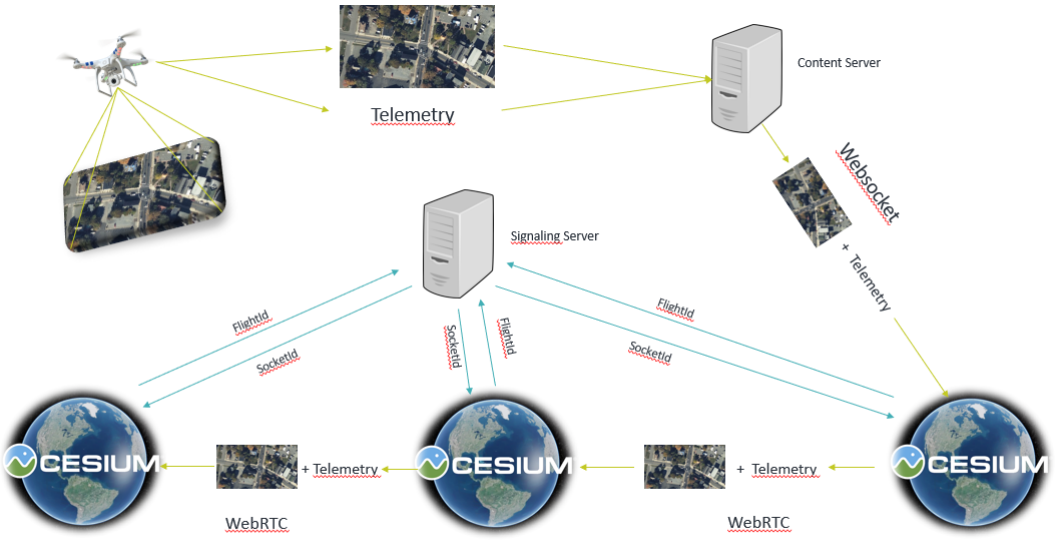

Figure 3.9 Data Transfer Between Components

In the proposed system, it is aimed to reduce the load on the server by distributing video and telemetry data among client applications. WebRTC, which allows peer to peer communication, was used to meet this need. Video and audio transmission is possible via WebRTC media stream API. However, telemetry data is in text format and it is not possible to share video and text files synchronously with WebRTC's current capabilities. For this reason, within the scope of the thesis, it was decided to send the image and telemetry data in text format over the same channel. It is possible to realize this need with the datachannel API offered by WebRTC, and since Cesium is a javascript-based mapping application, it can be integrated with WebRTC.

A signaling server is needed to establish and manage a network between WebRTC-based applications. This server keeps the records of active users in the system, which flight videos these users requested, and enables dynamic connections between applications. WebRTC does not provide a standard for how to create a signaling server. However, within the scope of the thesis, this server was developed based on Websocket. WebRTC topology can be designed according to the needs of the application. An example scenario is given in Figure 3.10. In this structure, the signaling server forwards the first client requesting the flight1 video directly to the content server. This client starts receiving video frames by establishing

a websocket connection with the content server. When a second client requesting the same video included in the system, the signaling server triggers the second client to establish a WebRTC connection with the first client. The two clients send the information required for creating WebRTC connection to each other over the signaling server, and the WebRTC connection is established and the video frames are started to be received over this channel. If the first user leaves the system, the signaling server redirects the second user to the content server, allowing the websocket connection to be established. In this way, the data flow is ensured continuously.

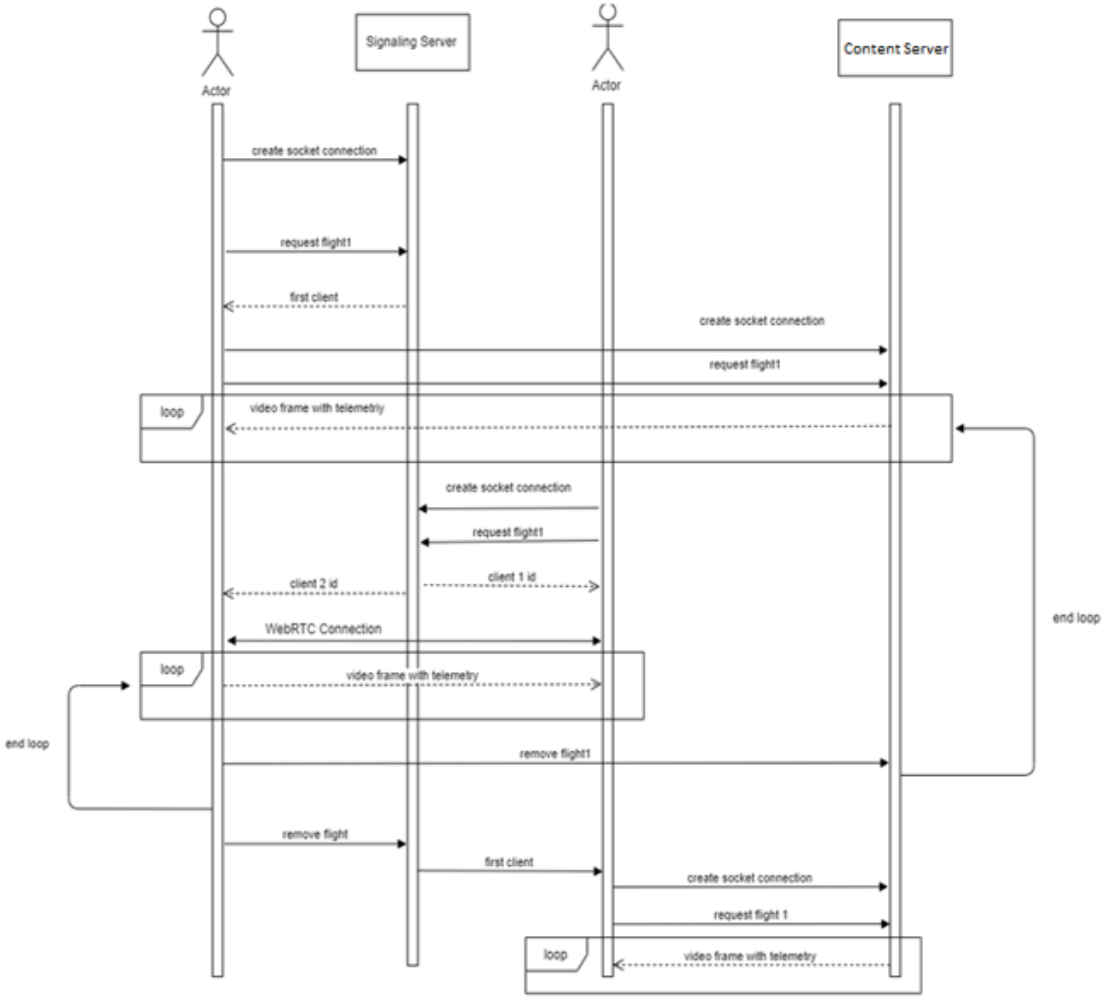

Figure 3.10 WebRTC squence diagram

With the proposed topology, the video received once from the content server is distributed

among other clients in a peer-to-peer approach. In case there are 2 different videos and 3 active clients in the system, the Signaling server directs client 1 to the content server to receive the flight1 video. Similarly, it directs client2 to the content server to receive the flight 2 video. Client1 transmits flight1 data to client 2 and client 3 directly via WebRTC. On the other hand, client2 transfers flight2 video to client 1 and client3 via WebRTC. In this way, while 2 video instances are taken directly from the server, each of 3 client applications can display 2 videos.

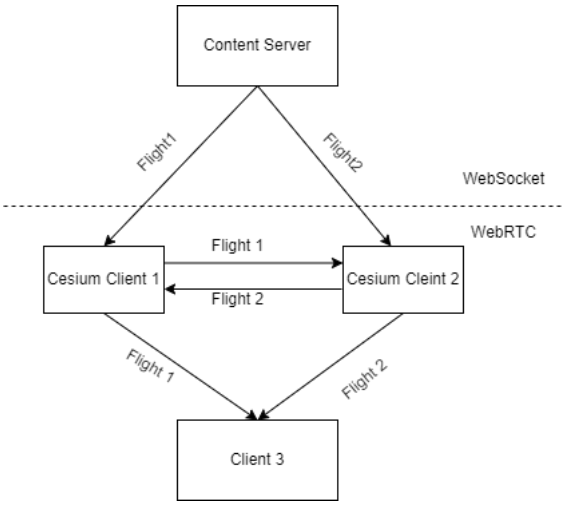

Figure 3.11 WebRTC Topology

In addition to video and telemetry data, the data required to establish a WebRTC connection are transferred between client applications. The data is provided through the signaling server until the connection between the two clients is established, and after the connection is established, data is provided peer to peer. When clients first join the system, they register to the signaling server. When the signaling server triggers two clients to establish a WebRTC connection, the first client creates an SDP offer containing the session description information and sends it to the other client over the signaling server. The client receiving this offer creates its own session description and sends an Answer SDP to the first client over the signaling server. After this step, clients request ICE candidates from browser. Since the browser knows the local IP, it sends this information to the client as ICE candidate. The client sends the incoming ICE candidate information to the other client via the signaling

server. Similarly, the second client receives an ICE candidate from the browser and sends it to the first client via the signaling server. If two clients are in the same local network, ICE candidates from the browser are sufficient for communication. If the clients are running behind NAT or firewall, ICE candidates from the browser will not be enough. In this case, the clients obtain their network information through the STUN servers. STUN server ensures connection if client applications are running behind full cone, restricted cone and port restricted cone NAT. However, if the clients are running behind symmetric NAT, then STUN is not enough to establish a connection and it is necessary to use TURN servers. In summary, client applications request ICE candidates from browser, STUN and TURN servers, respectively, to establish a connection. After the two clients share network information with each other, the data flow is provided peer to peer. The diagram illustrating these steps is given in Figure 3.12.

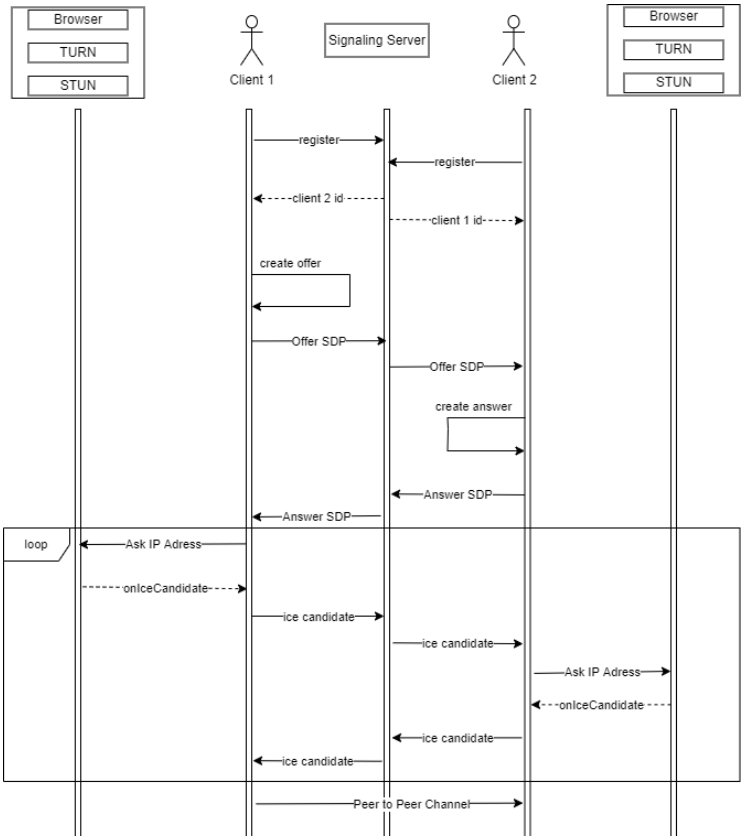

Figure 3.12 WebRTC Peer SDP/ICE Transfer

3.4.2. Data Visualization

Within the scope of the thesis, Cesium.js was used as a data visualization tool due to reasons such as being javascript-based, capable of 3D viewing, facilitating mathematical calculations with the interfaces it offers, and providing infrastructure for writing custom shader programs. After parsing each image-telemetry pair coming from content server via websocket or from other Cesium instances via WebRTC, images are georeferenced and visualized on virtual globe.

To georeference the images, the look vectors of the 4 corner points were calculated using the FOV width and FOV height information of the camera from which the image was taken, and then the intersection points with the ground were calculated. The camLook vector is accepted as [0, 0 , -1] since the camera used in the thesis is directly downward. The camlook vector was first rotated with respect to the frustum angles and thus the vectors of the corner points in the camera coordinate system were obtained. The rotation angles applied to calculate the corner points in the camera coordinate system are given in Table 3.7. Images from the drone used in the thesis are obtained with 16:9 aspect ratio. Besides, as stated in [71], the diagonal FOV angle of this drone camera is 78.8 degrees. Therefore, vertical and horizontal FOV angles should be calculated from diagonal FOV. The formulas used in these calculations are given in Equation 3.5. F_{H} represents horizontal FOV, F_{V} represents vertical FOV, F_{D} reprensents diagonal FOV, S_{H} represent horizontal size, S_{V} represents vertical size and S_{D} represents diagonal size.

lowerLeftCorner	fovWidth / 2, fovHeight / 2
loweRightCorner	-fovWidth / 2, fovHeight / 2
upperRightCorner	-fovWidth / 2, -fovHeight / 2
upperLeftCorner	fovWidth / 2, -fovHeight / 2

Table 3.7 Image Corner Points

$$
\begin{aligned}
F_{\mathrm{H}} &= \arctan((\tan(F_{\mathrm{D}}/2) \times (S_{\mathrm{H}}/S_{\mathrm{D}})) \times 2 \\
F_{\mathrm{V}} &= \arctan((\tan(F_{\mathrm{D}}/2) \times (S_{\mathrm{V}}/S_{\mathrm{D}})) \times 2
\end{aligned}
\tag{3.5}
$$

where;

$$S_{\mathrm{D}} = \sqrt{S_{\mathrm{H}}^2 + S_{\mathrm{V}}^2}$$

These corner points are illustrated in Figure 3.13. The code block developed to obtain the rotation matrix from these angles is given in Listing 3.9.

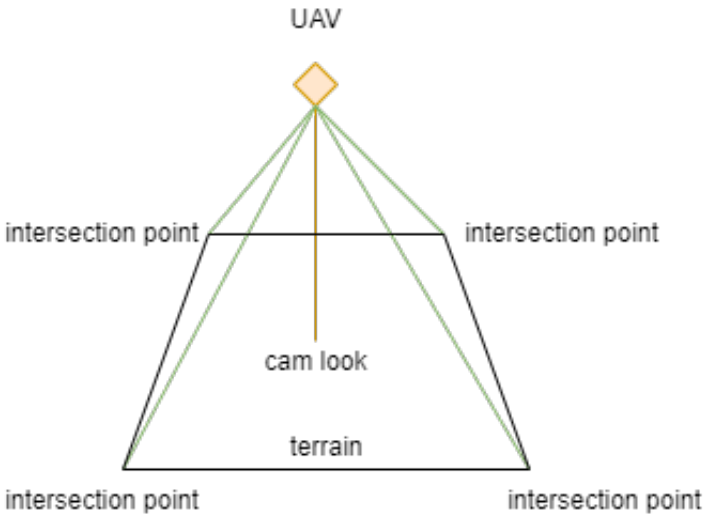

Figure 3.13 UAV Terrain Intersection

```
function getCameraFrame(angleX, angleY) {
    angleX = degrees_to_radians(angleX);
    angleY = degrees_to_radians(angleY);
    var R = Cesium.Matrix3.multiply(Cesium.Matrix3.fromRotationX(angleY,
    new Cesium.Matrix3()),
    Cesium.Matrix3.fromRotationY(angleX, new Cesium.Matrix3()),
    new Cesium.Matrix3());
}
```

Listing 3.9 Frustum Rotation

After the camLook vector is rotated relative to the frustum points, drone orientation must also be taken into account. Generally, drones produce telemetry data in the NED coordinate system. In this coordinate system illustrated in Figure 3.14, the yaw angle represents the rotation around the z axis, the pitch angle represents the rotation around the y axis, and the roll angle represents the rotation around the x axis. The directions of yaw, pitch, roll angles in

Cesium and NED coordinate system are given in Table 3.8 and Table 3.9. When these tables are examined, it is seen that the North direction is referenced as 0 in the NED coordinate system, but the East direction is referenced as 0 in Cesium. For this reason, 90 degrees rotation in the z axis is required for the axes of the two systems to match. The process of calculating the rotation matrix using the heading, pitch, roll and position information of the drone is given in Listing 3.10. The rotation of the coordinate system to ECEF is done in the headingPitchRollQuaternion method. Therefore, as a result of the operations performed in Listing 3.10, the rotation in the ECEF coordinate system is obtained.

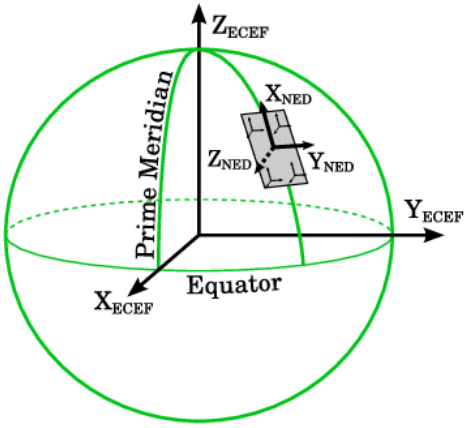

Figure 3.14 North-East-Down [22]

Cesium	
Yaw	Down
Pitch	South
Roll	East

Table 3.8 Cesium Directions

UAV	
Yaw	Down
Pitch	East
Roll	North

Table 3.9 UAV Directions

```
var hpr = new Cesium.HeadingPitchRoll(heading, pitch, roll);
var orientation = Cesium.Transforms.headingPitchRollQuaternion(position, hpr);
var rotationCameraToEarth = Cesium.Matrix3.fromQuaternion(orientation);
```

Listing 3.10 Rotation Camera to Earth

The formula used to find the coordinates of the vector drawn by the camera towards the ground in the world coordinate system is given in Equation 3.6. V_{look} represents the corner point of the image in the World Coordinate System, R_{B}^{E} represents the body-to-earth rotation matrix, R_{C}^{B} represents the camera-to-body rotation matrix, $R_{\text{x}}(\alpha)$ represents the rotation matrix based on the FOV width, $R_{\text{y}}(\beta)$ represents the rotation matrix based on the FOV height, and V_{cam} represents the vector drawn from the camera center to the ground.

$$V_{\text{look}} = R_{\text{B}}^{E} \times R_{\text{C}}^{B} \times R_{\text{x}}(\alpha) \times R_{\text{y}}(\beta) \times V_{\text{cam}} \tag{3.6}$$

After calculating the Look Vector, the code block that calculates the intersection of this vector with the terrain is given in Listing 3.11.

```
var lookRay = new Cesium.Ray(position, lookVec);
var intersection = globe.pick(lookRay, scene);
```

Listing 3.11 Terrain Intersection Calculation

After these calculations were made, the aerial photograph and the footprint of this photograph on the ground were displayed on the terrain over Cesium Entity classes. Images can be added to primitives by using the ImageMaterialProperty capability provided by Cesium. In addition, the positions of the primitives can be changed dynamically by using the CallbackProperty. The code developed using these capabilities is included in Listing 3.12. An entity is created for each flight video and its attributes are updated as new data comes in.

```
viewer.entities.add({
    polygon : {
        hierarchy : new CallbackProperty(function(){
            return getBasePolygonPoints();
        },false),
        material : new ImageMAterialProperty({
            image : "base64String"
        }),
     }
    });
}
```

Listing 3.12 Cesium Entity

3.5. Test Scenarios

To test the performance of WebRTC, 27 test steps were executed under the test scenarios stated below. It is aimed to measure the min, max and mean values of the delays in the packets and load on the content server, with the changing number of users, when the simulated data produced in section 3.2.1 is distributed between client applications over Websocket and WebRTC.

- Distribution of a single video source to variable number of users over Websocket.
- Distribution of a single video source to variable number of users over WebRTC.
- Measuring the effect of the network topology applied when data is distributed over WebRTC to the same number of users
- Distribution of multiple videos produced with 25 fps to all users in the system via Websocket.
- Distribution of multiple videos produced with 25 fps to all users in the system via WebRTC.
- Distribution of multiple videos produced with 15 fps to all users in the system via Websocket.

- Distribution of multiple videos produced with 15 fps to all users in the system via WebRTC.

The test steps listed below were followed to test the Direct georeferencing method applied to the simulated data.

- Measuring the distance between the selected reference points on the coordinated image and Google Map by using the Qgis application
- Measuring the distance between the selected reference points on the coordinated image and Bing Map by using the Qgis application
- Measuring the distance between the selected reference points on the coordinated image and Openstreet Map by using the Qgis application

Chapter 4

APPLICATION and RESULTS

In this section, the components of the software developed, the development environments and libraries used, and the findings obtained as a result of the tests are explained in detail.

4.1. Implementation of Applications

In this section, detailed information is given about the libraries, programming languages and development environments used while developing content server, signaling server, map client and android application.

4.1.1. Content Server

In the proposed system, the content server acts as a data source for client applications. It is responsible for serving the data it receives from the data generation layer to client applications. Content Server has been developed with Java programming language using Spring Framework 2.0.1 due to rapid prototyping. Developments have been made on Intelij Idea 2019.3.4 Ultimate Edition.The image and telemetry data transferred within the system must be transmitted synchronously to the client applications. For this reason, there is a need to send the image and telemetry in text format over the same channel.

Therefore, spring-websocket:5.0.5 is used to transfer data between Content Server and client applications. The tools used in the development environment are listed in Table 4.1.

Programming Language	Java
Intelij Idea	2019.3.4
Spring Framework	2.0.1
spring-websocket	5.0.5

Table 4.1 Content Server Development Environment

To create a wobsocket endpoint on Spring, a new config class must be created with @Configuration and @EnableWebsocket annotations. The config class written for Content Server is given in Listing 4.1. This class determines which endpoint the websocket is serving and which classes will handle this message when the connection message arrives. Content server is set to run on port 9000. Therefore, clients that want to connect to the content server should connect to the ”ws://xxxx:9000/uav” address.

```
@Configuration
@EnableWebsocket
public class UavSocketConfig implements WebSocketConfigurer{
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry){
        registry.addHandler(new UavSocketHandler(), "/uav").setAllowOrigins(*);
    }
}
```

Listing 4.1 Spring Websocket Config

The UavSocketHandler class is responsible for monitoring the active sessions in the system and the transferred videos. It creates a task that runs in the background for each client that joins the system. This task runs periodically according to the fps value of the data created in the 3.2.1 section, reads the frames recorded in the local file system, combines these frames with the telemetry information that these frames correspond to, creates the data structure described in 3.1 and sends them over the socket. When a message is received from the client to the server, this message is processed in the handleTextMessage method. Part of this

method is given in Listing 4.2. Client applications send the flight name they requested and, if this is a cancellation message, the cancel=true parameter in this message.

```
@Override
public void handleTextMessage(WebSocketSession session, TextMessage message){
   synchronized(this){
    String payload = message.getPayload();
    JSONObject jsonObject = new JSONObject(payload);
    String flightName = (String) jsonObject.get(flightName);
    String isCancel = (String) jsonObject.get("cancel");
    ..
   }
}
```

Listing 4.2 Websocket Text Message Handler

The application keeps the session information and transfers the image and telemetry data to the client applications through the code given in Listing 4.3 every time the task runs.

```
session.sendMessage(new TextMessage("datamodel"));
```

Listing 4.3 Websocket Send Message

The input, output and endpoint information of the Websocket service are summarized in Table 4.2. Example of using the service will be given in section 4.1.3.

Service endpoint	ws://xxxx:9000/uav
Inputs	flightName and cancel in json format
Output	Image and telemetry data in accordance with Data Model

Table 4.2 Content Server Parameters

4.1.2. Signaling Server

WebRTC requires an intermediate server to orchestrate client applications. After client applications share information over this server, they do the actual data sharing peer to peer.

Within the scope of the thesis, Signaling server was developed with javascript programming language in Node.js 12.18.4 rutime environment on Webstrom 2020.2.2. The tools used in the development environment are listed in Table 4.3. Node.js is a run-time platform that allows writing server-side applications with JavaScript. In addition, the 4.15.3 version of the express library which provides capabilities such as routing and templating and allows web application development with node.js is used. Socket.io version 1.7.3 was used to write the socket service. The code block, which creates the minimal infrastructure necessary for writing socket interfaces, is given in Listing 4.4.

Programming Language	JavaScript
WebStrom	2020.2.2
Node.js	12.18.4
Express	4.15.3
Socket.io	1.7.3

Table 4.3 Signaling Server Development Environment

```
1 var express = require ('express');
2 var app = module.exports = express.createServer();
3 var io = require ('socket.io')(app);
4 app.listen(3000, function(){});
```

Listing 4.4 Signalling Server Initialization

”signal”, ”request-flight”, ”remove-flight” and ”disconnect” channels are listened on the socket. The ”signal” channel is used for client applications to transfer necessary information to each other while establishing a WebRTC connection. Other channels are used when a client requests a new video, cancels a video it is viewing, and disconnects from the signaling server. These 3 channels directly change the WebRTC topology. The developed socket channels are given in Listing 4.5. The ”request flight” channel connects clients requesting the same video in order of join to the system. The first user who joined the system becomes the master and acts as a data source. Each client that joins the system is responsible for receiving the video from the client that joined before and distributing it to the client that joined after.The ”remove-flight” channel removes the requesting client from the topology

and re-establishes the connection between other clients. For example, if the client calling this channel is in the middle of the topology, the signaling server triggers the clients in front of and behind this client and allows them to establish a connection. The ”disconnect” channel works with the same logic as the ”remove-flight” channel. The only difference is that ”remove-flight” changes the topology for a single video, while the ”disconnect” channel changes all topologies client joined. The parameters that should be sent to these channels are listed in Table 4.4. Example of using the service will be given in section 4.1.3.

signal	sdp or ice candidates
request-flight	flight name
remove-flight	flight name
disconnect	none

Table 4.4 Channel Data

```
socket.on('request-flight', function (flightName) {
    sokcetMap.set(socket.id, flightName)
    if(!topologyMap.has(flightName)){
        topologyMap.set(flightName, [sokcet.id]);
        io.to(socket.id).emit('first-client', flightName);
    }else{
        topologyMap.get(flightName).push(socket.id);
        var socketList = topologyMap.get(flightName);
        var sourceSocketId = socketList[socketList.lenght - 2];
        io.to(socket.id).emit('create-client', sourceSocketId);
        io.to(sourceSocketId).emit('user-joined', sourceSocketId);
    }
});
socket.on('remove-flight', function (flightName) {
    var clients = topolgyMap.get(flightName);
    var index = clients.indexOf(socket.id);
    if(clients.lenght == 2){
        if(index == 0){
            io.to(clients[1]).emit('first-client', flightName);
        }else{
            io.to(clients[0]).emit('client-removed-flight', clients[1], flightName);
        }
    }else if(clients.length > 2){
        if(index == 0){
            io.to(clients[1]).emit('first-client', flightName);
        }else{
            var firstSocket = clients[index - 1];
            io.to(firstSocket).emit('client-removed-flight', clients[index],flightName);
            var secondSokcet = clients[index + 1];
            io.to(socket.id).emit('create-client', firstSocket, flightName);
            io.to(firstSocket).emit('user-joined', secondSokcet, flightName);
        }
    }
});
socket.on('disconnect', function () {
io.sockets.emit('user-left', sokcet.id);
for(var [key, value] of topologyMap.entries()){
    var index = value.indexOf(socket.id);
    if(value.length == 2){
        if(index == 0){
            io.to(value[1]).emit('first-client', key);
        }else if(value.lenght > 2){
            if(index == 0){
               io.to(value[1]).emit('first-client', key);
            }else{
                var firstSocket = value[index - 1];
                var secondSocket = value[index + 1];
                io.to(firstSocket).emit('user-joined', secondSocket, key);
            }
        }
    }
}
```

Listing 4.5 Signalling Server Channels

4.1.3. Map Application

Cesium.js version 1.87.1 was used as a web mapping tool within the scope of the thesis. Developments were made with Javascript programming language in Node.js runtime environment on WebStorm 2020.2.2 version. The 4.15.3 version of the express library was used to create the web application and the 1.7.3 version of the Socket.io to establish the websocket connections. The tools used in the development environment are listed in Table 4.5.

Programming Language	JavaScript
WebStrom	2020.2.2
Cesium.js	1.87.1
Node.js	12.18.4
Express	4.15.3
Socket.io	1.7.3

Table 4.5 Cesium App. Development Environment

This application developed on Cesium communicates with Signaling server and content server via websocket protocol. ”connect”, ”first-client”, ”create-client”, ”user-joined”, ”user-left”, ”client-removed-flight” and ”signal” channels are listened via Websocket. Messages from the ”connect” channel are received when the connection with the signaling server is successful. If a video is requested for the first time in the system, it is reported via the ”first-client” channel. When a message is received from this channel, the client application is triggered to establish a websocket connection with the content server. The ”user-joined” channel is used to notify the client that joined previously when a new client joins the system. The ”create-client” channel is used to trigger the newly joined user to establish a WebRTC connection with the previous client. The ”user-left” channel is used to notify all other clients when a user leaves the system. The ”client-removed-flight” channel is used to notify other clients when a user stops watching a video. The ”signal” channel is used for two clients to transfer the information necessary to establish a WebRTC connection. Each client keeps track of which client it connects to obtain which video, and updates its topology as messages come from these channels. Data coming to these channels are listed in Table 4.6.

connect	none
first-client	flight name
create-client	socketId of the peer
user-joined	socketId of the peer, flight name
user-left	socketId
client-removed-flight	socketId of the peer, flight name
signal	sdp or ice candidates

Table 4.6 Channel Data

When a client receives a message over the ”first-client” channel, it establishes a socket connection over the service point specified in Table 4.2 and sends the entries specified in this table to the content server over socket. Example code block is provided in Listing 4.6.

```
self.ws = new WebSocket("ws://xxxx:9000/uav");
var flightInfo = JSON.stringify ({
    'flight' : flightName,
    'cancel' : false | true,
});
sendMessage(flightInfo);
function sendMessage(msg){
    waitForSocketConnection(self, function(){
        self.ws.send(flightInfo);
    });
}
```

Listing 4.6 Creating Websocket Connection

In the application, the connection between client applications is provided peer to peer over WebRTC. Any installation or library is required to include WebRTC in the project. Since the Data Model introduced in Section 3.2 is in string format, WebRTC datachannel capability was used within the scope of the study. According to the coming messages from signaling server, client applications create a peer instance over the RTCpeerConnection interface and create the data transfer channel with the createDataChannel method. As the client receives ice candidates, it sends them to the remote peer via the ”signal” channel. Besides, it creates a local description and sends it over the ”signal” channel also. These messages are transmitted over the ”signal” channel of the remote client, and it creates its own local descriptions and sends them to the source client over the same channel.Once

client applications have agreed to establish a connection, they can begin transmitting data to each other via the datachannel.send() function.This transferred data is forwarded to the ondatachannel event.The data received via the ondatachannel event is parsed in the client application and the uav image in the scene is updated.If this data is to be transferred to another client, it is sent via the datachannel.send() method.The code block containing the ”user-joined” and ’signal’ channels is given in Listing 4.7 and Listing 4.8.

```
self.socket.on('signal', function (fromId, message){
        var signal = JSON.parse(message){
            if(signal.sdp){
                self.connections[fromId].setRemoteDEscription(new
                RTCSessionDescription(signal.sdp)).then(function(){
                    if(signal.sdp.type == 'offer'){
                        self.connections[fromId].createAnswer()
                        .then(function(description){
                            self.connections[fromID].setLocalDescription(description)
                            .then(function(){
                                    self.socket.emit('signal', fromID, JSON.stringify({
                                    'sdp': self.connections[fromId].localDescription}));
                            }
                        }
                    }
                }
            }else if(signal.ice){
                self.connections[fromId].addIceCandidate(new RTCIceCandidate(signal.ice));
            }
        }
    }
```

Listing 4.7 Signal Channel Implementation

```
self.socket.on('user-joined', function(id, data){
    if(!self.connections[id]){
        self.connections[id] = new RTCPeerConnection();
        var sendChannel = self.connections[id].createDataChannel("sendChannel");
        sendChannel.onopen = function (event){
            self.connections[id].onicecandidate = function (){
                if(event.candidate != null){
                    self.socket.emit('signal', id, JSON.stringfy({
                    'ice' : event.candidate}));
                }
            }
        }

        self.sendChannelMap.set(id, sendChannel);

        self.connections[id].ondatachannel = function(event){
            var receiveChannel = event.channel;
            receiveChannel.onmessage = function (event){
                var provideList = self.provideMap.get(flightName);
                for(var i = 0; i < provideList.lenght; i++){
                    var sendChannel = self.sendChannelMap.get(socketID));
                    if(sendChannel != endefined && sendChannel.readyState == "open"){
                        sendChannel.send(event.data);
                    }
                }

                self.handleDataChange(event.data);
            }
            self.receiveChannelMap.set(id, receiveChannel);
        }
    }

    self.connections[id].createOffer().then(function(description){
        self.connections[id].setLocalDescription(description).then(function(){
            self.socket.emit('signal', id, JSON.stringfy({'sdp' :
            self.connections[id].localDescription}));
    }
}
```

Listing 4.8 user-joined Channel

The sections covered up to this chapter deal with how video and telemetry data are distributed in the system. However, these obtained data need to be coordinated and visualized dynamically on Cesium.The Entity API provided by Cesium allows drawing geometries such as polygons on the 3D virtual globe, as well as adding model files with .glb extension. Using this API, a drone model whose position is updated as the drone moves, and 5 polygon instances have been added to draw the area seen by the drone as a prism. The code block that

allows adding the drone model and base polygon is in Listing 4.9. A UAV instance is created for each flight video, and data such as image and orientation in this instance are updated as updates come. The created entities are also updated using the properties of this class.

```
uav.viewer.entities.add({
    name: uav.name,
    position : new CallbackProperty(function(){
        return uav.getPosition();
    },false),
    orientation : new CallbackProperty(function(){
        return uav.getOrientation();
    },false),
    model : {
        uri :  "model url",
        minimumPixelSize : 120,
        maximumPixelSize : 180,
        maximumScale : 2
    }
});

uav.viewer.entities.add({
    polygon : {
        hierarch : new CallbackProperty( function (){
            return uav.getBasePolygonPoints();
        },false),
        material : new ImageMAterialProperty({
            image: "base64 string"
        }),
        stRotation : new CallbackProperty(function(){
            retrun uav.heading;
        },false)
    }
});
```

Listing 4.9 UAV Entities

The getBasePolygonPoints() method calculates the 4 vertices where the image intersects the terrain by following the steps described in section 3.4.1. This method is given in Listing 4.10.

```
function getBasePolygon() {
    var lowerLeftCorner = calculateTerrainIntersection(fovWidth / 2, fovHeight / 2);
    var loweRightCorner = calculateTerrainIntersection(-fovWidth / 2, fovHeight / 2);
    var upperRightCorner = calculateTerrainIntersection(-fovWidth / 2, -fovHeight / 2);
    var upperLeftCorner = calculateTerrainIntersection(fovWidth / 2, -fovHeight / 2, );
    return [lowerLeftCorner, upperLeftCorner, upperRightCorner, loweRightCorner];
}
```

Listing 4.10 Base Polygon Calculation

When the client application get updates via socket or WebRTC, the lat, lon, height, yaw, pitch, roll and image variables in the uav instances are updated. Before the scene is rendered, the entities make all calculations according to the current data in the uav class, and a continuously updated video is visualized on the scene.

4.1.4. Android Application

Android Studio 2020.3.1 as Android development environment, Gradle 3.4.2 as build tool and Xiaomi Redmi Note 8 Pro Android phone as debug/runtime environment were used while developing application. Developments were made with the Java programming language. Since WebRTC capability is not embedded in android platforms, google-WebRTC version 1.0.28513, play-services-location version 18.0.0 to use Google map services, socket.io-client version 0.5.0 to establish websocket connection was added to the project as dependency. The tools used in the development environment are listed in Table 4.7.

Programming Language	Java
Android Studio	2020.3.1
Gradle	3.4.2
debug/runtime	Xiaomi Redmi Note 8 Pro
google-WebRTC	1.0.28513
play-services-location	18.0.0
socket.io-client	0.5.0

Table 4.7 Android App. Development Environment

The steps of finding the location, calculating the rotations and taking the images through the application are explained in section 3.2.2 with the sample codes. Therefore, it will not be

mentioned again under this section. The images obtained as a result of aligning the sensor data of the Android phone with the aircraft model on the Cesium are given in Figure 4.1.

Figure 4.1 Plane Model on Cesium

When the application is started, it connects to the websocket service via the signaling server and then PeerConnection is created. Socket connection is given in Listing 4.11. The onRemoteCloseConnection channel is called when the remote peer terminates the connection. The onOfferReceived , onAnswerReceived and onIceCandidateReceived channels are used for peers to pass information necessary to establish a connection between them. The onNewPeerJoined channel is used when a new client is joined to the system. The onOfferReceived , onAnswerReceived and onIceCandidateReceived method implementations are provided in Listing 4.12.

```
try {
    Socket websocket = IO.socket("http://xxxx:3000");
    websocket.connect();
catch (URISyntaxException e) {
    e.printStackTrace();
}
```

Listing 4.11 Android Websocket Connection

```
public void onOfferReceived(final JSONObject data) {
    try {
        remotePeer.setRemoteDescription(new AndroidSdpObserver("androidSdpObserver"), new
        SessionDescription(SessionDescription.Type.OFFER, data.getString("sdp")));
        signal();
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
public void onAnswerReceived(JSONObject data) {
    try {
        remotePeer.setRemoteDescription(new AndroidSdpObserver("androidSdpObserver"),
        new SessionDescription(SessionDescription.Type.fromCanonicalForm(
        data.getString("type").toLowerCase()), data.getString("sdp")));
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
public void onIceCandidateReceived(JSONObject data) {
    try {
        remotePeer.addIceCandidate(new IceCandidate(data.getString("id"),
        data.getInt("label"), data.getString("candidate")));
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
private void signal() {
    remotePeer.createAnswer(new AndroidSdpObserver("createAnswer") {
        @Override
        public void onCreateSuccess(SessionDescription sessionDescription) {
            super.onCreateSuccess(sessionDescription);
            remotePeer.setLocalDescription(new AndroidSdpObserver("createAnswer"),
            sessionDescription);
            emitMessage(sessionDescription);
        }
    }, new MediaConstraints());
}
```

Listing 4.12 Signalig implementations

Then, a dataChannel is created over this peer and registered. The code block that runs the processing steps is given in Listing 4.13.

```
private void createPeerConnection() {
    PeerConnection.RTCConfiguration rtcConfig =
            new PeerConnection.RTCConfiguration(peerIceServers);

    rtcConfig.tcpCandidatePolicy = PeerConnection.TcpCandidatePolicy.ENABLED;
    rtcConfig.rtcpMuxPolicy = PeerConnection.RtcpMuxPolicy.NEGOTIATE;
    rtcConfig.continualGatheringPolicy = PeerConnection.ContinualGatheringPolicy
    .GATHER_CONTINUALLY;
    rtcConfig.keyType = PeerConnection.KeyType.ECDSA;
    remotePeer = peerConnectionFactory.createPeerConnection(rtcConfig, new
    PeerConnectionObserver("remotePeerCreation") {
        @Override
        public void onIceCandidate(IceCandidate iceCandidate) {
            super.onIceCandidate(iceCandidate);
            onIceCandidateReceived(iceCandidate);
        }
    });

    DataChannel.Init init = new DataChannel.Init();
    init.protocol = "json";
    dataChannel = remotePeer.createDataChannel("dataChannel", new DataChannel.Init());
    dc_observer = new DcObserver();
    dataChannel.registerObserver(dc_observer);
}
```

Listing 4.13 Android create datachannel

The application creates a task that runs periodically in the background using TimerTask. This task transmits image and telemetry data to Cesium client via datachannel.

```
ByteBuffer data = ByteBuffer.wrap(msg.getBytes(Charset.defaultCharset()));
dataChannel.send(new DataChannel.Buffer(data, false));
```

Listing 4.14 Datachannel send message

4.2. Results and Discussion

In this section, the results of WebRTC and Websocket performance tests, the metrics obtained as a result of direct georeferencing are given and potential improvement methods

are explained.

4.2.1. WebRTC and WebSocket Comparison

In this section, when various number of videos are distributed to various number of client applications over WebRTC and websocket, the load on the server and the amount of delay on client applications are measured comparatively. In all tests, the min, max and mean values of each client were measured, and these values were entered in the tables by separating them with commas. In addition, the averages of delays in users and the amount of load on the server were also measured and inserted to the tables. All tests were performed on Windows 10 operating system, Google Chrome version 101 as web browser.

In Test 1, it is aimed to measure the change in the load on the server when the number of clients increases, in the case of data transfer via websocket. In this test, content server and cesium applications were run on the same machine. Since WebRTC is not used, there is no need to run the signaling server. In this test, as shown in Figure 4.2, when a new client is joined to the system, a new websocket connection is established with the content server and image and telemetry data are transferred over this channel. This test was performed for 1 , 2, 4, 6, 8 and 10 clients separately and the results are listed in table 4.8.

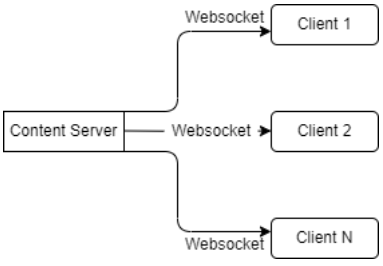

Figure 4.2 Test 1 Data Flow

Test 1 : Min, Max and Mean Package Delays Per Client with Websocket										
Test No	Client1	Client2	Client3	Client4	Client5	Client6	Client7	Client8	Client9	Client10
1.1	6,922,16									
1.2	7,847,17	7,912,18								
1.3	4,1089,24	4,1089,24	5,897,29	5,1255,24						
1.4	7,1412,41	6,1974,25	7,2920,70	4,3212,119	7,4143,110	11,5622,298				
1.5	7,3136,126	8,2732,90	5,989,45	7,4261,70	8,4478,176	12,8830,271	8,20749,3562	6,90266,5080		
1.6	6,7860,284	12,13965,810	7,9523,464	6,30418,7151	7,24122,1278	4,20510,1732	7,26678,1240	12,21317,6714	15,52592,2415	14,27145,1742

Table 4.8 Test 1 Results

The min, max and mean delays of all users were calculated from the data in Table 4.8. In addition, the amount of load on the server is measured and given in table 4.9. When these data are examined, it is seen that the min delay varies between 4-7 ms, regardless of the number of users. In addition, when the number of users increases, it is seen that the max delay increases from 922 ms to 27145 ms. As the number of users increases, the load on the server increases linearly. In addition, when the number of clients is less than 6, the average delay is almost the same, and when the number of clients exceeds 6, the delays in the system increase also linearly. Similarly, it is seen that the average delay is almost the same when the load on the server is less than 12 mb/s, but the amount of delay increases rapidly after passing this threshold. Besides, it has been observed that when the number of users exceeds 8, socket connections are lost. Graphs showing the relationship between the number of users, server load and average delay are given in Figure 4.3.

Test 1 : Mean Package Delay and Server Load with Websocket				
Test No	Min Delay(ms)	Max Delay(ms)	Mean Delay(ms)	Server Load(mb/s)
1.1	6	922	16	2.1
1.2	7	847	17	4.1
1.3	4	1435	25	8.2
1.4	4	5622	110	11.2
1.5	5	20749	1177	15.3
1.6	4	27145	2423	17.8

Table 4.9 Test 1 : Min, Max, Mean Delay and Server Load

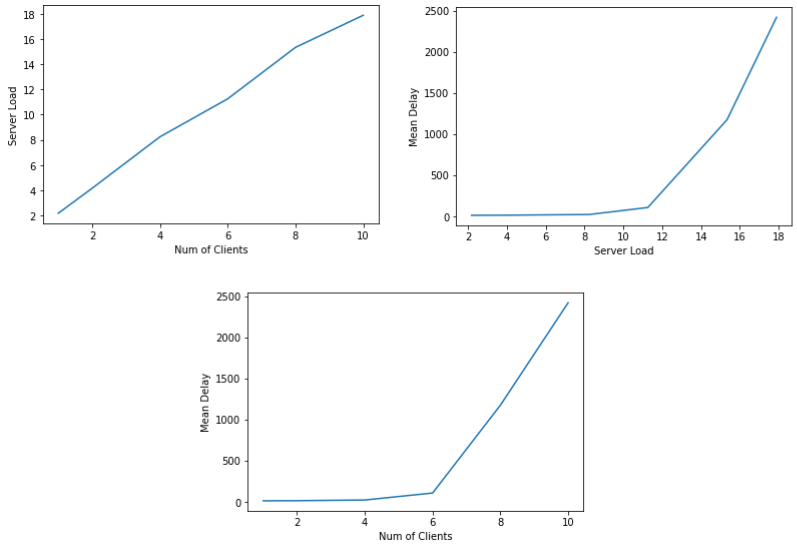

Figure 4.3 Test 1 Server load - Mean Delay- Num of Clients Graphs

In Test 2, it is aimed to measure the change in the amount of load on the server as the number of clients increases, when data transferred via WebRTC. In addition, it has been observed how the metrics change when the clients are connected to each other with different topologies while there are the same number of clients in the system. In this test, content server, signaling server and cesium applications were run on the same machine. The tests were repeated separately for 2, 4, 6, 8 and 10 users and the results are summarized in table 4.10. In Test 2.1, while there were two users in the system, the first user received the video over the socket and then transferred the video to the other client via WebRTC. This structure is shown in Figure 4.4. When there are 4 users in the system, 2 different data transfer paths are created and these are illustrated in Figure 4.5 and Figure 4.6. The case of 6 users in the system has been tested over 3 different topologies, and the structure illustrated in Figures 4.7, 4.8 and 4.9. When there are 8 and 10 users in the system, half of the users receive the data over the socket and transfer it to the other half via WebRTC. These data flows are illustrated in Figure 4.10 and Figure 4.11.

Test 2 : Min, Max and Mean Package Delays Per Client with WebRTC										
Test No	Client1	Client2	Client3	Client4	Client5	Client6	Client7	Client8	Client9	Client10
2.1	7,900,18	13,1371,30								
2.2	6,737,23	8,912,42	7,1005,74	8,965,92						
2.3	7,1200,22	12,1350,55	5,1270,26	13,1481,56						
2.4	7,701,24	13,850,43	17,1231,65	4,1884,32	9,2034,219	22,1539,297				
2.5	7,718,34	12,968,98	8,1053,38	15,2064,110	8,1716,41	16,1675,102				
2.6	5,1700,22	12,943,55	17,2073,100	22,1861,267	29,945,387	42,2179,451				
2.7	3,798,110	7,916,275	8,1100,210	23,1263,337	9,2755,512	19,3122,812	6,2178,3575	22,6677,8127		
2.8	6,800,194	17,912,1057	7,2298,1414	24,3651,3422	15,2901,2445	37,4221,5437	25,1958,2277	57,4671,8247	72,2341,4322	108,5367,11380

Table 4.10 Test 2 Results

Figure 4.4 Test 2.1 Data Flow

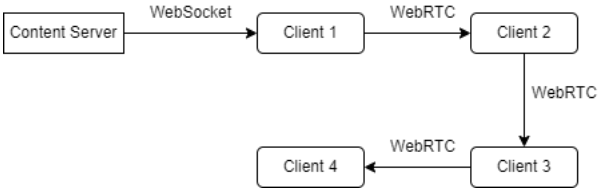

Figure 4.5 Test 2.2 Data Flow

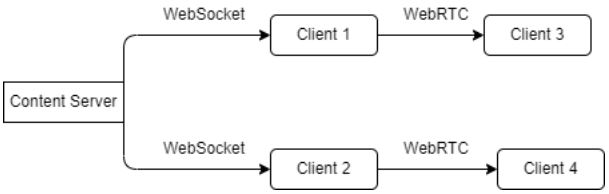

Figure 4.6 Test 2.3 Data Flow

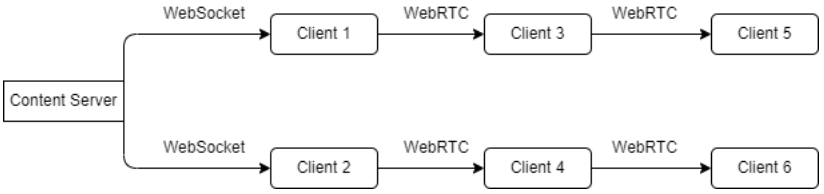

Figure 4.7 Test 2.4 Data Flow

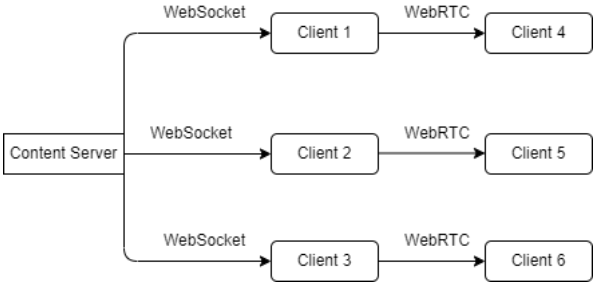

Figure 4.8 Test 2.5 Data Flow

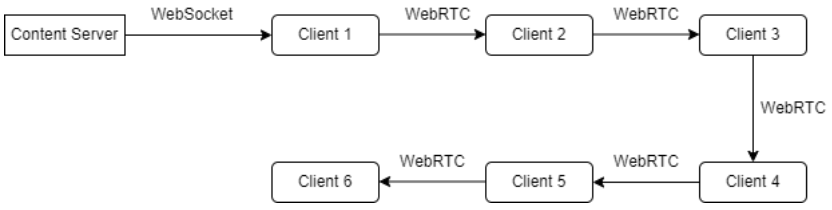

Figure 4.9 Test 2.6 Data Flow

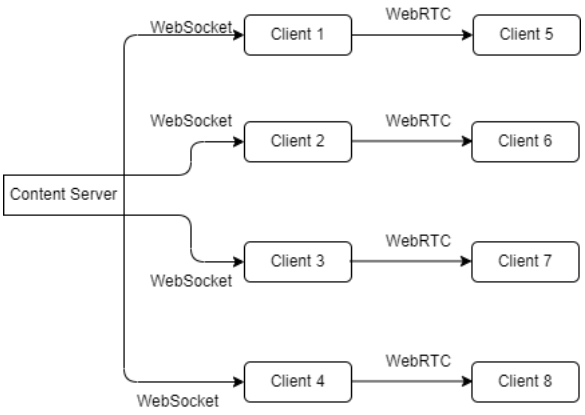

Figure 4.10 Test 2.7 Data Flow

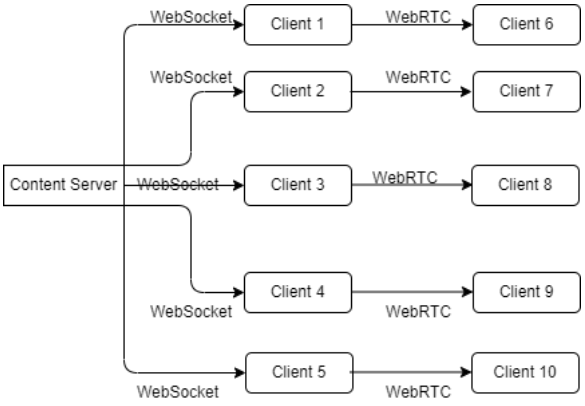

Figure 4.11 Test 2.8 Data Flow

The min, max and mean delays of all users are calculated based on the data and listed in Table 4.10. In addition, the amount of load on the server is measured and given in table 4.11. When these data are examined, it is seen that the min delay varies between 3-7 ms, regardless of the number of users. Besides, when the number of users increases, it is seen that it reaches up to a maximum of 6667 ms. When the number of clients in the system exceeds 6, it is seen that the load on the server starts to increase, but reaches a maximum of 10 mb/s. The results showed that the applied topology has a significant effect on the system load and the average delay. As a result of 3 tests run for 6 users, delays of 113.33ms, 70.5ms and 213.6ms were obtained. In response, the server load is measured at 4,195, 6,421 and 2,201 mb/s.This test shows that the system can serve more users by increasing the amount of delay in bandwidth-constrained systems. In addition, it is seen that less delay can be obtained

by increasing the load on the server in systems with higher bandwidth. Graphs showing the relation between the number of users, server load and average delay are given in Figure 4.12. However, it should be noted that these graphs are only valid for topologies created within the scope of the this thesis. Different results will be obtained for different topologies and test environments.

Test 2 : Mean Package Delay and Server Load with WebRTC				
Test No	Min Delay(ms)	Max Delay(ms)	Mean Delay(ms)	Server Load(mb/s)
2.1	7	1371	24	2.0
2.2	6	1005	57	2.1
2.3	7	1481	39	4.2
2.4	4	2034	113	4.1
2.5	7	2064	70	6.4
2.6	5	2.179	213	2.2
2.7	3	6667	1297	8.5
2.8	6	5367	3525	10.2

Table 4.11 Test 2 : Min , Max, Mean Delay and Server Load

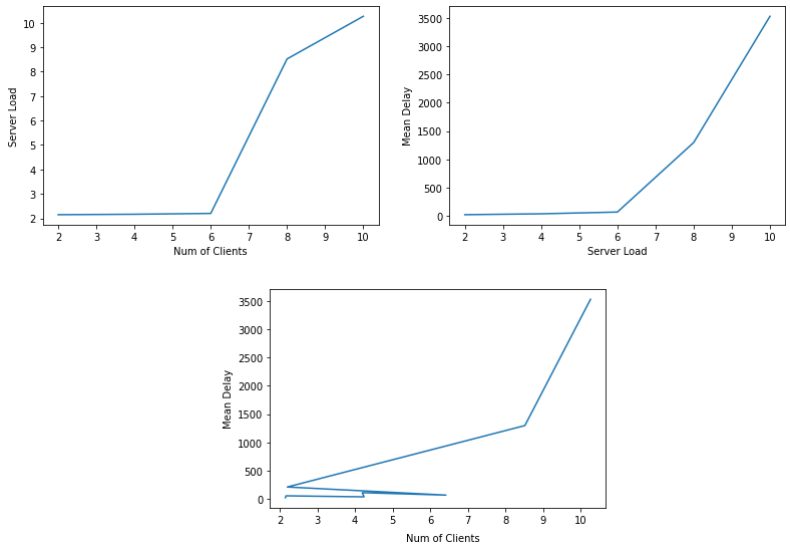

Figure 4.12 Test 2 Server load - Mean Delay- Num of Clients Graphs

When the results of Test 1 and Test 2 performed on the local system are compared; when the number of clients in the system exceeds 8 websocket becomes unresponsive and WebRTC continues to transfer data, the amount of load on the server decreases significantly with WebRTC, on the other hand, in some cases, the amount of delay in WebRTC is higher than websocket.

In Test 3, Test 4, Test 5 and Test 6 scenarios, each of the server and client applications were run on different machines in the same network. In Test 3, it is aimed to measure the amount

of load and delay on the server when there is more than one user and more than one video source in the system and the frame generation rate is 25 fps.The data transmission channels used in this test are shown in Figure 4.13. There are as many different videos as the number of users in the system and it is aimed for each user to view all the videos in the system. For example, in the scenario where there are 3 users, there are 3 videos in the system and these videos are transferred to the clients over 3 socket connections. The results obtained by the measurement made are listed in Table 4.12. When there are 2 users and 2 videos in the system, the average server load is 8.674 mb/s, and the average delay is measured as 26.6 ms. However, in cases where there are 3 and 4 users, the socket connections are lost.

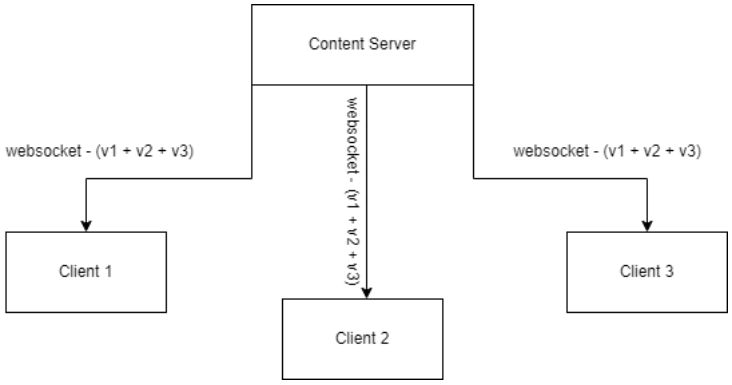

Figure 4.13 Test 3 Data Flow

Test 3 : Mean Package Delay and Server Load with Websocket							
Test No	Num. of Videos Per Client	Client 1	Client 2	Client 3	Client 4	Mean Delay(ms)	Server Load(mb/s)
3.1	2	14,921,24	9,1024,29			26	8.6
3.2	3	7,821,1312	15,937,1456	9,1145,1378		1382	17.1
3.3	4	9,911,4856	12,768,4942	17,874,5324	10,1014,5021	5302	18.9

Table 4.12 Websocket and 25 FPS Results

In Test 4, data was transferred via WebRTC with 25fps using the structure in Figure 4.14 and the metrics obtained are summarized in Table 4.13. When the results are compared with the results of Test 3, it is seen that when the data is transferred over WebRTC the server load is halved, the average delays are very close to each other and WebRTC continues to provide service with delays of 70.6 and 128.25 ms for 3 and 4 users while websocket becomes unresponsive.

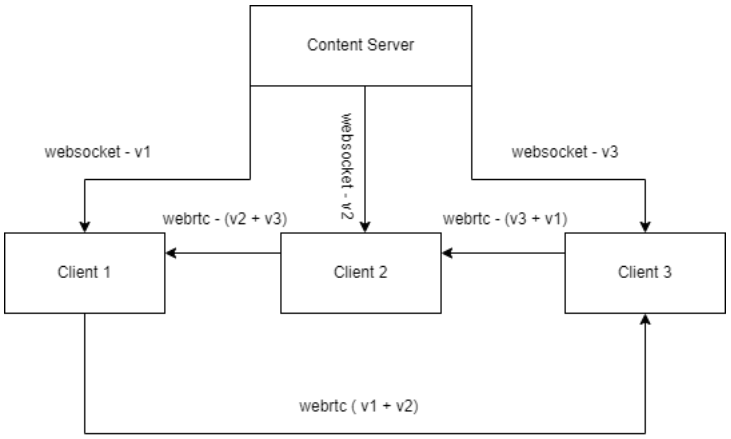

Figure 4.14 Test 4 Data Flow

Test 4 : Mean Package Delay and Server Load with WebRTC							
Test No	Num. of Videos Per Client	Client 1	Client 2	Client 3	Client 4	Mean Delay(ms)	Server Load(mb/s)
4.1	2	7,978,28	9,795,29			28	4.6
4.2	3	8,1011,65	15,974,69	9,1127,78		70	6.8
4.3	4	12,1427,115	15,1632,132	9,1339,139	17,1211,127	128	8.4

Table 4.13 WebRTC and 25 FPS Results

In Test 5, data was transferred over the websocket with 15fps and the results are summarized in Table 4.14. When the metrics are compared with the results of Test 3, it is seen that the amount of load on the server decreases with the decrease in the fps value. While the websocket could not respond for 3 users in test 3, it was seen that it continued to serve with a delay of 125.6 ms in test 5. Websocket became unresponsive again for 4 users.

Test 5 : Mean Package Delay and Server Load with Websocket							
Test No	Num. of Videos Per Client	Client 1	Client 2	Client 3	Client 4	Mean Delay(ms)	Server Load(mb/s)
5.1	2	8,897,22	12,1012,24			23	5.1
5.2	3	9,872,127	13,994,116	9,1015,134		125	11.4
5.3	4	10,14575,3894	9,22972,5527	15,21901,4935	8,28791,5124	4917	19.1

Table 4.14 Websocket and 15 FPS Results

The data produced with 15 fps in Test 6 was transferred via WebRTC. The metrics obtained are summarized in Table 4.15. In this scenario, it has been seen that WebRTC can serve for 2, 3 and 4 clients. Compared to test'5, it is seen that it reduces the server load by half. It has been seen that while it has approximately the same delay for 2 users, it reduces the delay by half for 3 users.

Test 6 - Mean Package Delay and Server Load with WebRTC							
Test No	Num. of Videos Per Client	Client 1	Client 2	Client 3	Client 4	Mean Delay(ms)	Server Load(mb/s)
6.1	2	11,901,27	9,869,26,24			26	2.8
6.2	3	9,1012,61	15,990,60	13,1260,64		61	4.1
6.3	4	11,1427,108	17,1632,121	11,1339,117	12,1211,119	116	5.3

Table 4.15 WebRTC and 15 FPS Results

4.2.2. Direct Georeferencing

In this section, the results obtained by the direct georeferencing method used in the thesis are compared with Google Maps, Bing Map and Openstreet Map. The results obtained by calculating the points where the 4 corners of the image intersect the Earth's surface and displaying them on Cesium are given in Figure 4.15. Openstreet map was used as the base map. There is a discrepancy between the buildings on the base map and the aerial photographs that are coordinated with direct georeferencing. In order to measure these shifts in the images, aerial photographs were opened in the QGis application and the distance between reference points was measured.

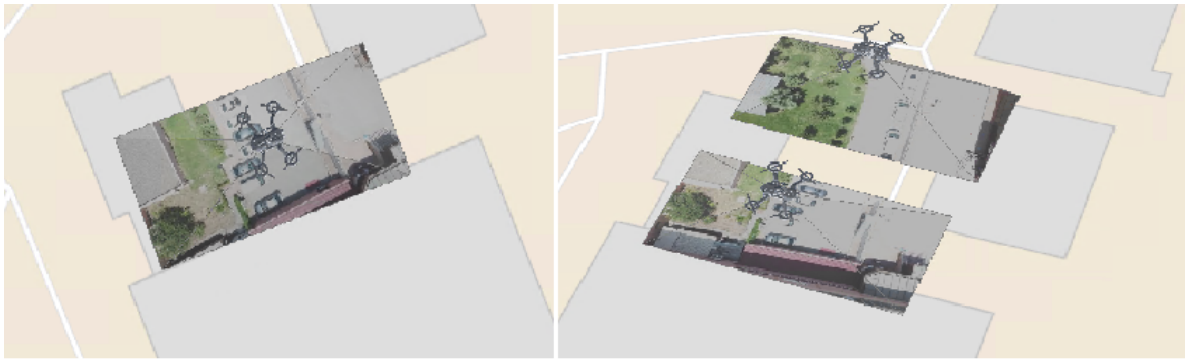

Figure 4.15 Georeferenced Images on Cesium with Drone Model

The pavement determined as a reference in Figure 4.16 was marked on Bing Map and georeferenced aerial photography, and the distance between them was measured as 2.68 meters.

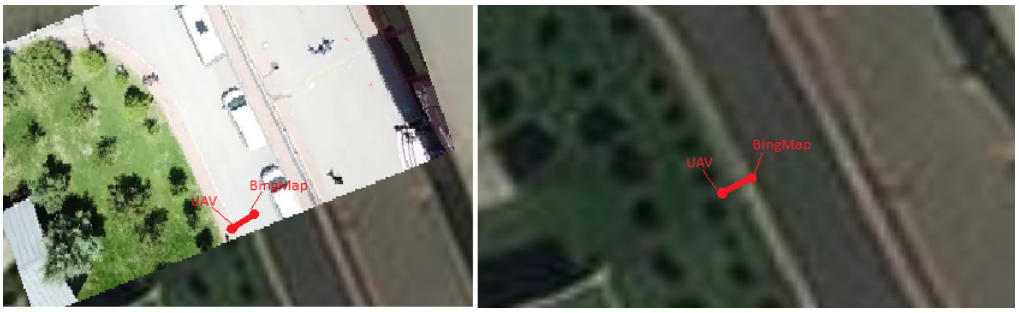

Figure 4.16 Georeferenced Image - Bing Map Comparison - 1

In Figure 4.17, the width of the road is marked in the Bing Map and the aerial photograph, and a 3.01 meter shift was measured between these two maps.

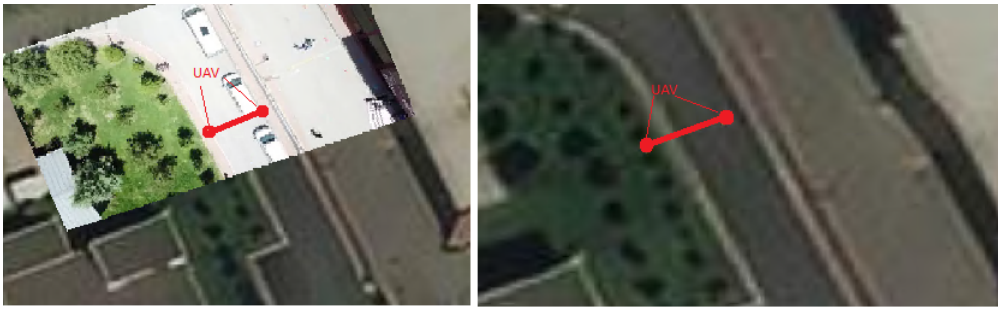

Figure 4.17 Georeferenced Image - Bing Map Comparison - 2

In Figure 4.18, corner point of the road is marked in the Bing Map and the aerial photograph, and it has been determined that there is 5.4 meters shift between these two maps. The measurements made are listed in Table 4.16.

Figure 4.18 Georeferenced Image - Bing Map Comparison - 3

Bing Map			
Meas. 1	Meas. 2	Meas. 3	Mean (m)
2.68	3.01	5.4	3.34

Table 4.16 Bing Map measurements

In Figure 4.19, the corner point of the sidewalk was taken as a reference and it was determined that there was a 6.81 meter difference between the aerial photograph and Google Maps.

Figure 4.19 Georeferenced Image - Google Map Comparison - 1

In Figure 4.20, the corner point of the sidewalk was taken as a reference and it was determined that there was a 2.7 meters difference between the aerial photograph and Google Maps.

Figure 4.20 Georeferenced Image - Google Map Comparison - 2

The pavement determined as a reference in Figure 4.21 was marked on Google Map and georeferenced aerial photography, and the distance between them was measured as 6.08 meters. The measurements made are listed in Table 4.17.

Figure 4.21 Georeferenced Image - Google Map Comparison - 3

Google Map			
Meas. 1	Meas. 2	Meas. 3	Mean (m)
6.81	2.27	6.08	5.05

Table 4.17 Google Map measurements

In Figure 4.22, the corner point of the building is marked in the Openstreet Map and the aerial photograph, and a 5.01 meter shift was calculated between these two maps.

Figure 4.22 Georeferenced Image - Openstreet Map Comparison - 1

In Figure 4.23, the section that appears as a staircase in the aerial photograph coincides with the roof in the Openstreet Map. Therefore, it has been determined that there is a difference of at least 4.24 meters.

Figure 4.23 Georeferenced Image - Openstreet Map Comparison - 2

Openstreet Map		
Meas. 1	Meas. 2	Mean (m)
5.01	4.24	4.625

Table 4.18 Openstreet Map Measurements

In addition to the distance difference in aerial photographs and base maps, there are also shifts in reference points in these base maps. In Figure 4.24 and Figure 4.25, the distances between Google Map and Bing map were measured as 4.38 meters and 3.11 meters.

Figure 4.24 Google Map - Bing Map Comparison - 1

Figure 4.25 Google Map - Bing Map Comparison - 2

The location information from the Android embedded GPS, the orientation from the sensors and the image from the camera were taken and displayed on Cesium. Processing steps for direct georeferencing are the same as in the test cases described in the previous steps. The images obtained as a result of the direct georeferencing are given in Figure 4.26. Reference points were taken from the Qgis application and the shifts were approximately measured with the basemap. Measurements are given in Figure 4.27. It has been determined that there are approximately 59 meters shifts on average.

Figure 4.26 Android - Georeferenced Images with GPS

Figure 4.27 Referenced Points Shifts with GPS

The location information of the place where the images was captured taken from the Open Street map instead of using the GPS, and the same test was performed and the results are given in Figure 4.28. Reference points were taken from the Qgis application and the shifts

were approximately measured with the basemap. Measurements are given in Figure 4.29. It has been determined that there are approximately 13.5 meters of shifts on average.

Figure 4.28 Georeferenced Images with Manually Entered Coordinates

Figure 4.29 Referenced Points Shifts with Manuel Entered Coordinates

4.2.3. Discussion

When the results obtained in Section 4.2.1 are analyzed, it is seen that when the number of clients in the system increases and when the data is distributed over websocket, the amount of load on the server and the amount of delay on the users increase linearly. When the number of users exceeds a threshold, socket connections become unresponsive. On the other hand, when the same test is done over WebRTC, it is seen that the load on the server is less compared to websocket. With the WebRTC solution, the delay on the user can be increased, and the load on the server can be reduced. In addition, when the Test 2 results are investigated, it is

seen that the WebRTC topology applied to the same number of users has a great effect on delay and server load. In systems with low bandwidth, the load on the server can be reduced by using WebRTC and thus more users can be served. In addition, with different WebRTC topologies to be applied, users can be responded to faster by increasing the load on the server in systems with high bandwidth.

Within the scope of the thesis, direct georeferencing approach was used to georeference the aerial images. The intersection points of the rays drawn from the four corners of the image and the ground surface were calculated and the image was placed between these coordinates. Although this method does not affect the results much since there is not much change in the terrain in the area studied within the scope of the thesis, and drone images are obtained with low flight, it poses a problem in regions where the topography varies. The method followed is given in Figure 4.30. The image is clamped to the flat ground indicated by the green line, ignoring the topographical changes that occur along the Earth's surface. To overcome this problem, terrain correction was applied using surface polygon. With this method, changes in the terrain are also taken into account. Since there is no change in terrain in the area studied within the scope of the thesis, this test was carried out in a mountainous region. Figure 4.31 shows the default base map of Cesium and the Sentinel-2 satellite image of the same region. The results obtained by using the surface polygon capability provided by Cesium are given in Figure 4.32.

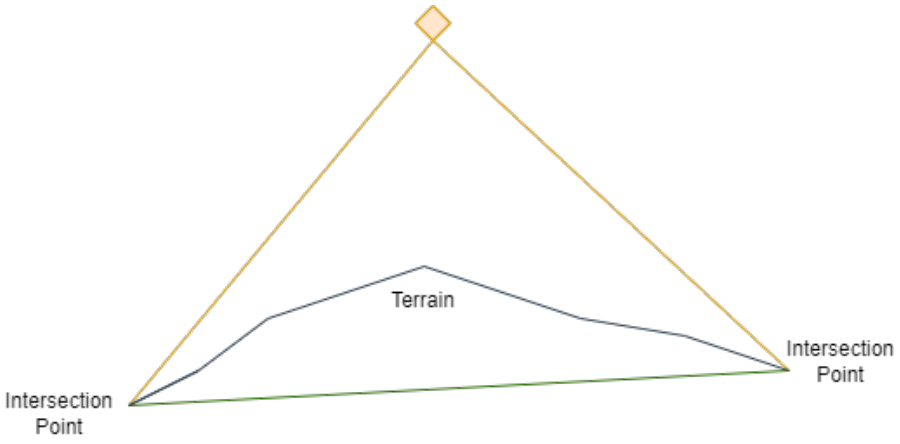

Figure 4.30 Terrain Intersection

Figure 4.31 Cesium Basemap and Sentinel - 2 Image

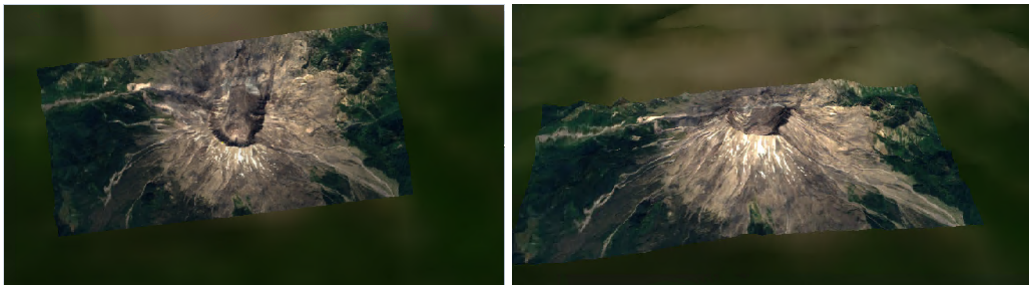

Figure 4.32 Surface Polygon on Cesium

Although this method improves the positioning of the image, since the image is clamped on the terrain by calculating the 4 corner points, the regions that are not in the UAV's field of view should be neglected while making the calculations. With the surface polygon method, although the image taken from the drone camera does not contain red regions, as shown in Figure 4.33, the image is clamped here as well. This means that a pixel corresponding to the top of the building in the aerial photograph is assigned to the point where the ray drawn along this direction intersects the terrain. Besides, images cannot be clamped on 3D models with the surface polygon method. For this reason, it is necessary to develop a ray tracing algorithm that calculates at which point each ray drawn from the camera intersects with the topography or buildings. These algorithms can be developed with glsl-based programs.

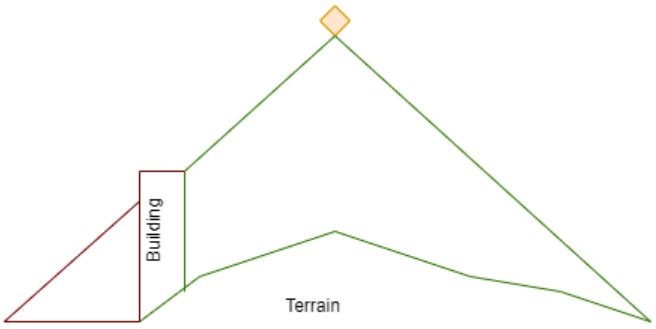

Figure 4.33 Ray Tracing Issue - Buildings

To summarize, as a result of direct georeferencing made with terrain intersection and terrain correction processes applied within the scope of the thesis, a maximum of 6.08 meters shift was detected between Google Maps, Bing Map and Openstreet Map. It is thought that the reason for these shifts may be due to inaccuracies in the FOV width and FOV height parameters used in the calculations. However, it should be taken into account that there are shifts in the images provided by Google Map, Bing Map and Openstreetmap, since these images are also captured from different perspectives. In addition, the imageries obtained from satellites contain geometric distortions because of the atmospheric conditions, curvature of the Earth, topographic effects, rotation of the Earth and the sensor errors. These geometric distortions cause changes in the scale of the image, irregularities in the angular relations between the elements in the image, incorrect positioning of the objects in the image and occlusion of one image element by another [72]. Within the scope of the thesis, it is mentioned that direct georeferencing can be done with 3 methods. The first is to add the image on the map by calculating the intersection points with ground, the second is to clamp the image on the terrain using surface polygon by considering the changes in the topography, and the third is to clamp the image to the building and topography by developing ray tracing algorithms with the help of glsl programs. Although the 3rd method is the method that will give the most accurate result, improvements have been made with the existing Cesium capabilities up to a point, but the ray tracing problem has not been issued within this study.

When the direct georeferencing process is applied using the position and orientation values obtained using the GPS, camera and sensors embedded in the Android phone, approximately

60 meters of shifts occur in the image. In addition, when the coordinates from which the image was taken were selected on Openstreet Map and entered into the system manually, it was observed that the shifts decreased to the level of 15 meters. This shows that the accuracy of the location information received via Android GPS and also orientation information is not sufficient. In addition, when orientation data is received using the ARCore API, it has been observed that this library works more stable by capturing surfaces in small areas, but cannot calculate orientation because it cannot capture surfaces at long distances. For this reason, it is considered that the use of this library within the scope of direct georeferencing is not appropriate.

With the results obtained, it has been seen that video and telemetry data from multiple platforms can be distributed to multiple clients without need for powerful servers and high bandwidth. In this respect, it is thought that the study made a contribution to the literature. In addition, georeferencing process was performed in milliseconds and images were displayed on the terrain with the 3D drone model on Cesium. In this study, it is considered that the result obtained is sufficient since it is not aimed to provide operations that require high position accuracy such as determining target coordinates and aimed to display the video streams from the operation field on the terrain at their approximate positions. This study on the ability to play near real-time drone videos on Cesium terrain is considered to be an important contribution. Solving the ray tracing problem over Cesium, increasing the positioning accuracy and making operations such as coordinate determination over this architecture are new research topics. The use of visual odometry techniques, for example, can be considerd to improve position and orientation of UAV telemetry data. In addition, the ability of people to geo-cast through the Android application is seen as another contribution to the literature.

Chapter 5

CONCLUSION

The main focus of this study is to transfer synchronized aerial photography and telemetry data between client applications without the need for powerful intermediate servers and visualize them on a Cesium-based application by direct georeferencing in real time. In this context, comparisons were carried out between Websocket and WebRTC as data transfer protocols, and it was observed that WebRTC greatly reduced the load on the server compared to websocket due to its peer-to-peer structure. When the number of active users in the system increases, the load on the websocket increases linearly, while it is seen that the server load can be reduced with different topologies applied over WebRTC. In addition, while the websocket transfers data directly between server and client, in the WebRTC solution, some of the clients receive data directly from the server, while others receive it through another client. This structure increases the amount of delay in the system. Considering these metrics, WebRTC is considered to be a suitable solution for systems with low bandwidth to serve more users. In addition, by applying different WebRTC topologies in systems that do not have bandwidth problems, the amount of load on the server can be increased and the amount of delay can be reduced. For this reason, WebRTC is considered to be a strong alternative to Websocket.

The applied direct georeferencing method is based on calculating the intersection of the corner points of the image with the terrain. Terrain correction was performed on the obtained image using surface polygon. When the coordinates obtained by the applied direct

georeferencing method were compared with Google Maps, Bing Map and Openstreet Map for the selected reference points, it was determined that there was a maximum shift of 6.08 meters. It should be taken into account that this amount of shift may be affected by the FOV width and FOV height values used in the calculations. In addition, satellite images used for comparison also contain geometric distortions. Therefore, to get more accurate results, spatial comparisons should be made with true orthophotos or with ground control points. In addition, the developed method clamps the image only on the terrain, not on the 3D building models. It is necessary to develop a GLSL-based applications in order to clamp the image to the building models, but the ray tracing problem could not be solved with the existing Cesium capabilities.

With the developed application, it was seen that images transferred from Android phone with WebRTC can be visualized on Cesium-based clients with direct georeferencing method. In addition, errors in georeferencing are increasing due to inconsistencies in the GPS and orientation sensors embedded in Android phones. Therefore, there is a need to develop methods to make position and orientation data more accurate and consistent. In addition, orientation information was obtained from the ARCore library, but it was seen that this library could not be used in this context because it did not work correctly on distant surfaces.

With the developed Cesium-based geovisualization method, it is possible to increase the situational awareness of the people who manage the operation in situations such as war and natural hazards that require urgent intervention. In this study, data streaming from multiple platforms is distributed to multiple clients using WebRTC. Thus, the system can be fed directly with the client/server architecture, as well as it is designed to work peer-to-peer in order to provide the same image flow in cases where the network is affected, the bandwidth is low or servers do not have sufficient processing power. In addition, the developed system can be fed not only by drones, but also by any platform that produce image and orientation information. With the developed Android-based application, in cases where drones cannot be reached immediately to the operation area, people in the region are able to perform geo-casting via their own mobile phones. In this way, it is possible to obtain many video streams from the same region from different perspectives. Besides, in the next stage of the

study, it is planned to reduce the errors caused by Android GPS and to solve the ray tracing problem with the GLSL-based developments to be made on Cesium to improve positional accuracy.

REFERENCES

[1] Ahmed Errami and Mohammed Khaldoun. Sensing by Drone. *2018 Int. Conf. Electron. Control. Optim. Comput. Sci.*, pages 1–5, **2018**.

[2] I. Jeon, S. Ham, J. Cheon, A. M. Klimkowska, H. Kim, K. Choi, and I. Lee. A real-time drone mapping platform for marine surveillance. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, 42(2/W13):385–391, **2019**. ISSN 16821750. doi:10.5194/isprs-archives-XLII-2-W13-385-2019.

[3] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. SkyEyes: Adaptive video streaming from UAVs. *Proc. Annu. Int. Conf. Mob. Comput. Networking, MOBICOM*, pages 2–6, **2016**. doi:10.1145/2980115.2980119.

[4] Yeung Siu Fung, John C.S. Lui, and David K.Y. Yau. Secure real-time streaming protocol (RTSP) for hierarchical proxy caching. *Int. J. Netw. Secur.*, 7(3):310–322, **2008**. ISSN 1816353X.

[5] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. WebRTC technology overview and signaling solution design and implementation. *2015 38th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2015 - Proc.*, (May):1006–1009, **2015**. doi:10.1109/MIPRO.2015.7160422.

[6] Kyle Taylor. Master degree thesis. (November):1–77, **2011**.

[7] Albert. Abelló. Lozano. Performance analysis of topologies for Web-based Real-Time Communication (WebRTC). page 96, **2013**.

[8] Vaclav Stebra. Webrtc communication portal, **2018**.

[9] Juan Pablo Garc. IPTV using P2PSP and HTML5 + WebRTC The Peer-To-Peer Straightforward Protocol. (March):1–5, **2014**.

[10] Masaki Kohana and Shusuke Okamoto. A data sharing method using webrtc for web-based virtual world. *Lect. Notes Data Eng. Commun. Technol.*, 17:880–888, **2018**. ISSN 23674520. doi:10.1007/978-3-319-75928-9_81.

[11] Toru Kobayashi, Hiroaki Matsuoka, and Shouta Betsumiya. Flying Communication Server in case of a Large-scale Disaster. *Proc. - Int. Comput. Softw. Appl. Conf.*, 2:571–576, **2016**. ISSN 07303157. doi:10.1109/COMPSAC.2016.117.

[12] Agnieszka Chodorek, Robert Ryszard Chodorek, and Paweł Sitek. *Monitor Urban and Industrial Areas*. **2021**. ISBN 4812617480.

[13] Dennis Edler and Mark Vetter. The Simplicity of Modern Audiovisual Web Cartography: An Example with the Open-Source JavaScript Library leaflet.js. *KN - J. Cartogr. Geogr. Inf.*, 69(1):51–62, **2019**. ISSN 25244965. doi:10.1007/s42489-019-00006-2.

[14] Cesium. https://cesium.com/, **2022**.

[15] T. Santhanavanich, S. Schneider, P. Rodrigues, and V. Coors. INTEGRATION and VISUALIZATION of HETEROGENEOUS SENSOR DATA and GEOSPATIAL INFORMATION. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, 4(4/W7):115–122, **2018**. ISSN 21949050. doi:10.5194/isprs-annals-IV-4-W7-115-2018.

[16] Paul R Wolf, Bon A. Dewitt, and Benjamin E. Wilkinson. *Elements of Photogrammetry with Applications in GIS, 4th ed. 696 pp*. **2014**. ISBN 9780071761123.

[17] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey, and J. M. Reynolds. 'Structure-from-Motion' photogrammetry: A low-cost, effective

tool for geoscience applications. *Geomorphology*, 179:300–314, **2012**. ISSN 0169555X. doi:10.1016/j.geomorph.2012.08.021.

[18] D. Hein and R. Berger. TERRAIN AWARE IMAGE CLIPPING for REAL-TIME AERIAL MAPPING. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, 4(1):61–68, **2018**. ISSN 21949050. doi:10.5194/isprs-annals-IV-1-61-2018.

[19] Daniel Hein, Thomas Kraft, Jörg Brauchle, and Ralf Berger. Integrated UAV-based real-time mapping for security applications. *ISPRS Int. J. Geo-Information*, 8(5), **2019**. ISSN 22209964. doi:10.3390/ijgi8050219.

[20] Marcin Hoffmann, Paweł Kryszkiewicz, and Georgios P. Koudouridis. Modeling of Real Time Kinematics localization error for use in 5G networks. *Eurasip J. Wirel. Commun. Netw.*, 2020(1), **2020**. ISSN 16871499. doi:10.1186/s13638-020-1641-8.

[21] Android. https://source.android.com/devices/sensors/sensor-types, **2022**.

[22] Chris Murphy and Hanumant Singh. Rectilinear coordinate frames for Deep sea navigation. *2010 IEEE/OES Auton. Underw. Veh. AUV 2010*, (May 2014), **2010**. doi:10.1109/AUV.2010.5779654.

[23] Piotr Kardasz and Jacek Doskocz. Drones and Possibilities of Their Using. *J. Civ. Environ. Eng.*, 6(3), **2016**. doi:10.4172/2165-784x.1000233.

[24] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, 7:48572–48634, **2019**. ISSN 21693536. doi:10.1109/ACCESS.2019.2909530.

[25] Neil Cameron. *Electronics Projects with the ESP8266 and ESP32*. **2021**. ISBN 9781484263358. doi:10.1007/978-1-4842-6336-5.

[26] Lijing Zhang and Xiaoxiao Shen. Research and development of real-time monitoring system based on WebSocket technology. *Proc. - 2013 Int. Conf. Mechatron. Sci. Electr. Eng. Comput. MEC 2013*, pages 1955–1958, **2013**. doi:10.1109/MEC.2013.6885373.

[27] Soumya Kanti Datta, Jean Luc Dugelay, and Christian Bonnet. IoT Based UAV Platform for Emergency Services. *9th Int. Conf. Inf. Commun. Technol. Converg. ICT Converg. Powered by Smart Intell. ICTC 2018*, pages 144–147, **2018**. doi:10.1109/ICTC.2018.8539671.

[28] Hongchi Zhang, Anas Al-Nuaimi, Xiaoyu Gu, Michael Fahrmair, and Ryota Ishibashi. Seamless and efficient stream switching of multi-perspective videos. *2012 19th Int. Pack. Video Work. PV 2012*, pages 31–36, **2012**. doi:10.1109/PV.2012.6229741.

[29] M. Bacco, M. Catena, T. De Cola, A. Gotta, and N. Tonellotto. Performance Analysis of WebRTC-Based Video Streaming Over Power Constrained Platforms. *2018 IEEE Glob. Commun. Conf. GLOBECOM 2018 - Proc.*, pages 1–7, **2018**. doi:10.1109/GLOCOM.2018.8647375.

[30] Peter Miksa Hell and Peter Janos Varga. Drone Systems for Factory Security and Surveillance. *Interdiscip. Descr. Complex Syst.*, 17(3):458–467, **2019**. ISSN 1334-4684. doi:10.7906/indecs.17.3.4.

[31] Quanfeng Duan, Zhenghe Liang, and Limin Wang. Efficient File Sharing Scheme Based on WebRTC. (Ic3me):730–734, **2015**. doi:10.2991/ic3me-15.2015.142.

[32] Shuai Zhao, Zhu Li, and Deep Medhi. Low delay streaming of DASH content with WebRTC data channel. *2016 IEEE/ACM 24th Int. Symp. Qual. Serv. IWQoS 2016*, (January 2018):3–5, **2016**. doi:10.1109/IWQoS.2016.7590414.

[33] Khanista Namee and Ghadeer Mohsen Albadrani. Applying IoT, web real-time communication and cloud computing to maximize emergency medical service

(EMS) efficiency. *PervasiveHealth Pervasive Comput. Technol. Healthc.*, pages 115–119, **2019**. ISSN 21531633. doi:10.1145/3361758.3361777.

[34] Agnieszka Chodorek, Robert R. Chodorek, and Krzysztof Wajda. Media and non-media WebRTC communication between a terrestrial station and a drone: The case of a flying IoT system to monitor parking. *Proc. - 2019 IEEE/ACM 23rd Int. Symp. Distrib. Simul. Real Time Appl. DS-RT 2019*, pages 1–4, **2019**. doi:10.1109/DS-RT47707.2019.8958706.

[35] Agnieszka Chodorek, Robert Ryszard Chodorek, and Alexander Yastrebov. Weather sensing in an urban environment with the use of a uav and webrtc-based platform: A pilot study. *Sensors*, 21(21):1–20, **2021**. ISSN 14248220. doi:10.3390/s21217113.

[36] leaflet. https://leafletjs.com/, **2022**.

[37] Tymoteusz Horbiński and Dariusz Lorek. The use of Leaflet and GeoJSON files for creating the interactive web map of the preindustrial state of the natural environment. *J. Spat. Sci.*, 00(00):1–17, **2020**. ISSN 1449-8596. doi:10.1080/14498596.2020.1713237.

[38] Sumanth Reddy Enigella and Hamid Shahnasser. Real Time Air Quality Monitoring. *2018 10th Int. Conf. Knowl. Smart Technol. Cybern. Next Decad. KST 2018*, pages 182–185, **2018**. doi:10.1109/KST.2018.8426102.

[39] Edsel Matt O. Morales, Jojene R. Santillan, and Meriam Makinano-Santillan. Near-real time rainfall monitoring in the Caraga Region, Mindanao, Philippines using openlayers API and Javascript. *ACRS 2015 - 36th Asian Conf. Remote Sens. Foster. Resilient Growth Asia, Proc.*, (October), **2015**.

[40] Yuqin He, Zhenbo Bi, Haobin Tian, Kai Duan, Jiashuai Wu, and Hongwei Wang. Application of OpenLayers in marine information monitoring. *E3S Web Conf.*, 118:0–4, **2019**. ISSN 22671242. doi:10.1051/e3sconf/201911803006.

[41] Kanishk Chaturvedi. Web based 3D analysis and visualization using HTML5 and WebGL. (April):2–7, **2014**.

[42] Wenwen Li and Sizhe Wang. PolarGlobe: A web-wide virtual globe system for visualizing multidimensional, time-varying, big climate data. *Int. J. Geogr. Inf. Sci.*, 31(8):1562–1582, **2017**. ISSN 13623087. doi:10.1080/13658816.2017.1306863.

[43] A. B. Cahyono and R. A. Zayd. Rapid mapping of landslide disaster using UAV-photogrammetry. *J. Phys. Conf. Ser.*, 974(1), **2018**. ISSN 17426596. doi:10.1088/1742-6596/974/1/012046.

[44] Witold Abramowicz. *Lecture Notes in Business Information Processing: Preface*, volume 87 LNBIP. **2011**. ISBN 9783642218293.

[45] Aakash Sehrawat, T. Anupriya Choudhury, and Gaurav Raj. Surveillance drone for disaster management and military security. *Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017*, 2017-Janua:470–475, **2017**. doi:10.1109/CCAA.2017.8229846.

[46] The Cascaded-mtl, Current German Regulation, Proposed Method, and Scale-adaptive Crowd Detection Following. SCALE-ADAPTIVE REAL-TIME CROWD DETECTION AND COUNTING FOR DRONE IMAGES Markus K Â¨ uchhold , Maik Simon , Volker Eiselein and Thomas Sikora Communication Systems Group , Technische Universit Â¨ at Berlin. pages 943–947, **2018**.

[47] Sayani Sarkar, Michael W. Totaro, and Khalid Elgazzar. Intelligent drone-based surveillance: application to parking lot monitoring and detection. (November):3, **2019**. ISSN 1996756X. doi:10.1117/12.2518320.

[48] Zainab Zaheer, Atiya Usmani, Ekram Khan, and Mohammed A. Qadeer. Aerial surveillance system using UAV. *IFIP Int. Conf. Wirel. Opt. Commun. Networks, WOCN*, 2016-Novem, **2016**. ISSN 21517703. doi:10.1109/WOCN.2016.7759885.

[49] B. Coifman, M. McCord, R. G. Mishalani, M. Iswalt, and Y. Ji. Roadway traffic monitoring from an unmanned aerial vehicle. *IEE Proc. Intell. Transp. Syst.*, 153(1):11–20, **2006**. ISSN 17480248. doi:10.1049/ip-its:20055014.

[50] Iván Santos-González, Alexandra Rivero-García, Jezabel Molina-Gil, and Pino Caballero-Gil. Implementation and analysis of real-time streaming protocols. *Sensors (Switzerland)*, 17(4), **2017**. ISSN 14248220. doi:10.3390/s17040846.

[51] Dooyeol Yun and Kwangsue Chung. DASH-Based Multi-View Video Streaming System. *IEEE Trans. Circuits Syst. Video Technol.*, 28(8):1974–1980, **2018**. ISSN 10518215. doi:10.1109/TCSVT.2017.2690958.

[52] Yin Qun and Zhang Jianbo. Development of remote video monitoring system based on TCP/IP. *10th Int. Conf. Comput. Sci. Educ. ICCSE 2015*, (Iccse):596–600, **2015**. doi:10.1109/ICCSE.2015.7250316.

[53] Kundan Singh and Henning Schulzrinne. Unified Messaging using {SIP} and {RTSP}. *IP Telecom Serv. Work.*, (November 2000):31–37, **2000**.

[54] Faculty Of and Information Technology. BRNO UNIVERSITY OF TECHNOLOGY DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA Bachelor ' s Thesis Specification. **2021**.

[55] Yong Liu, Yang Guo, and Chao Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Netw. Appl.*, 1(1):18–28, **2008**. ISSN 1936-6442. doi:10.1007/s12083-007-0006-y.

[56] Ming Zhang, Chen Su, Yuan Liu, Mingyuan Hu, and Yuesheng Zhu. *Unmanned aerial vehicle route planning in the presence of a threat environment based on a virtual globe platform*, volume 5. **2016**. ISBN 1501213962. doi:10.3390/ijgi5100184.

[57] Liangfeng Zhu, Xin Pan, and Gongcheng Gao. Assessing Place Location Knowledge Using a Virtual Globe. *J. Geog.*, 115(2):72–80, **2016**. ISSN 17526868. doi:10.1080/00221341.2015.1043930.

[58] Nenad Višnjevac, Rajica Mihajlović, Mladen Šoškić, Željko Cvijetinović, and Branislav Bajat. Prototype of the 3D cadastral system based on a NoSQL database and a Javascript visualization application. *ISPRS Int. J. Geo-Information*, 8(5), **2019**. ISSN 22209964. doi:10.3390/ijgi8050227.

[59] Richard G. Donohue, Carl M. Sack, and Robert E. Roth. Time series proportional symbol map with leafet and jquery. *Cartogr. Perspect.*, 76(January):43–66, **2013**. ISSN 10489053. doi:10.14714/CP76.1248.

[60] Imas Sukaesih Sitanggang, Asep Rahmat Ginanjar, Muhamad Syukur, Rina Trisminingsih, and Husnul Khotimah. Integration of spatial online analytical processing for agricultural commodities with OpenLayers. *ICECOS 2017 - Proceeding 2017 Int. Conf. Electr. Eng. Comput. Sci. Sustain. Cult. Herit. Towar. Smart Environ. Better Futur.*, pages 167–170, **2017**. doi:10.1109/ICECOS.2017.8167127.

[61] Xujie Kang, Jing Li, and Xiangtao Fan. Spatial-temporal visualization and analysis of earth data under cesium digital earth engine. *ACM Int. Conf. Proceeding Ser.*, pages 29–32, **2018**. doi:10.1145/3289430.3289447.

[62] Klokan Firm. https://www.klokantech.com/company/, **2022**.

[63] Mátyás Gede. Using Cesium for 3D Thematic Visualisations on the Web. *Proc. ICA*, 1:1–4, **2018**. doi:10.5194/ica-proc-1-45-2018.

[64] M. Buyukdemircioglu and S. Kocaman. Geovisualization of aerial photogrammetric flights for data quality assessment. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, 43(B4-2021):333–338, **2021**. ISSN 16821750. doi:10.5194/isprs-archives-XLIII-B4-2021-333-2021.

[65] Dilek Tezel, Mehmet Buyukdemircioglu, and Sultan Kocaman. Accurate assessment of protected area boundaries for land use planning using 3D GIS. *Geocarto Int.*, 36(1):96–109, **2021**. ISSN 10106049. doi:10.1080/10106049.2019.1590466.

[66] Gianfranco Forlani, Fabrizio Diotri, Umberto Morra di Cella, and Riccardo Roncella. Indirect UAV strip georeferencing by on-board GNSS data under poor satellite coverage. *Remote Sens.*, 11(15), **2019**. ISSN 20724292. doi:10.3390/rs11151765.

[67] O. Mian, J. Lutes, G. Lipa, J. J. Hutton, E. Gavelle, and S. Borghini. Direct georeferencing on small unmanned aerial platforms for improved reliability and accuracy of mapping without the need for ground control points. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, 40(1W4):397–402, **2015**. ISSN 16821750. doi:10.5194/isprsarchives-XL-1-W4-397-2015.

[68] M. Rabah, M. Basiouny, E. Ghanem, and A. Elhadary. Using RTK and VRS in direct geo-referencing of the UAV imagery. *NRIAG J. Astron. Geophys.*, 7(2):220–226, **2018**. ISSN 2090-9977. doi:10.1016/j.nrjag.2018.05.003.

[69] M Muthumalathi. REVIEW ON INFORMATION OF TELEMETRY SYSTEM FOR INDUSTRIAL APPLICATIONS IN WIRELESS SENSOR NETWORKS. XI(1217):1217–1221, **2019**.

[70] dji developer. https://developer.dji.com/onboard-sdk/documentation/guides/component-guide-telemetry.html, **2022**.

[71] DJI Mavic Pro. https://www.dji.com/mavic, **2022**.

[72] Chintan P.Dave, Rahul Joshi, and S. S. Srivastava. A Survey on Geometric Correction of Satellite Imagery. *Int. J. Comput. Appl.*, 116(12):24–27, **2015**. doi:10.5120/20389-2655.