

Facilitating Real-time Road Event Detection on Collaborative UAV and Ground Vehicle

by

Kiran Davuluri

A thesis in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan–Dearborn
2024

Master's Thesis Committee:

Assistant Professor Zheng Song, Chair
Dr. Jinzhu Chen, General Motors Company
Assistant Professor Probir Roy

Kiran Davuluri

kirandav@umich.edu

ORCID iD: [0000-0003-0991-0142](https://orcid.org/0000-0003-0991-0142)

© Kiran Davuluri 2024

Dedication

This work is dedicated to all those who have supported me throughout my journey. My deepest gratitude goes to my parents, Indira Davuluri and D N V Satyanarayana, my elder brother Varun Davuluri.

Acknowledgements

Throughout my academic journey, I have been fascinated to contribute to research initiatives and projects, even in a small way. It was Dr. Zheng Song, my advisor who fanned the little spark in me since the inception of my master's and provided me with an amazing opportunity to join and contribute to projects at Pervasive Computing Lab. I am deeply grateful to him for his unwavering support and guidance throughout my Master's journey. At Percom Lab, Prof. Song has always given me the freedom to express ideas and develop them into novel solutions. His encouragement and support in difficult times have immensely helped me to overcome my limitations and evolve as a better being. His contributions have been invaluable, and this work would not have been possible without his guidance. His contributions have been greatly appreciated and have played a crucial role in my growth and success. Furthermore, I would also like to express my sincere gratitude to Dr. Probir Roy and Dr. Jinzhu Chen for giving me the opportunity to share my research with the committee and the wider community. Dr. Roy's feedback and insights have been invaluable in helping me to improve my work. Finally, this work wouldn't have been possible without my co-authors, especially my labmates Summit, Khairul, and Zhenquan. Thank you all for being a part of my journey.

Preface

This work is at the intersection of developing novel solutions for road safety and future mobility. Despite many challenges, I have always been fascinated by looking at undefined boundaries and searching for solutions. Motivated by the recent technological advancements and the greater scope for new avenues in this area, I wanted to make meaningful contributions that can help the society in large.

I sincerely hope this work can help underscore the need for novel distributed frameworks and middleware support for future mobility systems. Our work hopes to help further initiatives in the direction of event detection and road safety.

I am profoundly grateful for all that I have learned along the way and for the invaluable contributions of those who have joined me on this journey. The support from my colleagues, the resources provided by my institution, and the collaborative spirit of my peers have all played a crucial role in bringing this work to fruition. This experience has not only enriched my knowledge but also strengthened my resolve to continue contributing to the ever-evolving domain of road safety and vehicular transportation systems.

Table of Contents

Dedication	ii
Acknowledgements	iii
Preface	iv
List of Figures	vii
List of Tables	viii
Abstract	ix
Chapter	
1 Introduction	1
1.1 Overview	1
1.2 Existing event detection and sensing technologies	2
1.3 Event Detection on Drones	3
1.4 Problem Statement & Solution Overview	4
1.5 Contributions	5
1.6 Structure of the thesis	5
2 Drone-Car collaboration for Future Mobility: Our Vision	6
2.1 Real-time Drone-Car Collaboration	6
2.2 Enabling Technologies	7
2.3 Applications in Future Mobility	8
2.3.1 Enhancing Driving Safety	9
2.3.2 Improving Fuel Efficiency	10
2.3.3 Car-mounted-Drone-as-a-Service (CDaaS)	10
3 Background & Motivation	12
3.1 Drone as a Service	12
3.2 Existing Work and Opportunity	15
3.2.1 Real-time Image Processing on Drones	15
3.2.2 Motivating Examples and Datasets	16
3.2.3 Regions of Interests	17
3.3 Empirical Study	18
4 GRIFFIN Design	22

4.1	Overview	22
4.2	Challenges	23
4.3	GRIFFIN Overview	23
4.3.1	Generic Workflow for Distributed Partition Processing	24
4.3.2	Utility-Aware Partitions Distribution	25
4.3.3	Programming support	29
5	Reference Implementation	33
5.1	Runtime Workflow	33
5.2	Fine-tuning CV modules	34
5.3	Programming Aspects	35
6	Evaluations	41
6.1	Testbed Implementation	41
6.2	Evaluations on event applications	41
6.3	Evaluation of unit modules	43
6.3.1	Fine-tuning CV modules for enhanced detections	44
6.3.2	Programming modules and toolchain	44
6.3.3	Abstractable Components	46
7	Discussion	48
8	Future Directions	50
8.1	Dataset and Testbed	50
8.2	CV algorithms for drone computations	52
8.3	Advanced Sensing and Efficient Communication	53
9	Conclusion	55
	Appendix: Relevant Paper Publications	56
A.1	Griffin: Facilitating Real-time Road Event Detection on Collaborative UAV and Ground Vehicle	56
A.2	Real-time Drone-Car Collaboration for Future Mobility: Vision and Challenge	58
A.3	BMEye: Public Health-Oriented Body Mass Index Monitoring using Commodity WiFi	59
A.4	SWiDir: Enhancing Smartphone-based Walking Direction Estimation with Passive WiFi Sensing	60
	References	61

List of Figures

1.1	Sensing Range of Drones and Ground Vehicles	4
2.1	Two Typical Architectures in Drone-Car Collaboration	8
2.2	Drone assistive event detection and hazard avoidance	9
3.1	Emergence of Commercial Drones [1]	12
3.2	Speeding Detection	17
3.3	Collision Avoidance	17
3.4	Illustration of partitioning-based approach	17
3.5	Our empirical study shows that partitioning-based approach suits best according to the trade-off between QoS-metrics and drone's limited battery	19
4.1	Illustration of GRIFFIN Workflow	22
4.2	Generic Workflow of GRIFFIN	24
4.3	GRIFFIN SDK	31
4.4	Processing modules in drone and car executables	32
5.1	System Components of GRIFFIN Prototype	33
5.2	Example code highlighting abstractions in cache API	38
5.3	Code snippet sample for an application's Programming Interface	40
6.1	Evaluation on varying Network Performance on two event applications: Collision and Lane Changing	42
6.2	Trade-off between Utility, Accuracy and Latency while detecting the two events	43
6.3	CV Modules Performance	44
6.4	Drone's energy w.r.t Partitions Distribution	45
6.5	Latency w.r.t Partitions Distribution	45

List of Tables

5.1	Generic and Application based CV algorithms	35
6.1	Packet Loss for varying network statuses	42
6.2	Evaluation on toolchain	46
6.3	Evaluation metrics for abstractions implemented for GRIFFIN	46

Abstract

Smart road systems have undergone significant advancements, integrating technologies like the Internet of Things, Artificial intelligence, and big data analytics to enhance the efficiency, safety, and sustainability of transportation networks. Despite these developments, road accidents remain a pressing issue, underscoring the need for advanced road event detection systems. Traditional vehicle-mounted sensors (e.g., Lidar, Radar, and stationary cameras) each have limitations, such as restricted sensing ranges and high costs. Drones, with their flexible mobility and extended sensing capabilities, offer a promising solution for road event detection. However, their limited battery capacity and computational resources necessitate distributing the computing to external resources.

This thesis proposes a novel approach for real-time road event detection through a collaborative system combining drones and ground vehicles (GVs). By leveraging the computational capabilities of GVs and the aerial advantages of drones, this framework aims to deliver fast, accurate, and energy-efficient event detection. Key contributions include the design and implementation of a distributed middleware system, GRIFFIN, facilitating drone-car collaboration for real-time road event detection, and empirical evaluations demonstrating its effectiveness. The work concludes with a discussion of challenges and future directions for further enhancing drone-based road event detection systems.

CHAPTER 1

Introduction

1.1 Overview

Over the past decade, smart road systems have evolved significantly, integrating advanced technologies to create more efficient, safe, and sustainable transportation networks. These systems leverage the Internet of Things (IoT), artificial intelligence (AI), and big data analytics to monitor and manage road conditions in real time. Innovations such as intelligent traffic lights, connected vehicle systems, and smart sensors have been deployed to optimize traffic flow, reduce congestion, and enhance road safety. Additionally, smart road systems are being designed and built to support the development of autonomous vehicles by providing crucial infrastructure for vehicle-to-everything (V2X) communication. This enables vehicles to interact with each other and with road infrastructure, facilitating smoother and safer driving experiences. We can see that the adoption of smart road and safety systems has also been driven by the need for more sustainable urban environments, with technologies like dynamic tolling and adaptive traffic management helping to reduce emissions and energy consumption. The ongoing advancements of smart road systems promise to play a critical role in shaping the status of future mobility.

At the same time, ground vehicles (GVs) have played a significant role in building these systems. They have evolved from traditional manually driven models to those equipped with advanced driver assistance systems. This highlights the transition from conventional vehicles to those with modern technologies like autonomous driving. In addition to that, these days ground vehicles are mounted with sensor capabilities that can sense the vehicle's surroundings. For e.g. cruise control sensors, parking assistance, traffic sign recognition sensors, vision-sensors, GPS sensors, proximity-sensors, etc.

While we see advancements in road safety systems and ground vehicles each day, statistics show that around 6 Million Vehicle Accidents take place in the US alone each year, taking around 45k lives on average.

In the United States, over 300 million vehicles ply on roads, each consuming an average of \$150 worth of gasoline per month. The sheer volume of vehicles also underscores the importance

of efficient road safety measures. One critical aspect is the need for robust road event detection systems, which can help identify and respond to incidents such as accidents, traffic congestion, and hazardous conditions. By leveraging advanced technologies like sensors, cameras, and small-scale computational devices, these systems can provide real-time information to drivers and traffic authorities. This can lead to safer, more efficient travel and secure driving decisions on roads.

These new-generation Ground Vehicles (GVs) hold the potential for enhanced convenience and safety but still struggle with challenges in handling unpredictable scenarios. This can primarily be attributed to the limited sensing range of car-mounted sensors. In particular, the limited sensing range in GVs can fail to detect events at a longer range [2]. For instance, the vision sensor of a ground vehicle can be obstructed by various objects such as other vehicles, trees, or Road Side Units (RSUs). This obstruction may impede the timely detection of critical road events that require prompt attention. Consequently, the restricted sensing capacity provides drivers and driver assistance systems with less time and accurate information to respond. Also, autonomous driving systems have high risk for being invaded for privacy and other security concerns. On the other hand, alternative detection methods such as Lidar, stationary-camera sensing, etc come with their own cons that prove expensive and in-accurate. These problems can be alleviated by lightweight aerial mobility systems such as Drones. 1) They provide flexible mobility and extended sensing viewpoints in contrast to car-based sensors. 2) their on-board devices can be used for computations in real-time. 3) They are added up as surveillance and security monitoring devices for ground vehicles.

However, drones have limited battery capacity for computations and other important tasks. This proves essential to seek more robust computing resources that can provide more accurate and sustainable results. Therefore, The above factors motivate for a robust system that detects road events combining lightweight external sensing capabilities and additional computing resources.

1.2 Existing event detection and sensing technologies

Understanding the necessity of building efficient event detection systems on roads is crucial for enhancing safety and driving decisions. Several existing approaches are employed to achieve this goal, each with unique benefits and limitations. Below, we explore these approaches in detail, including vehicle-based sensors, advanced detection technologies, stationary monitoring units, and aerial surveillance systems.

Firstly, **car-mounted sensing technologies** include sensors like cameras, GPS, and accelerometers installed on vehicles. These sensors collect real-time data on vehicle behavior, road conditions, and traffic flow, which can be used to detect events like accidents or sudden braking. However, they are limited by the need for widespread adoption and consistent data sharing across vehicles, and

they may not cover all areas, especially in rural or less-trafficked regions.

Secondly, **Lidar, Radar, and Infrared technologies** are essential for detecting objects and obstacles with high precision. Lidar provides detailed 3D maps of the surroundings. Radar is effective in various weather conditions, and Infrared sensors can detect heat signatures, making them useful for spotting pedestrians or animals at night. Despite these advantages, these technologies can be expensive to deploy and maintain, and they may have reduced effectiveness in adverse weather conditions or highly reflective environments. These methods are often based on movements of objects, etc that often may give inaccurate results.

Thirdly, **stationary camera sensing at Roadside Units (RSUs)** involves installing cameras at strategic locations such as intersections and highways. These cameras provide continuous monitoring and are crucial for detecting incidents like traffic jams or road obstructions. However, their coverage is limited to specific areas, and they require substantial infrastructure investment. Moreover, they may face challenges in real-time data processing and integration with other systems, and they are vulnerable to vandalism or environmental factors.

Above approaches come with their limitations while being useful for specific use-cases. Drones, on the other hand, provide additional benefits that possess to assist in event detections. At the same time, drones need to be compensated with their shortcomings that is discussed next.

1.3 Event Detection on Drones

Modern drones such as DJI and Pixhawk [3] offer advantages to be deployed for realistic scenarios due to their lightweight, inexpensive, and easy-to-deploy nature, allowing for rapid response and coverage of large areas. They can provide timely warnings by quickly identifying and relaying critical information with reduced response times. The flexible viewpoints with high-quality sensing capabilities and mobility of drones enable them to capture comprehensive views of the scene, ensuring accurate and detailed data capturing. Additionally, drones can assist in computations and communications with ground systems, integrating seamlessly with existing systems to enhance data processing and analysis capabilities. Their adaptability allows for easily configurable modifications with ground station devices in terms of network contacts, data transfer, etc., making them a versatile tool for various event detection and monitoring needs.

Given the advantages of drones, implementing road event detection applications on drones still faces significant challenges, mainly caused by the inconsistency between the high demand for computing and energy resources of computer vision algorithms and the poor resource supply of drones. In order to provide timely guidance on the GV's micro-level driving decisions, the event detection application needs to be executed in a real-time fashion. Nevertheless, existing drone-based event detection systems fail to satisfy such real-time requirements: 1) many systems process images

in an offline fashion [4]; 2) some systems offload the captured images/videos to edge servers, which may cause high latency due to the fluctuating network conditions; 3) some systems reduce the resolution of the captured video/image, which causes low event detection accuracy.

Stand-alone small-scale drones equipped with cameras and sensors to monitor specific areas from the air. These drones are particularly used for small-scale operations such as vehicle detections, real-time image feed, or surveying areas quickly, such as during emergencies or large public events, providing a bird’s-eye view. These operations do not necessarily include real-time computations with external ground resources. They are helpful in relaying information to ground stations for manual verifications. However, drones have limited flight time and range, and their use is restricted by regulations, weather conditions, and potential privacy concerns. Additionally, they require skilled operators and may not be able to provide continuous monitoring. Moreover, the drone’s flying status makes it infeasible to increase the computing and energy resources on drones for our tasks.

1.4 Problem Statement & Solution Overview

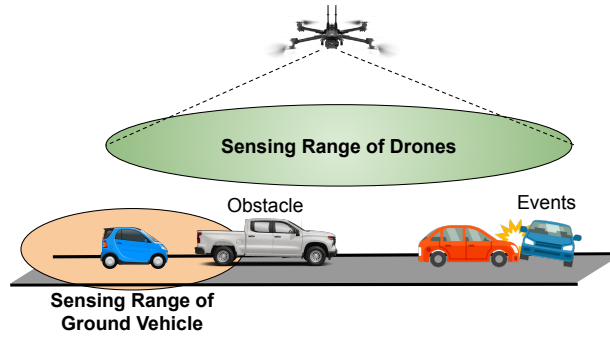


Figure 1.1: Sensing Range of Drones and Ground Vehicles

As depicted in figure 1.1, drones offer broader view range on roads to capture multiple vehicles at the same instance and their interactions. As discussed previously, they show additional capabilities making them favourable for event detections than existing alternatives. However, drones come with their own limitations. This underscores the need for a real-time road event detection system that can effectively mitigate those limitation, and satisfy the urgent demand for making drone-view based road-event detection fast, accurate, and energy efficient. Our work aims to develop a novel road event detection framework by taking the advantage of drone-based sensing. This work also explores the possibility for capturing images from drones and utilizing external resources to process them simultaneously to make detections fast, accurate and reduce drone’s processing energy.

Since modern GVs have capabilities to 1) host servers that can process data and conduct robust computations, 2) communicate with drones and 3) can be followed by the latter at comparable speeds, utilizing them for processing data and communications proves to be a suitable solution in realistic scenarios for our problem.

1.5 Contributions

In particular, this thesis makes the following key contributions:

- Based on the limitations of existing event detection approaches and the opportunity to balance drones' energy and resource constraints with the car's supporting capacity as a processing facility, we envision and introduce the concept of Drone-Car collaboration for Real-time Road Event Detection.
- We study the existing techniques for drone-based image processing. Further, we conduct an empirical study that compares these processing mechanisms.
- Motivated by the study, we design and implement GRIFFIN, a distributed middleware system comprising a robust runtime and programming support that facilitates the development of drone-car collaborative road event detection applications.
- We evaluated two event applications using GRIFFIN. Our evaluations show that GRIFFIN achieves optimal for efficiently detecting event applications in real-time. Finally, we conclude our work by discussing the challenges and key takeaways that can taken up as future directions.

1.6 Structure of the thesis

The rest of the thesis work is organized as follows: Chapter 2 presents our vision for building a framework for road event detections. Motivated by the work in Chapter 3, Chapter 4 details the design of GRIFFIN. Chapter 5 provides the details on the reference implementation of GRIFFIN's prototype. Further, Chapter 6 highlights the results and observations from our evaluations using two event applications. Finally, we conclude the work in Chapter 9 by discussing the challenges and potential future directions in Chapter 7 and Chapter 8 respectively.

CHAPTER 2

Drone-Car collaboration for Future Mobility: Our Vision

In this chapter, we look at the possibility of collaboratively processing on drones and external resources such as cars that can be of convenience and support for event detections to meet the desired goals. In this regard, we study the enabling technologies and pertinent factors that motivate us to decide on the drone-car collaborative framework for detecting road events in real-time.

2.1 Real-time Drone-Car Collaboration

A century after Model T's invention [5], emerging technologies like autonomous driving, electric vehicle (EV), shared transportation push mobility into a new age of innovation. Vehicles are gradually transformed into movable computing hubs [6], equipped with decent sensing, networking, computing capabilities and battery power. Along with the improvements of hardware, Advanced Driver-Assistance System (ADAS) applications [7] has been rapidly iterating towards enhancing operability, safety, and fuel efficiency and realizing the goals of future mobility, i.e., convenient [8], zero death [9], and carbon neutrality [10].

As seen in the previous chapter, ADAS relies heavily on various sensors to capture the surrounding context of the vehicles. All of the vehicle's sensors are integrated within its structure, which limits their sensing capabilities. However, these sensors tend to limited ranges often impeding the accuracy of information. Consequently, the restricted sensing capacity provides drivers and driver assistance systems with less time to respond, limiting the potential applications and effectiveness of ADAS.

Utilizing Unmanned Aerial Vehicles (UAVs or drones) can effectively mitigate this limitation. Drones, equipped with robust hardware capabilities and aerial mobility, can gather data from specific locations and angles, providing more flexible viewpoints and requiring low maintenance cost. The advanced cameras and sensors onboard these UAVs deliver detailed sensing information, enabling drones to offer a comprehensive and global perspective of fine-grained observations.

However, drones face challenges such as limited flight duration, restricted ranges, and constrained computational capacities. Conversely, cars possess greater computational capabilities than

drones but are hindered by limited fields of view, sensing ranges, and restricted mobility. As it emerges from previous points the need for additional processing capabilities, certain alternatives such as transmitting data to RSUs or base stations on ground hold for good suitability. In such cases, there are high chances for extended processing and transmitting delays due to the distance and network connectivity between drones and ground stations. RSUs can be availed for on-road information verification or for corroboration of fine-grained details.

The inherent strengths and weaknesses of both drones and cars have given rise to an emerging distributed system paradigm which we coin as “**Real-time Drone-Car Collaboration**”, where drones and cars form a sensing and computing continuum to collaboratively perform real-time tasks. This new paradigm holds promise for enhancing driving safety, fuel efficiency, and enabling innovative applications. We envision for a robust paradigm with real-time drone-car collaboration that can enhance future mobility. This paradigm is introduced through the description of its “sensing-processing continuum” system model and workflow. We further discuss the technologies that impact the paradigm and future mobility applications that would benefit from this paradigm.

Although real-time collaboration can fundamentally change how ADAS systems collect and utilize context, fully realizing this paradigm demands years of cross-disciplinary efforts in distributed systems, computer vision, wireless communication, software engineering, and middleware.

2.2 Enabling Technologies

The global drone market is projected to reach over \$42 billion by 2025 [11]. This growth is driven by advancements in drone technology and increased applications across various sectors. As industries increasingly integrate drone technology for varied applications—from agriculture and surveying to delivery and emergency services—the capabilities of these aerial devices continue to evolve. The following sections highlight three key advances: improved drone capabilities, integration with ground vehicles, and advanced sensor technologies, showcasing a shift towards a more interconnected technological ecosystem.

1) Advancements in Drone: Recent advancements in drone technology have led to rapid growth, encompassing both hardware and software, with a projected compound annual growth rate (CAGR) of 57.5% from 2021 to 2028. Modern commercial drones available off the shelf can achieve speeds up to 60mph, operate in challenging weather conditions such as strong winds, snow, and rain. Typically, the battery life of these drones ranges from 30 minutes to one hour, enabling extended flight times. While the software of many drones is not open source, they feature robust communication systems for control and video transmission, alongside software capabilities like object tracking and obstacle avoidance, that are supported by powerful hardware. These developments facilitate integration with ground vehicles for enhanced data collection and processing.

2) Drone-ready Ground Vehicles: Ground vehicles now serve as docking stations for drones [12], equipped with extensive charging capacities ranging from 50 to 200 kWh and mechanical designs to support drone landing, flying, and recharging. Major automakers like Ford, General Motors (GM), Toyota, and Honda are integrating hardware for drone operations. For example, Ford already has filed patents for a vehicle-mounted aerial drone container [13], a drone close-proximity operating system [14], a UAV integration system, and a UAV sanitation system. The transformation is expected to increase a vehicle’s utility in drone operations by as much as 75%, enhancing its role beyond mere transportation. Additionally, the advanced computing resources and ample energy supplies of ground vehicles enable them to function as mobile command centers or data processing hubs for drones, thereby supporting extended missions and facilitating complex computations.

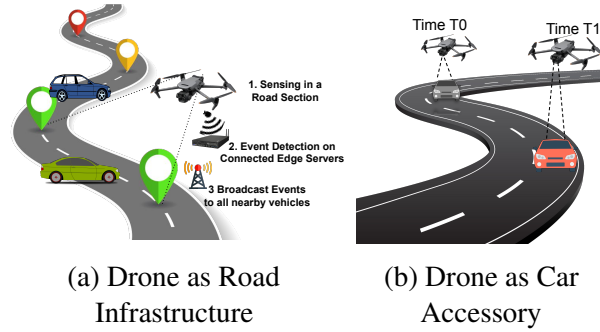


Figure 2.1: Two Typical Architectures in Drone-Car Collaboration

3) Sensor Technologies: Drones and ground vehicles rely on various sensors to perceive their surroundings. Video cameras are commonly used for object detection, while rangefinders and LiDAR (Light Detection and Ranging) [15] provide supplementary data. In situations where video analysis is inadequate, RFID (radio frequency identification) tags help locate nearby objects [16]. Additionally, specialized sensors like metal detectors and x-ray machines can be added as security for specific applications[17].

2.3 Applications in Future Mobility

In this section, we present a range of applications that vividly showcase the potential of drone-car collaboration, emphasizing its transformative impact. These applications serve as compelling examples that exemplify our vision of the powerful synergy between drones and cars, illustrating how their collaboration can revolutionize future mobility and unlock new possibilities on enhancing driving safety, improving fuel efficiency and providing various services.

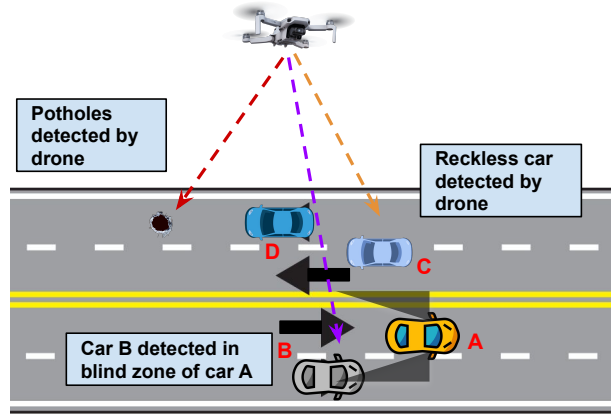


Figure 2.2: Drone assistive event detection and hazard avoidance

2.3.1 Enhancing Driving Safety

Car accidents can result from factors like reckless driving, inexperience, speeding, adverse weather, distracted driving, and impaired driving [18]. Additionally, limited car visibility can cause issues such as blind spots, difficulty detecting small objects in low light or adverse weather, and hidden hazards. By collaborating with cars in real-time, drones can enhance driving safety and help mitigate these risks.

Here we use some example applications to demonstrate how drones can enhance driving safety, improve fuel efficiency, and provide on-demand drone services, as depicted in Fig. 2.2.

1) Road Condition Detection: Road hazards, such as black ices, water accumulation, and potholes may be hard to detect from the vehicle's point of view. Many times the drivers don't have enough time to react when a hazard is detected. From a bird-eye's view, a drone can apply CV algorithms to detect such events when the ground vehicle is a distance away, and sends early warning to the vehicle or nearby vehicles.

2) Driving Behavior Detection: Drones can also identify reckless and speeding vehicles, anticipate road accidents ahead, as well as objects in the blind spots of the ground vehicle. While existing car-mounted sensors can also detect road events (e.g., vehicles in the blind spot, emergence stops), drones provides additional viewpoints and can capture distant events, giving drivers more time to react.

3) Abnormal Vehicle Status Detection: The ground vehicle itself can suffer from part failures and operational defects. Some examples include the driver forgets to turn on the light at night or the light is too dim. Drones have the capability to detect such abnormal statuses from outside the vehicle and promptly send alerts.

2.3.2 Improving Fuel Efficiency

The way vehicles are driven directly impacts fuel consumption, emissions, and emission control effectiveness [19]. Aggressive driving worsens these issues by inefficiently using emission control systems and releasing excessive pollutants [20]. Drones collaborating with cars enhance fuel efficiency through real-time data provision, performance analysis, and energy-saving recommendations, while optimizing routes, promoting eco-friendly behaviors, and assisting in maintenance to reduce fuel consumption, lower emissions, and foster a sustainable transportation system.

4) Energy-Efficient Routing: Drones can analyze various factors, including traffic patterns, road conditions, and energy consumption models of cars, to suggest energy-efficient routes. Furthermore, through real-time monitoring of traffic conditions, drones can identify congested areas and traffic slowdowns, enabling the selection of more optimal routes that bypass congestion and minimize idle time. This proactive approach not only enhances fuel efficiency but also helps relieve traffic congestion.

5) Micro-level Driving Decision Optimization: Drones can sense the traffic cameras and congestion ahead of the ground vehicle, and provide micro-level driving suggestions to improve fuel efficiency. For example, the drone could suggest the car to speed up a bit and avoid waiting for the red light, or suggest the car to maintain a low speed and avoid unnecessary accelerations and breaking.

2.3.3 Car-mounted-Drone-as-a-Service (CDaaS)

Car-mounted-Drone-as-a-Service refers to a service model where drones are mounted on cars, typically special-purpose vehicles, to provide on-demand drone services. In this model, the car serves as a platform for deploying and operating the drone, allowing for greater mobility, flexibility, and accessibility of drone capabilities. With Car-mounted-Drone-as-a-Service, a variety of drone-based services can be offered directly from the vehicle. The drone can be launched and controlled remotely from the car or autonomously, providing aerial capabilities for tasks such as aerial photography, videography, surveying, mapping, inspection, delivery, and more. CDaaS has potential applications in various industries as well, including emergency services, transportation and logistics, construction, agriculture, infrastructure inspection, and entertainment.

6) Last-Mile Delivery: Drone-based last-mile delivery refers to the utilization of drones for the transportation of packages in the final leg of the delivery process [21]. Various approaches exist for implementing drone-based last-mile delivery, including standalone drone delivery, separate operations of drones and trucks, and collaborative drone-truck delivery [22]. Drones can also be used in fully autonomous delivery systems by pairing them with a UGV [23, 24].

7) Law Enforcement: UAVs can support law enforcement in preventing crime and capturing

criminals. When dealing with terrorists, UAVs can be used for detection of explosives and weapons in the area. UAVs equipped with metal detectors can scan a wide area for suspicious objects. When something suspicious is detected, a GPS signal can be transmitted (such as to a ground vehicle) and the area can be thoroughly searched by the drone using X-ray [17].

8) Accident Fast Reaction: In case of emergencies, a UAV can quickly reach the location and collect video and data. For instance, if sensors at an intersection suggest a high likelihood of a crash, a UAV can be dispatched to scan the area and relay data to emergency services [25]. This use of drones allows for rapid understanding and response to traffic accidents. Studies [26] show that drones can enhance emergency response, using their aerial view and GPS to locate people and coordinate rescue efforts in disaster zones.

CHAPTER 3

Background & Motivation

In this chapter, we discuss the background that formulates as the basis for our work. Firstly, we present an overview of the hardware and software components embedded in drones and also discuss the emergence of drones in the commercial space for various tasks, dubbed "Drone as a Service." Following that, we look at examples of motivating event applications captured using bird-eye view images. Further, we introduce existing approaches that can be employed for the real-time processing of drone-captured images. For several practical usage scenarios, we collected a set of images from real-world videos and discussed the potential benefits of our proposed partition-based approach. We further compare the performance of the partition-based approach with existing approaches, which motivates GRIFFIN, a partition-based distributed processing framework.

3.1 Drone as a Service

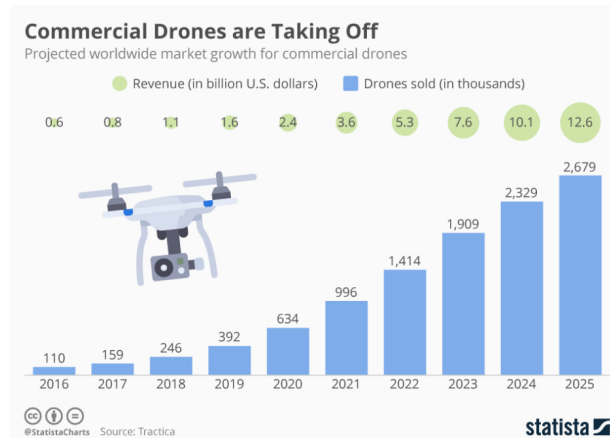


Figure 3.1: Emergence of Commercial Drones [1]

Hardware and battery components. Drones, or Unmanned Aerial Vehicles (UAVs), are advanced systems composed of various critical components and cutting-edge technologies, enabling a vast array of applications across numerous industries. The foundational element of a drone is its airframe, which provides the structural support necessary for housing all other components.

These airframes are crafted from materials such as plastic, carbon fiber, or metal, chosen for their durability and lightweight properties, which are crucial for flight efficiency. The propulsion system, integral to the drone's movement, includes motors, propellers, and electronic speed controllers (ESCs). These components work in unison to convert electrical energy into mechanical energy, generating the necessary lift and thrust. The flight controller, often considered the drone's main processing unit, plays a pivotal role by processing data from various onboard sensors to maintain balance, execute pilot commands, and perform autonomous functions.

Drones are typically powered by lithium-polymer (LiPo) batteries, known for their high energy density and lightweight properties, which are essential for maximizing flight time and payload capacity. Some advanced drones may also use alternative power sources like fuel cells for longer endurance. The onboard computational resources of drones have evolved significantly, with many now featuring onboard computers and Graphics Processing Units (GPUs) that handle complex tasks such as image processing, machine learning, and real-time data analysis. These computational systems enable drones to perform sophisticated functions like object recognition, path planning, and autonomous navigation.

Communication in drones. Communication modules are crucial for drone operations, facilitating the exchange of control signals and data between the drone and the ground control station (GCS). Radio transmitters and receivers operate on frequencies such as 2.4 GHz and 5.8 GHz, providing reliable communication for controlling the drone and receiving telemetry data. GPS and GNSS systems provide critical positional data, ensuring precise navigation, waypoint tracking, and geofencing capabilities. Telemetry systems relay real-time data such as speed, altitude, battery status, and system health to the operator, enabling informed decision-making during flights. Video transmission systems, often referred to as FPV (First-Person View) systems, allow operators to receive live video feeds from the drone's onboard cameras, facilitating precise control and situational awareness.

Sensors and payloads. Drones are equipped with a variety of sensors and payloads that expand their capabilities. Cameras, including RGB, thermal, and multispectral types, are commonly used for applications such as aerial photography, videography, inspection, and agricultural monitoring. Lidar sensors provide detailed 3D mapping capabilities that are useful in surveying, construction, and environmental monitoring. Infrared sensors detect heat signatures, aiding in search and rescue operations, firefighting, and wildlife monitoring. Additional sensors like ultrasonic, magnetometers, barometers, and airspeed sensors contribute to accurate altitude measurement, orientation detection, and flight stability.

Online flight sensing. Flight sensing data collected by these sensors is invaluable for monitoring and controlling drone performance. This data includes speed measurements obtained through GPS and airspeed sensors, which are crucial for adjusting flight dynamics and handling wind conditions.

Altitude data, derived from barometers, GPS, and ultrasonic sensors, ensures safe and precise navigation, particularly in complex environments. The drone's attitude, encompassing pitch, roll, and yaw angles, is monitored by gyroscopes and accelerometers, providing critical information for maintaining stability and executing complex maneuvers. Position and navigation data, delivered by GPS and GNSS systems, enable accurate mapping and waypoint navigation, which is essential for tasks like surveying and delivery.

Environmental data, such as temperature, humidity, and air pressure, can influence flight performance and sensor accuracy, highlighting the need for continuous monitoring. Battery status, including voltage, current, and temperature, is tracked to prevent power-related issues and ensure safe return-to-home (RTH) procedures. Control inputs and outputs data help diagnose and adjust the drone's response to pilot commands, enhancing control precision. Obstacle detection and avoidance systems, utilizing cameras, Lidar, or ultrasonic sensors, play a critical role in preventing collisions and ensuring safe flight paths, especially in complex environments or during autonomous operations. These methods prove expensive and difficult to deploy.

Computational resources in drones. Drones equipped with in-house edge computational resources represent a significant advancement in UAV technology, enabling them to perform complex data processing tasks directly onboard without relying on external servers or ground stations. These drones are fitted with powerful onboard processors, such as GPUs and specialized AI chips, which allow for real-time analysis and decision-making. This capability is crucial for applications requiring immediate responses, such as autonomous navigation, object detection, and obstacle avoidance. In-house edge computing enables drones to process high-resolution images, video streams, and sensor data on the fly, facilitating tasks like precision agriculture, where real-time analysis of crop health and soil conditions is necessary, or in emergency response scenarios, where rapid identification of hazards can save lives. By reducing the latency associated with data transmission and processing, drones with edge computing capabilities offer enhanced operational efficiency, improved data security, and greater autonomy, making them ideal for a wide range of industries, including agriculture, construction, public safety, and environmental monitoring. This technology not only streamlines operations but also expands the potential applications of drones, as they can now function effectively in remote or data-sensitive environments where continuous connectivity is not guaranteed.

Commercial drones in industries. Recently, commercial drones are revolutionizing industries such as agriculture, where they provide precise crop monitoring and spraying; construction, where they offer site surveying and progress tracking; and public safety, where they assist in search and rescue operations, disaster response, and law enforcement. The comprehensive data collected by drones not only enhances their operational efficiency and safety but also supports post-flight analysis, maintenance, and continuous improvement in drone technology and applications. As drone

technology advances, its applications are expected to expand further, offering innovative solutions and efficiencies across a broader range of sectors. The integration of in-house edge computing in drones is particularly promising, as it allows for more sophisticated onboard data processing, enabling quicker decision-making and greater autonomy. This advancement is critical for operations in challenging environments where real-time processing and minimal latency are essential. As drones continue to evolve, their role in various industries will likely grow, providing new capabilities and efficiencies that were previously unimaginable.

3.2 Existing Work and Opportunity

3.2.1 Real-time Image Processing on Drones

Drones capture high resolution images that provide larger viewpoints in real-time and also offer seamless flexibility in flight, capable of providing real-time guidance to cars. However, drones face certain limitations that requires the need for distributed processing in real-time.

Limited resources:- Drones are restricted with limited computing and energy resources [27]. Therefore, capturing and processing high-resolution imagery proves to be computationally intensive for them.

Comparable energy:- Larger drones with more computational capacity lead to their reduced flight time. Hence, ideally, it is suitable for drones to be lightweight. For such compact drones, energy consumption for flying and for computing/communication is comparable [28]. This urges us to optimize the energy consumption for the tasks, which is often solved by offloading part of the tasks to external resources.

Unstable communication channel:- Modern drones can be launched from cars and can communicate with the latter in real-time. However, the network bandwidth may be limited and unsteady based on the unstable communication channel [29]. It might not be beneficial to send the whole image for distributed processing always.

To address the above limitations of drones and to maintain adequate QoS-wise (latency and accuracy) event detection, it is beneficial to pre-process the image frame before being offloaded to a more powerful computing platform such as a car. We study few existing approaches for pre-processing (ranging from image augmentations to neural-network based feature extractors) the drone-captured images against offloading (such as frame offloading) and processing (segmentation) alternatives. We discuss each of them below:

- **Compression of image frame drone [30, 31, 32]:** The image is compressed by certain % amount reducing its size. In this case, the image can get blurred, and important features may be lost. Hence, the detection accuracy may fluctuate compared to the original frame.

- **Image Cut on drone [33, 34, 35]:** A certain portion of the complete image is cropped and processed. Here, if the cropped amount is more, certain image information, which is important for detection, may be compromised. However, if the cropped amount is less, the processing latency may remain relatively high.
- **Frame offloading to car [36, 37, 38, 39]:** When the complete image is offloaded to car and processed on it, the overall detection latency may be affected due to network instability, although drone's energy may be saved due to reduced processing on it. Also, car's processing of insensitive regions may prolong processing latency. We can note that any (pre-)processing techniques employed post frame-offloading would lead to more latency while the accuracy is dependent on the specific approach.
- **Feature Extraction of frame on drone:** A simple feature extractor (edge detector) is employed to extract important features (edges) and reduce the image size on drone. Although this reduces the processing latency and energy on drone; it may offer meager accuracy results or false detections due to the partial feature captures compared to the features in the original frame. Additionally, post-processing steps at car based on the results from drone may induce latency.
- **Partitioning & Feature Extraction on drone:** The image frame is partitioned and features are extracted (here edges) from the partitions in the drone and sent to the car for processing. Here, often, there is a greater chance of loss of most of the information as edges may lead to false detections. It is obvious that partitioning on drone, offloading partitions to car and employing feature extraction on these partitions may not be the best solution.
- **CNN-based feature extraction on drone [40, 41, 42]:** A lightweight CNN such as Efficient-Net Lite is employed for extracting features on drones, and these features are sent to car for processing. Such multi-layered feature extractors may compromise heavily on latency and energy while offering less advantage on accuracy compared to standard vehicle detection models such as YOLO, etc.
- **Segmentation on drone [43, 44, 45]:** An image is segmented for vehicles on drone, following which their results are offloaded to car.

3.2.2 Motivating Examples and Datasets

As the drone captures images, our system aims to detect events in real time around the ego-car and notify it as warning messages through established communication channels. We aim to

generalize our system’s design and runtime as a framework solution that is capable of detecting varied types of road events such as *Speeding Detection*, *Pothole Detection*, *Lane Changing*.

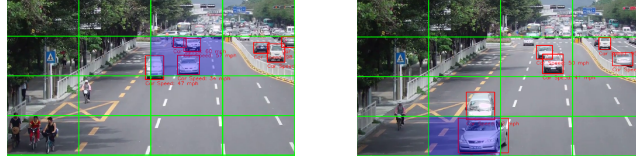


Figure 3.2: Speeding Detection



Figure 3.3: Collision Avoidance

To understand and facilitate the events’ requirements and the system’s runtime implementation, we collect approx. 500 video clips of driving vehicle trajectories from various online sources. These video clips capture different BEV road events at different resolutions.

3.2.3 Regions of Interests

We see that for detecting events from BEV captured images, the whole image is not needed. Instead, some regions are more important and essentially needed to be processed. As depicted in figures 3.2 and 3.3, each event demands processing specific neighborhood regions around the ego-car¹ that indicate the potential for an event to occur. For instance, to detect if it’s safe for the ego-car to change to the left lane, the system needs to focus on the left side (or the adjacent lane-side) of the GV for potential vehicles. Similarly, while detecting speeding events or accidents, there is a need to look for the regions around and/or ahead of the GV to identify them.

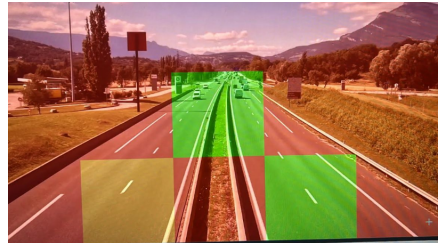


Figure 3.4: Illustration of partitioning-based approach

¹drone tracks and detects for events around the ego-car

Partition-based approach. Following the above motivations towards RoIs and limitations of existing approaches, we formulate an alternative mechanism where specific Regions of Interest (RoIs) can be selected and offloaded distributively to be processed for achieving better QoS-aware detection results. These partitions that have a high chance of an event occurring are given high importance and assigned as QoS-sensitive partitions, i.e., these partitions need to be processed immediately for possible event detection. For this, we divide the image frame into n equal parts (or RoIs). The number of partitions can be decided based on 1) The vehicle size distribution and 2) Event application specifications. Based on heuristics from our collected data and to simplify the working of the runtime, we set n as 25. For instance, as depicted in fig. 3.2, the QoS-sensitive regions (regions 3, 4, 8, 13) are to be processed and are decided dynamically based on the ego-car's location in the frame.

The other partitions that mostly identify with RSUs or have a lesser chance for events are QoS-insensitive regions that are not critical for detecting events immediately and, hence, are transmitted to the car for processing. The rest of the regions are deemed irrelevant and, hence, not considered for processing. This scheme (as displayed in fig. 3.4) allows for selective processing of input frames that reduces processing latency while not compromising on the event detection accuracy, achieving optimal QoS-aware event detection. This way, we believe the partitioning-based approach can avoid latency-intensive image augmenting procedures while providing superior accuracy compared to other approaches.

3.3 Empirical Study

We compare the performance of the partitioning-based approach w.r.t existing methods by applying these on 10 image frames that depict different event scenarios based on their QoS metrics. Since road event detection must be latency sensitive with optimal accuracy, we adopt the processing latency of the drone-captured image and vehicle detection accuracies in them as the metrics contributing to corresponding QoS-aware event detection performance. Figure 3.5c illustrates the average latency and accuracy scores plotted for each of the approaches. It can be seen that the partitioning-based approach achieves low latency and high accuracy, proving to be the optimal approach against other approaches. Now, we contrast and discuss each approach in detail compared to our partitioning-based approach. Further, figure 3.5d depicts the trade-off between the QoS and the drone energy consumed for each pre-processing approach. For this, 1) we calculate QoS scores based on the latency and accuracy achieved for each approach, and 2) measure the drone's energy consumed while processing the image frames by these methods.

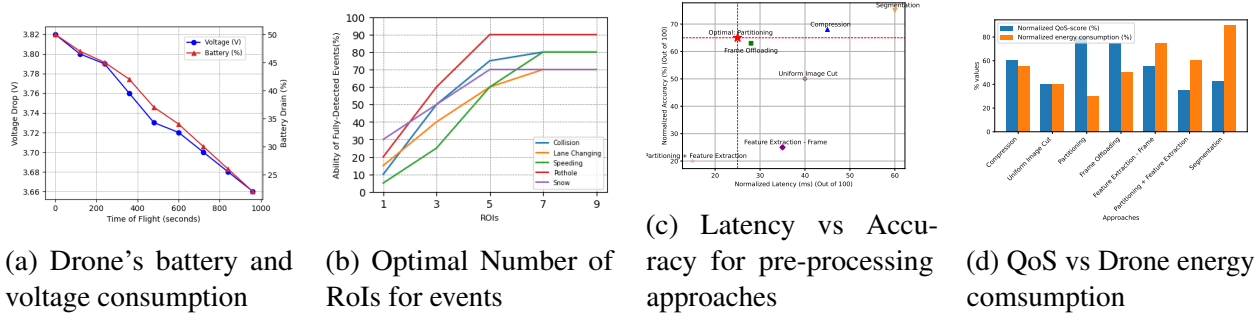


Figure 3.5: Our empirical study shows that partitioning-based approach suits best according to the trade-off between QoS-metrics and drone's limited battery

3.3.0.1 Image reduction techniques - compression, image cut

We apply image reduction techniques such as compression and image cuts on the image frames. We compress the image frame by a factor of 0.25, 0.50, 0.75, 0.95, and 0.98, respectively, and process them using the YOLOv8 model to detect vehicles. We notice the image pixels do not blur until the factor of 0.95. In the case of 0.98, the image contents are lost to a larger extent, compromising essential pixels for vehicles around the ego-car. It is important to note that compressing an image does not augment the image contents, leading to reduced energy consumption, but additionally introduces a latency overhead. Similar to image compression, we reduce the image size by uniformly cropping certain portion (50%) from the image frame (we cut the image frame proportionally from the height and width) and then process the image for vehicle detections. Image cropping has to be performed always based on the ego-car's location in the frame. Consequently, we do not consider non-uniform cropping, as the information around ego-car may be lost leading to incoherent QoS results. For these two reduction techniques the latency increases by 20% and 15% respectively, while the accuracy depletes by approximately 10% in both the cases.

3.3.0.2 Frame-level offloading and feature extraction

We look at offloading frames in contrast to transmitting only a certain portion of the complete image for processing. For this, we consider the comparison between frame-based and partition-based in two aspects: 1) *Simple Frame-offloading vs. partition-offloading* 2) *Frame-level vs. partition-level feature extraction*. For the first case, we consider the 10 image frames for frame-based offloading and select 3 – 5 partitions (RoIs) from each of these frames for partition-based offloading. We measure the processing latency for the complete frame batch and partition batch while recording the accuracies of detections using YOLOv8 inference model. We infer from the ablations that frame-based offloading achieves comparable yet 5% reduction in detection accuracy and takes 10% more latency.

Although both frame-based and partitioning-based approaches give similar QoS results, we further evaluate both approaches with other factors, such as energy consumption and performance in varied networks. Consequently, we find that the partitioning-based approach is better due to the following reasons: 1) the energy consumed by a drone for the partitioning approach is considerably less than the frame-based approach, as depicted in Fig. 3.5d. 2) considering different network conditions in realistic scenarios, selective partitioning always tends to have a lower time complexity than offloading a frame for processing directly. In such dynamic conditions, offloading less than required partitions also has a higher probability of event detection than a single frame 3) RoIs offer in-grained regarding vehicle-motion locations, their predictions, and event-probabilistic sensitive regions information that contribute to previous frames' heuristics that can be utilized in a better way for future partition predictions. It is tedious to recover such strategic data in frame-level processing. In the latter case, we are inquisitive to empirically evaluate if performing lightweight feature selectors is better than processing a frame or partition. Hence, we employ a simple feature extractor (canny edge detector [46]) to accumulate important features and then process the images. We notice that the average feature extraction latency for a single partition reduces by 85-95 % as compared to a frame. Further, feature extraction latency for 3 – 5 partitions (optimal number of partitions for an early warning) reduces by 45-60 % when compared to a frame. Whereas achieving similar accuracy or a decrease in accuracy of 5 -10 %. This phenomenon can be attributed to the drastic reduction in the number of features (approx. 90% feature reduction) post-pre-processing compared to the contents of original image frames. However, feature extractors consume approx. 15-20% time compared to processing the images. Moreover, the processing feature extracted images do not offer better results than simple frame or partition processing in terms of accuracy.

3.3.0.3 RoI partitioning vs advanced ML approaches

We see that segmentation and CNN-based feature extractors hold potential as alternative approaches to RoI partitioning and offloading. For numerous reasons, we adopt RoI offloading with vehicle identifications compared to segmentation approaches. Segmentation models are heavier and latency-consuming by 20-25% compared to the traditional detection models (refer to fig. 3.5c) and hence not suitable to be deployed onto edge resources for real-time inference. Further, segmentation models are not trained to infer BEV images from certain heights. Fine-tuning segmentation models is a tedious task, as generating segmentation labels and annotations is highly time-consuming. Currently, there are no publicly available datasets that provide BEV vehicles' masks and corresponding labels. Moreover, for detecting events in real-time, we do not need accurate size details and masking outputs for vehicles; we only need vehicle detection. In particular, segmentation is a computationally heavier task than detection and, hence, a more suitable operation at the frame level than the partition level. Also, segmentation results vary, with different image qualities potentially

being inflexible and reducing the accuracy. It can be concluded that segmentation and CNN-based feature extractors prove to be computationally expensive with fewer accuracy improvements in contrast to the partitioning approach.

Takeaway

Based on the study in this section, we see that existing approaches come with their own limitations in terms of QoS performance metrics and energy constraints that prove to be expensive for deployment in realistic scenarios. Consequently, we find RoI partitioning as the best approach to be adopted for captured image frames to deliver a QoS-aware and energy efficient system performance.

CHAPTER 4

GRIFFIN Design

4.1 Overview

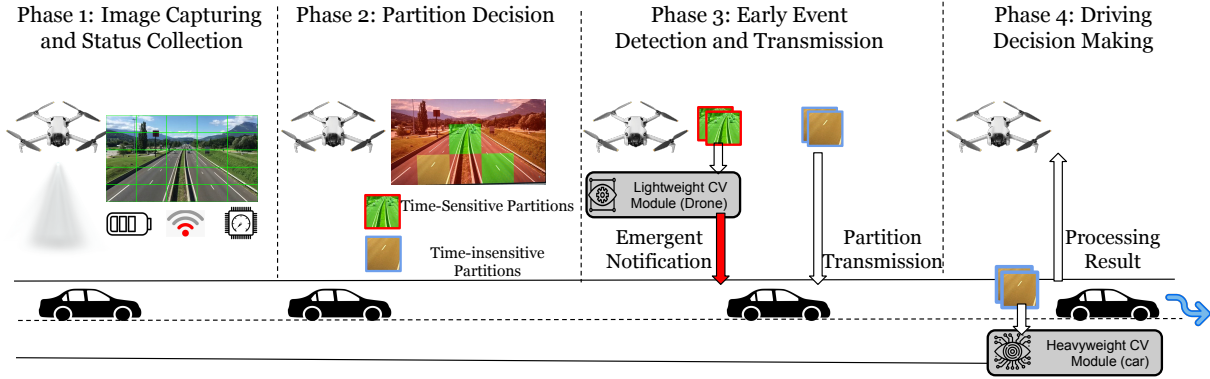


Figure 4.1: Illustration of GRIFFIN Workflow

We present GRIFFIN, a programming and runtime framework enabling real-time, drone-based road event detection by distributing image processing between a collaborative GV and UAV. GRIFFIN facilitates the development of various road event detection applications for computer vision experts by providing base CV modules fine-tuned for bird-eye image processing and shielding the distributive computing logic. An application developed using GRIFFIN is automatically distributed on the drone-car continuum, with GRIFFIN’s runtime optimizing the Quality of Service (QoS) of the application while minimizing the energy consumption of the drone. The principle guiding GRIFFIN’s design is that only a small region of interest from each captured image frame that contains useful information for real-time event detection should be processed with high QoS (i.e., shorter delay and higher accuracy), while the rest can tolerate the worst QoS (longer delay and poorer accuracy). Below we use an example to demonstrate the idea.

As illustrated by Fig. 4.1, for one application detecting car accidents that happen in front of the drone-associated car (which we call **”ego car”**), the region of interest is the image parts in front of the car on the same lane, which only counts for two blocks (we call it **”partitions”**) out of 15,

as marked with green in Phase 2. Instead of processing all partitions equally, priorities will be given to regions of interest that demand higher QoS in terms of accuracy and latency. To that end, after capturing an **Bird's Eye View** (BEV for short) image frame, the drone's runtime senses the status of the drone-car continuum, including battery, network, and historical task execution time (phase 1). Along with the processing results of the previous frames and the application-specific partition QoS demands, GRIFFIN dynamically decides which partition should be processed on the drone and which should be compressed and transmitted to the GV (phase 2). When processing the assigned partitions, the drone will send emergent notifications to the GV if an accident is detected that requires immediate attention. Simultaneously, it transmits the other partitions to the GV (phase 3). The GV then processes the received partitions and returns the processing results to the drone (phase 4). In such a way, we expect to optimize the event detection QoS without overwhelming the drone.

4.2 Challenges

Although we observe that the partitioning-based approach is promising, GRIFFIN's design faces key fundamental challenges that are listed below. Addressing these questions, we further elaborate on each of the key ideas that formulate GRIFFIN's design.

- *How to process the partitions in a general workflow for different event applications?* Detecting different road events requires vastly different processing logic in terms of which partitions should be prioritized and how processing partitions on cars can assist drone-based processing. Hence, we aim to build a system workflow that can be used for different applications.
- *How to efficiently distribute the workload, ensuring optimal utility and limiting drone's energy consumption?* Many factors, such as network conditions, processing power on drones and cars, and current energy status, may impact the utility of distributed processing, affecting its accuracy, energy consumption, and latency.
- *How to enable CV developers without distributed system knowledge to build such applications?* With GRIFFIN's design goal being to facilitate road event detection applications, the main users of GRIFFIN are CV developers who have limited knowledge and experience with distributed systems. We aim to abstract the low-level distributed processing and provide proper interfaces, allowing developers to focus on the application-specific logic itself.

4.3 GRIFFIN Overview

To facilitate the development of real-time road-event detection applications utilizing the integration of drones and cars, GRIFFIN features the following components:

- **Runtime System:** This system processes each frame and notifies the car if a target event is detected. It enables different applications to distribute their processing with a unified workflow and optimizes the partition of processing tasks between the drone and the car to maximize processing utility and minimize energy consumption.
- **Programming Support:** GRIFFIN provides intuitive programming interfaces allowing computer vision (CV) developers to create road-event detection applications without extensive knowledge of distributed computing. It also includes a compile-time tool that automatically converts a centralized application into distributed executables for deployment on drones and cars.

4.3.1 Generic Workflow for Distributed Partition Processing

Our key idea for building the generic workflow is to enable developers to specify how to calculate an importance index for each partition, by which the runtime system will decide how to distribute the processing of partitions. Fig. 4.2 shows how we designed GRIFFIN to process each frame taken by the drone’s camera following this idea. The steps are introduced in details below.

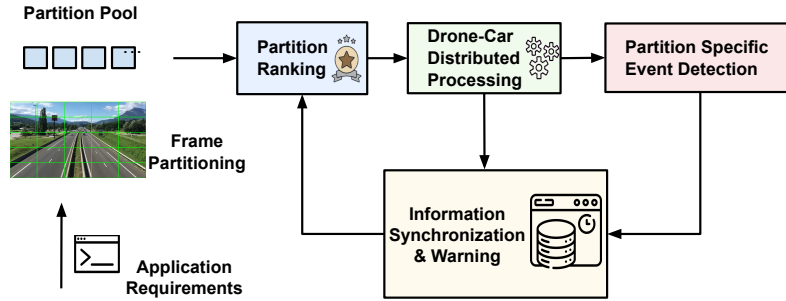


Figure 4.2: Generic Workflow of GRIFFIN

1. **Status-sensing and Frame Pre-processing.** Before processing each frame, GRIFFIN first senses the status of the network and the processing delays on the drone and on the car, to decide whether or not to process the frame. Utilizing the real-time status information and the previous frame’s processing results, GRIFFIN will either **discard** the whole frame (if the resources on the drone are low, the network condition is poor, and another frame has been processed recently), **offload** the whole frame to the car (when the network bandwidth is sufficient), or start the **partition**-based processing.
2. **Application-specific Partitioning and Ranking.** GRIFFIN reads the application-specific configuration files to divide the frame into partitions. It estimates the ego-car’s position using information obtained from previous frames and calculates the importance score for each

partition based on the ego-car’s location, an application-specific template provided by the CV developer, and information captured from previous frames. For example, in a pothole detection application, the template can assign a high score to the partition in front of the ego car. In a collision detection application, a partition may receive a high score if another vehicle, which was approaching the ego-car at high speed in the previous frame, is estimated to be in that partition in the current frame. All partitions will be ranked based on their importance values.

3. **Partition Distribution.** GRIFFIN utilizes the sensed status to determine how to distribute the partition processing. Specifically, it calculates how many partitions should be processed on the drone, how many on the car, and how many should be discarded. Based on their importance ranking, the most critical partitions will be processed on the drone to provide timely (or early) warnings if an event is detected. The next batch of partitions will be processed on the car, while the remaining partitions will be discarded. Our algorithm for determining the processing capacities is detailed in Sec. 4.3.2.
4. **Partition-based Image Processing** GRIFFIN’s runtime on drones and cars will process each assigned partition. Considering the differences in computing power and energy between the two, the CV modules executed on the car for event detection can be more powerful and accurate but are also more resource-consuming. In contrast, drone-based event detection, while potentially less accurate, avoids transmission latency and enables faster event detection and early warnings to the car. For each partition, the image processing detects target road events and related objects, such as the location of the ego-car and the speed of nearby cars for lane changing. The image processing module can access information from previous frames to achieve this.
5. **Information Synchronization and Warning.** Object and event detection results from previous frames will be stored in an in-memory cache on both the drone and the car. Dedicated APIs for storing and retrieving objects from the cache will be provided for CV developers within their application-specific image processing modules. In order to avoid the complexity of frequent cache synchronization, the image processing module can only utilize information from previous frames, not from other partitions of the current frame. The caches on the car and the drone will be synchronized once a frame is fully processed.

4.3.2 Utility-Aware Partitions Distribution

Here, we introduce how GRIFFIN estimates the optimal partition distribution between the drone and car from a partition pool given the drone’s battery consumption for transmission and processing

and uses the previous frame processing results to calculate the utility for each frame.

4.3.2.1 Overview

The drone's frame processor monitors the real-time flight data and decides on the existing instances of partition processing to fulfill the requirements. The drone's processing unit does this by calling the distribution algorithm to launch new instances of partitions distribution to drone, car for processing. Given the previous results and the online-flight data, the *partitions distribution* algorithm explores the optimal distribution pattern for partition instances in the frame to minimize the drone's energy while guaranteeing the event detection utility.

4.3.2.2 Modeling

We model our algorithm as a multi-objective closed optimization solver. Denoted by m is the partitions (say $\mathcal{P} = p_1, p_2, p_3 \dots p_m$) which are processed (for the current frame); the algorithm decides for the partitions in the pool $\mathcal{P}$ where to process them, on drone (n_1) or car (n_2) or to be discarded ($n_3 = m - n_1 - n_2$). For our work, we consider processing the partitions in a sequential manner. We aim to explore other processing methods, such as parallel processing, to reduce the latency over the partition load in the future.

We assume that the following information is provided as inputs or can be obtained by utilizing a dynamic oracle:

- Drone Status Sensing, including drone's current battery status, battery drain rate (r), flight direction, and speed.
- Image processing results for previous frames (window of β frames)
- Network and Energy Measurements: 1) the time for transmitting (T_t) a partition; 2) the time for processing a partition on drone T_d and on car T_c ; 3) the energy consumption for transmitting/processing a partition, denoted by E_t and E_d respectively. In this work, we consider taking their values by aggregating previous frames' measurements.

The time for sending a partition to the car is T_t , the time for processing on the drone is T_d , and for processing on the car is T_c . Accuracy for a partition that is processed on drone (A_d) and on car (A_c) is calculated as a normalized score based on the following objectives: 1) Confidence Score 2) Partition Weight 3) Importance of the partition towards event as depicted in the statement 4.1.

$$A_{normalized} = ConfidenceScore \times PartitionWeight \times PartitionImportance \quad (4.1)$$

We estimate the objectives' values for each partition.

- **Confidence Score:** This score represents the class-inference score (here, vehicle class) obtained when a partition is inferred for objects.
- **Partition Weight:** Each partition is assigned a weight that depends on its proximity to the ego car's location.
- **Importance of partition towards the event (I):** Importance value (I) for a specific partition is calculated based on the impact it provides towards the event. It contributes to the accuracy calculation in a partition as well as utility estimation during partition distribution.

Since GRIFFIN's runtime partitions a frame uniformly, we consider the time T_d and energy E_d for processing each partition on the drone as same.

The measurements $T_d, T_c, T_t, E_d, E_t, A_d$ and A_c are calculated from $\mathcal{R}$ that is extracted from the storage cache. We calculate these values as the standard deviation of the average of the same values (here, T_d) from the previous $n - k$ frames. We chose β as 3 for our cases. The higher the magnitude of k , the lesser the error expectancy of the measurements in the current frame-time. For instance, T_d is calculated as depicted in equation 4.2.

$$\begin{aligned} \mathcal{T}_d^n &= \sigma[\overline{T_d^{n-1}}, \dots, \overline{T_d^{n-k}}] \\ s.t. : \overline{T_d^k} &= \frac{1}{m} \sum_{j=n-1}^m T_d^j \end{aligned} \quad (4.2)$$

In particular, T_d is the actual time for processing on a drone that is obtained post-processing, and $\mathcal{T}_d$ is the time considered to estimate the utility for the current partition pool. Similarly, other measurements are estimated from the previous results $\mathcal{R}$.

f_1, f_2, α : We denote f_1 and f_2 as QoS-sensitive and regular utility functions respectively. We formulate the utilities based on the type of event. To compute the utility of drones and the utility of ground vehicles, we consider the effects of latency and accuracy measurements from $\mathcal{R}$ to influence the above functions. Taking the case of *accident detection* or *collision avoidance*, it is extremely important to reduce the latency and give the ground vehicle an early warning, as the ground vehicle can communicate with nearby vehicles and RSUs to confirm this information. Hence, as depicted in equations 4.3 and 4.5, we assign higher weights to the latency in utility function f_1 , and lower weights to its accuracy. In such a case, we specify f_1 & f_2 as

$$f_1(\mathcal{T}_d, \mathcal{A}_d) = \epsilon^{1/(\mathcal{T}_d \log \mathcal{A}_d)} \quad (4.3)$$

$$f_2(\mathcal{T}_t, \mathcal{T}_c, \mathcal{A}_c) = \epsilon^{(\mathcal{T}_c + \mathcal{T}_t) \times (1 - \frac{1}{\log(\mathcal{A}_c)})} \quad (4.4)$$

For event applications that are dependent on QoS-regular utility such as *Speeding Detection* or *Lane Changing*, the decisions may be more dependent on the accuracy than the detection latency. In such cases, we distribute the weight for latency and accuracy equally or emphasize the accuracy part more if the ground vehicle has no other ways of confirming the event. Then, we can depict f_1 & f_2 as

$$f_1(\mathcal{T}_d, \mathcal{A}_d) = \epsilon^{-\mathcal{A}_d \times \log(\mathcal{T}_d)} \quad (4.5)$$

$$f_2(\mathcal{T}_t, \mathcal{T}_c, \mathcal{A}_c) = \epsilon^{\mathcal{A}_c \times \left[1 - \frac{1}{\log(\mathcal{T}_t + \mathcal{T}_c)}\right]} \quad (4.6)$$

4.3.2.3 Problem Formulation and Solution

We adopt a Greedy approach to select the partition distribution that is utility and energy-wise optimal. The optimization solver comprises of a 1) **Process Loader**, 2) **Goal Calculator**, 3) **Utility Monitor** and 4) **Output Selector**.

As depicted in algorithm 1, The algorithm engine triggers the *Process Loader* that forwards the partitions combinations to the calculator engine. Based on the partition pool size m , it is decided whether to discard partitions from the pool or not. That is, if m exceeds a certain number, say q , another bucket will be allocated to decide on the number of partitions to be discarded. Essentially, this means that the three buckets (say $n1, n2, n3$) would be created as the combination if the condition is met. Otherwise, the combination comprises of two buckets for drone and car ($n1, n2$). Following this, the *Goal Calculator* estimates the utility for each partitions distribution combination. Following that, *Utility Monitor* searches for the best goal (eq. 4.9) obtained such that utility meets a threshold.

$$u_i = \alpha \times I_{avg}^{n1} \times f_1(\mathcal{T}_d, \mathcal{A}_d) + (1 - \alpha) \times I_{avg}^{n2} \times f_2(\mathcal{T}_t, \mathcal{T}_c, \mathcal{A}_c) \quad (4.7)$$

$$E_i = E_d^i \times n_{drone}^i + E_t^i \times n_{car}^i \quad \forall i \in [1, \dots, m] \quad (4.8)$$

Hence, the optimization problem can be formulated as follows:

$$\text{maximize: } \begin{cases} \frac{u_i}{e^r} - E_i & ; r \in [0, b_r) \\ u_i - E_i \times e^r & ; r \in [b_r, 1] \end{cases} \quad (4.9)$$

Utility-monitor monitors the utility for each combination and gives the best distribution combination that satisfies the constraints (utility threshold λ). Where b_r denotes the battery drain rate threshold value between $[0,1]$ that depicts the critical battery status during the drone's flight. A

higher drain rate is composed of a value near to 1 and vice-versa in the case of regular drain. We denote r in eq. 4.9 as the battery drain rate (battery power-drain/sec) for the overall battery consumption rate due to drone flight, computations, processing, communications, etc. α denotes the utility factor for a specific event that is configured by the developer. We break down α as a composition of latency (l) and accuracy (a) factors (upper-bound of the average of latency and accuracy factors) that we decide based on the type of event.

Finally, *Output Selector* selects the optimal combination and forwards the partitions subpools to the drone and car for processing.

Algorithm 1 Partitions Distribution

```

1: function DISTRIBUTE( $\mathcal{P}$ ,  $\mathcal{R}$ )
2:   Input: Array of partition pool  $\mathcal{P}$ , Dictionary of previous results  $\mathcal{R}$ 
3:   Output: Array of partition Ids for drone, Array of partition Ids for car
4:   Step 1: Sort pool based on Importance weights in descending order and decide on discarding factor
5:   Initialize sorted  $\mathcal{P}$  as an empty dictionary
6:   Initialize  $q$  as the discarding factor
7:   Initialize  $x$  as the number of combination buckets
8:   sorted  $\mathcal{P} \leftarrow \text{sort}(\mathcal{P})$ 
9:    $x \leftarrow 2$  or  $3$  based on  $q$ 
10:  Step 2: Assign Combinations to partitions
11:  Initialize  $\mathcal{C} \leftarrow \text{AssignCombinations}(\text{sorted } \mathcal{P}, x)$ 
12:  Step 3: Goal Calculation for each combination based on  $r$ 
13:  for each combination  $c$ ,  $c\_id$  in  $\mathcal{C}$  do
14:    goal = calculate_goal( $c$ ,  $f_1$ ,  $f_2$ ,  $E$ ,  $\alpha$ ,  $r$ )
15:    GoalStorage.append(goal,  $c\_id$ )
16:  end for
17:  Step 4: Search for the combination that satisfies utility constraint  $\lambda$ 
18:  for each goal,  $c\_id$  in GoalStorage do
19:    if  $u_{c\_id} > \lambda$  then
20:      Store  $c\_id$  if goal is maximum
21:    end if
22:  end for
23:  return  $\mathcal{C}_{c\_id}$ 
24: end function

```

4.3.3 Programming support

GRIFFIN's runtime design abstracts many of the distributed computing processes, allowing CV developers to focus on the application's logic itself. We first introduce how CV developers can implement application-specific logic, followed by the API support provided for interacting with GRIFFIN's runtime. Finally, we build a toolchain that helps developers write their applications in a centralized manner and automatically divides them into distributed executables for drones and cars. All these programming supports are exposed to application developers as GRIFFIN SDK (Software Development Kit).

4.3.3.1 Implementing Application-specific Logic

As revealed by the runtime design, CV developers can easily implement an application by providing the logic for the following four modules: a) Frame Pre-processing: decide whether or not to involve partition-based processing for an incoming frame; b) Partition Ranking and Distribution: calculate the importance value for a partition by pre-defining a template or using the objects or events detected in previous frames; c) Drone-based Partition Processing: detect objects and events in a partition using lightweight CV algorithms, store the results in the cache, and send notifications to the car if a target event is detected; d) Car-based Partition Processing: enhance the drone-based processing by replacing the lightweight algorithms with more powerful ones for greater accuracy.

Additionally, CV developers can specify their application-specific parameters in a configuration file. These parameters, which may not need to be changed frequently, include those for calculating utilities in the optimization process and the template for importance calculation.

4.3.3.2 API support

To further support CV developers, GRIFFIN offers the following APIs:

- CV APIs, which abstract the commonly used CV algorithms for applications to process BEV images.
- Notification, which enables partition processing modules to send notifications to the car.
- Cache, by which CV developers can store and retrieve events and objects either on the drone or on the car without managing their synchronization. When accessing an object or event from a previous frame, the time interval between the previous and the current frames as well as the distance between the object and the ego-car (in pixels) will also be provided.
- Optimizer, which takes a partition pool and returns the optimal partitions distribution scheme. Next, The partition sub pools are to be distributed to the drone's processing unit and the car's processing units. Since the Optimizer includes fine-grained decision and processing logics, the API encapsulates them by providing an input and output channel to get and put the input and result. This API interacts with Cache to store the processing measurements such as optimal utility achieved, and distribution output.

4.3.3.3 GRIFFIN SDK

The SDK includes all the files for CV developers to develop a distributed road event detection application. As depicted in fig. 4.3, developers can use the *toolchain* to create a template project and fill in the application's processing logic, using the APIs listed above. The *toolchain* also helps

compile the application into two executable files, for the drone and the car. These executables can be installed and run on the drone and car, and include all the runtime libraries introduced above.

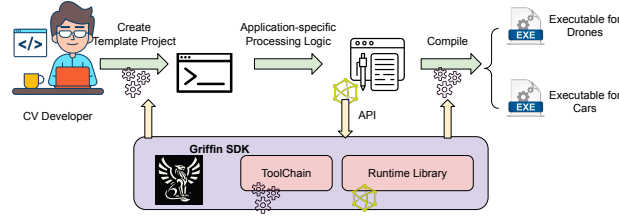


Figure 4.3: GRIFFIN SDK

4.3.3.4 Toolchain Components

To easily configure, build and deploy the CV modules as drone and car executables, a toolchain is designed in GRIFFIN. The toolchain enables users to easily mutate and re-configure the underlying modules. This toolchain provides two types of interactions (intrinsic and extrinsic) among the modules, API libraries and configuration files. The modules in each resource (depicted in 4.4) are tightly coupled to provide flexible interactions among them in the intrinsic layer. These modules exchange processing information and data during runtime extrinsically using API libraries such as notifications and communication interfaces that are present in each executable. Cache transfers and stores data using the two interactions types for intra-(within each resource) and inter-(between the resources) operations. The toolchain distributes the instances of drone based, car-based and app-specific processing logics that have been declared in programming interfaces.

Using the toolchain, the users can:

- incorporate and offload libraries and APIs by selecting in the toolchain interface or declaring the parameters and dependencies with other modules in JSON files.
- develop new instances of executables from the pool of libraries (or modules declared in Programming Interface) based on the users' necessities.
- mutate standalone processes (for e.g. *preprocessing*, *YOLOv5 image inference*) to third-party users in the open-source community.

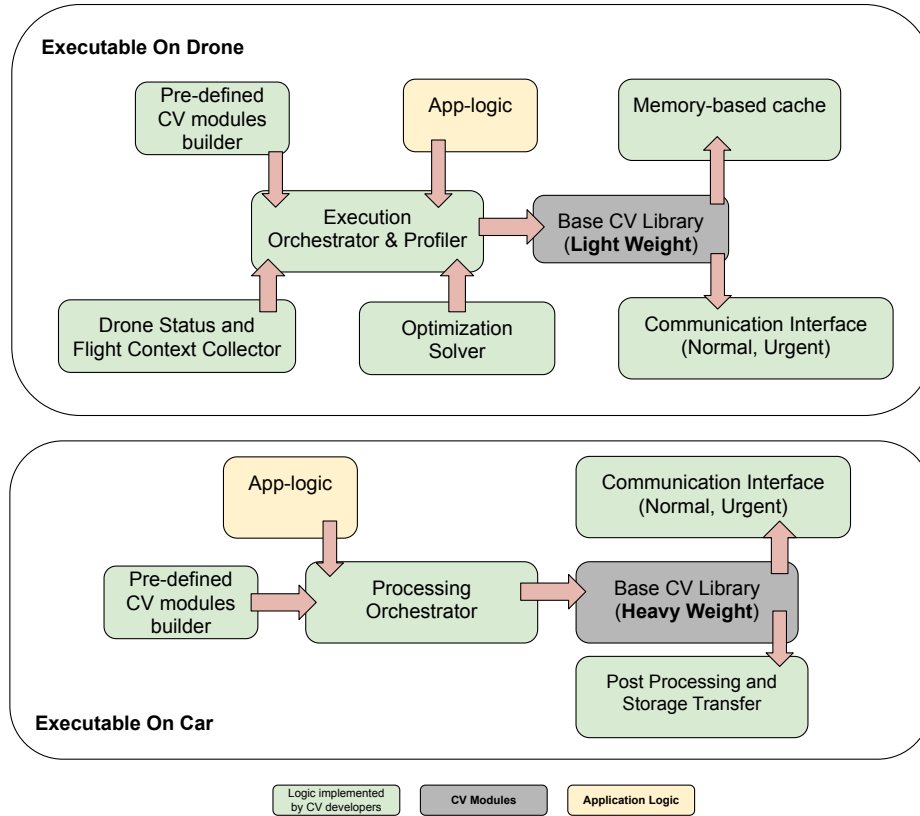


Figure 4.4: Processing modules in drone and car executables

Summary

- GRIFFIN is designed as a distributed middleware system that facilitates the development of drone-car collaborative road event detection applications. GRIFFIN detects road events based on image partitions to achieve high detection accuracy with minimum energy consumption.
- GRIFFIN provides a programming interface for CV developers to implement the application logic without considering the distributed processing, a toolchain to distribute the CV application into executables for the drone and the car, and runtime support for executing the CV applications.
- We deploy advanced Computer Vision algorithms on drone-captured views to perform energy-efficient partition distribution on frames by formulating a constrained optimization problem in realistic scenarios.
- We fine-tune built-in CV modules using our collected BEV data to enable accurate vehicle detections. We evaluate the fine-tuned modules on our collected data and see that the performance is enhanced by approx. 23% in terms of detection accuracy.

CHAPTER 5

Reference Implementation

This chapter introduces the overall reference implementation for GRIFFIN. Based on the runtime workflow of GRIFFIN, all individual components are implemented.

5.1 Runtime Workflow

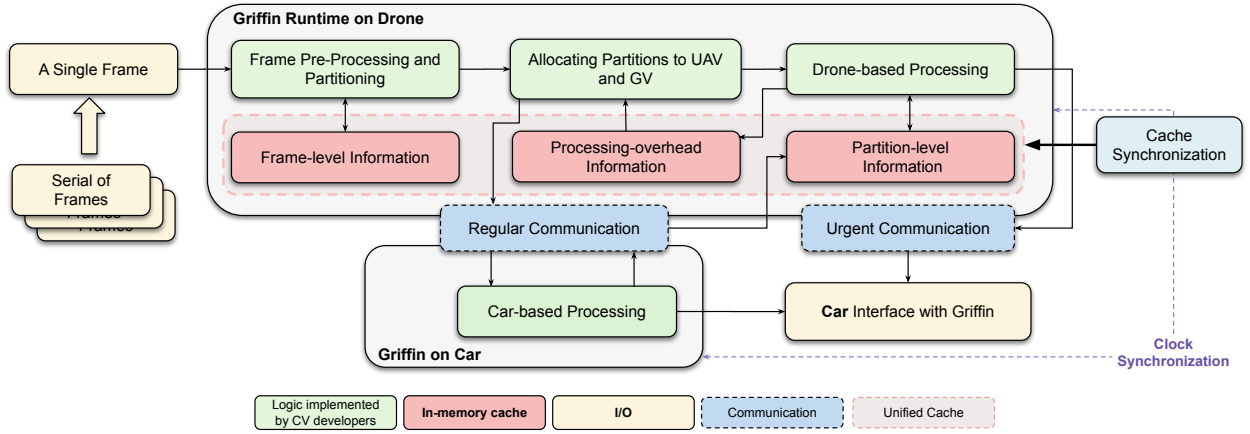


Figure 5.1: System Components of GRIFFIN Prototype

Based on the design solutions we have discussed in the previous parts, we adopt and formulate a simple yet robust runtime for GRIFFIN that 1) deploys the application-generic workflow onto a middleware framework 2) solves the dynamic partition workloads 3) supports CV developers to build and deploy applications.

To better understand GRIFFIN’s working, we describe its runtime in detail here. **①** For each frame captured by the camera, the drone undergoes a partition planning procedure to decide how and where to process the image. As illustrated in the runtime workflow (fig. 5.1), the frame will be divided into a fixed amount of partitions (i.e., 5*5 for each frame), and the procedure dynamically decides whether to process the partition on the drone or car. **②** Historical results are stored in the car (storage cache) to give context to the current frame’s processing result. Storing the captured

image frames and partitions in the cache would induce disproportionate information overload on the drone. To alleviate this, all the information that is necessary for (pre and post) processing and running GRIFFIN is stored in the cache temporarily and offloaded to a robust storage in the car at regular intervals. The processing destination for each partition will be based on their latency and processing energy requirements. ③ To minimize the drone’s energy consumed for event detection, effective distribution of partitions takes place to simultaneously process on drone and car. ④ Based on the event application, an early warning and/or regular warning is communicated to the car. The distributed partition processing aims to satisfy the following: 1) reduce the processing and transmission latency between car and drone, 2) minimize the processing overhead over the two, leading to an optimal utility for overall event detection.

Summary: The drone will collect real-time information related to data transmission and processing, the drone’s status, as well as the processing results of previous frames to make decisions for the partitions in the current frame. For partitions processed on the drone, if any event is detected that requires immediate attention of the autonomous driving system, the drone will notify the car by sending out-of-band data via established TCP connections. Due to the limited resources on the drone, the image processing algorithm on it will be less accurate but faster than the one on the car. Hence, if a frame is partitioned with some parts being processed on the drone, we expect the early notification sent by the drone to be less accurate. The overall utility of GRIFFIN can be seen as the combination of benefits derived by drone-based early detection and car-based regular detection.

The GRIFFIN prototype is implemented and experimented in Python (v3.8.0) and C++ and it contains of approximately 750 LOC (excluding third party libraries). We utilize our fine-tuned YOLOv5 and YOLOv8 as the lightweight and heavyweight modules, respectively, with PyTorch [47] being our inference engine.

5.2 Fine-tuning CV modules

We deploy YOLOv5 [48] and YOLOv8 [49] for real-time vehicle detection, with YOLOv5 being lightweight and faster but less accurate than the more accurate and heavier YOLOv8 [50]. When these models were tested on our BEV images (see chapter 5), the results revealed misclassifications and false detections, particularly with vehicles from different orientations and resolutions. This is likely because the built-in modules are trained on general object classes and ground-level vehicle images. To improve detection from drone-captured BEV images, we fine-tuned both modules using the Visdrone dataset [51] and our collected videos. We gathered 500 driving vehicle videos from online sources and 10 drone videos at varied heights using a DJI drone [3]. We annotated 300 image frames with Roboflow [52] to fine-tune the modules. The fine-tuned models significantly improved detection accuracy on our data (see chapter 6).

Application	Collision Detection	Speeding Detection	Lane Changing	Congestion Avoidance	Pothole Detection	Accident Identification	Snow Detection
Generic CV modules			Vehicle Detect(partition, model); Model Inference(frame, object-classId); Partitioning(frame, nRows, nCols); frameProcess(frame); droneProcessing(<partitions>); carProcessing(<partitions>);				
Application based CV modules	Obstacle Distance Calculation (ego-car relative-location, obstacle relative-location, distance threshold)	Speed Detection (partition, number of vehicles threshold, speed threshold)	Lane Change Decision (partition, ego-car - {location, frame-time, lane-detail}, obstacles -{location, frame-time, lane-details})	Congestion Detection (partition, distance threshold, vehicle density threshold)	Pothole size detection (frame, threshold)	Accident Detection (partition, model detection threshold, distance threshold)	Snow Density Estimation (frame, area-size threshold)

Table 5.1: Generic and Application based CV algorithms

5.3 Programming Aspects

GRIFFIN’s design allows CV developers to understand and execute the applications without having to understand the in-grained details. To do so, as we have discussed in chapter 4, GRIFFIN essentially implements the following aspects at the programming level: 1) Generic and application-specific CV modules, 2) Abstractions, 3) Configurations. Further, to show the viability and feasibility of GRIFFIN, we describe the processes that we implement in the prototype.

Generic and application-based CV modules:

We implement the CV processes(or called as algorithms) in two ways. Firstly, the CV processes that are common to all applications. They form to be the base, high-level implementations for GRIFFIN. These processes are generic to multiple use cases, and most of them can be implemented in open-source platforms. Their implementations possess fewer formal parameters to be defined. Some of the processes that we implement in this category include *Vehicle Detect*, *Model Inference*, *Partitioning*, *Frame processing*, among others. Secondly, the specific processes that each application demands. These are specific to applications and have a high chance that their logic and formal parameters will be modified during implementations each time. They may vary for each use case during deployments. An example of the processes is depicted in table 5.1. We design the algorithms in a manner that 1) certain application-based processes can be utilized by other applications as well. For e.g., proximity estimation that is declared primarily in collision avoidance is instantiated and used for Speeding detection and 2) These processes also interact with operations in dedicated APIs during runtime. The CV modules are declared in the programming interfaces. GRIFFIN is open for CV developers and experts to add application specific algorithms to the programming interfaces as shown in 5.2.

Abstractions:

Since GRIFFIN supports parametric and process-wise contextual heterogeneity during its development, it implements key abstractions for its runtime. We classify the abstractions into the following four topics:

- **Global and local contexts:** As we deal with frames and partitions concurrently, GRIFFIN maintains frame-level and partition-level processing operations. Through this, we believe the information synchronization and exchange between the processing outputs remains decoupled

with the necessary cohesive details.

- **Object Referencing and Hierarchical Inheritance:** We implement frame and partition as class instances with partition inheriting the properties of the frame. A partition is referenced and can be accessed as `frame<n>.partition<x><y>`. The partition pool are formatted as `frame<n>.pool`. The coordinates for a vehicle or object can be referenced as `frame<n>.object<id>.coordinates<xywh>`. Another concept, namely regions that are necessary for motion estimation of vehicles through images, is introduced. A frame can contain multiple overlapping regions, having common partitions. These common partitions possess high importance factors. Although we do not implement Regions in this work, they immensely contributed to partition-level information and importance calculation in preprocessing steps.
- **Relative notations:** We consider frames as global data points and partitions as local data points. To ensure the coherence between notations in frames and partitions, we implement relative notations for vehicle locations in a partition to be referenced as locations at the frame level. For this, we convert the local coordinates into the global context as depicted in eq. 5.1. The frame's left bottom is considered as the origin (0, 0) for the coordinate reference system in the frame.

$$\begin{aligned} X &= (m - 1) * (partitionWidth) + X' \\ Y &= (n - 1) * (partitionHeight) + Y' \end{aligned} \tag{5.1}$$

where m and n are the partition-column and partition-row in which the target vehicle is present. X & Y represent coordinates in the frame-based system and X' and Y' partition-based coordinates. We also implement relative speed estimation. This is due to the inconsistency of speeds of 1) drone viewpoint-ego car, 2) drone-vehicle 3) vehicle-vehicle 4) vehicle-RSU w.r.t ground reference. In order to avoid inaccurate measurements, all speeds of objects in an image are estimated from a single reference point. While implementing GRIFFIN, we assume the drone follows the ego-car at the latter's speed. Nevertheless, to calculate the speeds of the RSU, car, and drone with respect to the ground, we need to consider the relative speeds of these objects with respect to each other and the drone's speed with respect to the ground. Denoting the following with notations:

- v_d : Speed of the drone with respect to the ground.
- $v_{c|d}$: Speed of the car with respect to the drone.
- $v_{r|d}$: Speed of the RSU with respect to the drone.

- v_c : Speed of the car with respect to the ground.
- v_r : Speed of the RSU with respect to the ground.

The drone's speed is obtained from flight sensing data. The speed of the car and RSU with respect to the ground can be formulated based on the following equations:

$$\begin{aligned} v_c &= v_d + v_{c|d} \\ v_r &= v_d + v_{r|d} \end{aligned} \tag{5.2}$$

- **Dimensional matching:** To avoid confusion regarding frames and their processing w.r.t time instances, we introduce a concept of `frame-time`, where all the sequential indexing for data would be based on frames, irrespective of the captured frame-rate and discarded frames decided by GRIFFIN. GRIFFIN implements `pixel-distance` that is used in processing logics to calculate proximity, etc. `pixel-distance` is used in terms of *pixel-per-meter(ppm)* value that is a static value based on the frame and camera settings. Results are more accurate if *ppm* is decided dynamically or calibrated during runtime.
- **API operations:** APIs subdue the underlying processes by providing a layer that supports standard operations implemented through dedicated channels. For instance, `store`, `retrieve`, `update`, `lock`, and `delete` are used for specific storage API-based procedures. The regular offloading procedures are performed using a `batchOffload<start frame-time><end frame-time><specific measurements>`. The `<specific measurements>` parameter is optional. If not specified, all the measurements for the frames batch is offloaded to the permanent storage.
- **Process encapsulation into prospective microservice APIs:** Road event detections require implementations of multiple standalone CV algorithms such as *ego-car detect*, *RSU detect*, *lane detect*, *drone speed detection using images*. These algorithms have fine-grained CV performance and version dependencies that are to be considered. Hence, these algorithms, when converted to interactive microservices with resource hardware, alleviate multiple installation and development issues for users. Few such methods, such as ActiveTrack [53], are provided in-built in the latest drone systems for generic purposes. Similarly, dedicated services focusing on event detections are to be developed.
- **Permission and access control:** GRIFFIN manages permissions primarily for three user types: 1) GRIFFIN developers, 2) CV developers, and 3) third-party application users. To streamline access requests and control, GRIFFIN maintains a permission management approach for others to access its resources. Each user is given a capability list while releasing the GRIFFIN SDK.

GRIFFIN developer has access to *modify*, *read* and *write* to all aspects. CV developers have access control over the CV algorithms. They can *connect* and *request* for access to GRIFFIN SDK. Following this, they can only *modify*, *read*, and *write* to the CV algorithms that are declared.

Third-party application developers develop and create instances of event applications by *calling* GRIFFIN’s programming interface. Further, They are given access to *write* configurations and app-specific logics to the system at compile-time. They do not have rights to *modify* the process workflow, while they can *change parameters* in the processes. Since GRIFFIN is in the prototype stage, we present permission and access control in its ideation stage. Consequently, we open-source certain aspects of GRIFFIN’s resources. We wish to implement this feature at a later stage.

```

1  @StorageAPI
2  public class CacheStorage{
3      frame f, fid
4      @Entity
5      void store(){
6          self.post(fid, fid.objects)
7          self.post(fid.pid,f.partitionid.objects[id], bool(f.partition(detection)))
8          self.post(fid.pid, f.partitionId.objects[id], bool(f.partition(event)))
9      }
10
11     @Entity
12     obj retrieve(){
13         self.get(fid.prev(framet).objects[id].
14             relLocation)
15     }
16 }

```

Figure 5.2: Example code highlighting abstractions in cache API

Challenges. Although the above method can implement the runtime, offering ease of use, it has at least a few limitations. First, assuming the drone follows the car at a similar speed, viewpoint orientations are error-prone. Unlike a static camera viewpoint where values such as *ppm* are fixed, this case can have erroneous *ppm* instances leading to inaccurate pixel-distance measurements. This needs continuous calibrations. Second, the concept of frame time needs careful consideration when frames are skipped; Clear rules are required to be formulated for such dimensions. Third, GRIFFIN’s clock and cache synchronization periods are time-wise implementable. This conflicts with the concept of frame-time adopted for GRIFFIN. This asks for another matching procedure to avoid mismatches. Last, introducing regions as parent objects to partitions increases the probability of objects’ presence but induces processing delays.

Configurations:

Processing logics include parameters and compile-time dependencies, which tend to vary from event to event. These are provided as configurations that can be filled-in by developers or ideally

uploaded as configuration files. Configurations play an important role in GRIFFIN's design efficiency by bridging the gap between GRIFFIN's runtime CV modules and development constraints handled by the developers. This makes it easier for developers to update parametric values and replace new ones depending on event and runtime contexts. The summary below depicts the options that are provided as configurations to CV developers:

- **Frame pre-processing and partitioning:** Number of partitions a frame is split into, m .
- **Application specific logics:** Drone and car based Utility functions, f_1, f_2 .
- **Compile-time dependencies:** Dependency libraries that are required by application modules during installation.
- **Partitions Distribution solver:** ; Application based - utility-factor, α , latency-threshold t , accuracy threshold a .

```

17 App SpeedingVehicleDetection{
18     @override
19     FramePreProcessing(frame f){
20         if(storage.frame.retrieve(f.previous).processed==true){
21             skipFrame
22         }else{
23             f.processed = true
24             f.egoVehicleLocation = BaseCVModule.recognize(configure.selfVehicle)
25             storage.frame.save(f)
26         }
27     }
28
29     @override
30     PartitionAllocation(frame f){
31         partitions = f.partitionByTemplate(configure.PartitionTemplate)
32         foreach(partition in partitions){
33             partitionWeight[] = calculatePartitionWeight(partition,
34                 storage.frame.retrieve(f).egoVehicleLocation, configure.partitionWeight)
35         }
36         Optimizer = new Optimizer(drone.status)
37         partitions.allocate(Optimizer,partitionWeights)
38         foreach(partition in partitions){
39             if(partition.car==true){
40                 processPartitionOnCar(partition, partitionId)
41             }elseif(partition.drone==true)
42                 processPartitionOnDrone(partition, partitionId)
43             }else{
44                 drop(partition, partitionId)
45             }
46         }
47
48     @Override
49     DronePartitionProcessing(Partition p, PartitionId p_id)
50     {
51         objs = p.LightBaseCVModule.recognize("vehicles")
52         speedEgoVehicle = calculateSpeed(egoObj, egoObj.prev(currentTime - preObj.time))
53         speedOtherVehicles = []
54         foreach(obj in objs){
55             speedOtherVehicles[OtherVehicles[i]].append
56             (calculateSpeed(obj.location, obj.prev(current - preObj.time).location))
57         }
58         if(any(speedOtherVehicles)>speedEgoVehicle &&
59             distance(p.frame.egoVehicleLocation,obj.location)
60             <configure.locationThreshold){
61             urgentCommunication.send(warningCode = 100, message = "speeding vehicle close to
62                 ego vehicle")
63         }
64
65     @override
66     CarPartitionProcessing(Partition p, PartitionId p_id ){
67         objs = p.LightBaseCVModule.recognize("vehicles")
68         speedEgoVehicle = calculateSpeed(egoObj, egoObj.prev(currentTime - preObj.time))
69         speedOtherVehicles = []
70         foreach(obj in objs){
71             speedOtherVehicles[OtherVehicles[i]].
72             append(calculateSpeed(obj.location, obj.prev(current - preObj.time).location))
73         }
74         if(any(speedOtherVehicles)>speedEgoVehicle &&
75             distance(p.frame.egoVehicleLocation,obj.location)
76             <configure.locationThreshold){
77             regularCommunication.send(warningCode = 101, message = "speeding vehicle close to
78                 ego vehicle")
79         }
80     }

```

Figure 5.3: Code snippet sample for an application's Programming Interface

CHAPTER 6

Evaluations

In this chapter, we present a detailed discussion of our evaluation to show the efficacy of GRIFFIN. Firstly we present the testbed that has been implemented to build the GRIFFIN’s prototype.

6.1 Testbed Implementation

The SDK, comprising the library and toolchain, has been implemented over 700 LOC. We have collected 500 video clips of driving vehicles from various online sources for fine-tuning the built-in CV algorithms in GRIFFIN. We notice that there is a lack of real drones that can stably fly over highways, is fully controllable by developers (DJI only provides certain SDK), and is configurable with required computing resources, and lightweight that is power-wise adaptive to computing and communication. Hence, we simulate drone and car computing resources on Jetson TX2 and an edge server (Intel CPU I5 10th Gen with GTX 1650), respectively, and mimic road events by collecting 10 video clips using a DJI drone and our vehicles. Further, we mimic different network conditions (throughput, packet loss, dynamic bandwidth) using Iptables (or netem) to simulate the network conditions that can occur between drone and car in real-time. We set up a Redis cache [54] server on the drone Jetson TX2 resource to implement the in-memory cache utilized for storing event outcomes, pre-processing information, and detection results (frame-level, partition-level, and processing overhead) data.

6.2 Evaluations on event applications

We implement and measure the performance metrics of two applications, *Collision Avoidance* (denoted as application 1) and *Lane Changing* (denoted as application 2), using our SDK prototype. We simulate our system for detecting these applications using 3 *videos* and 2 *videos* containing 15 frames each from our collected dataset, respectively. For our case, we consider those frames where event takes place. We do not take cases of false positives or negatives for our evaluations. We aim to do that in future evaluations. We measure the drone’s energy consumption to process the videos in Joules by employing a Jetson-power-estimator. utility [55]. We use an external power monitor

(*HPE243i*) with Jetson Tx2 to monitor and collect instant energy consumption traces. We measure the average latency for each frame for each detection for the videos. The videos are processed at a rate of 25 frames per second (FPS).

Network Conditions	Bandwidth (Mbps)	Avg. Packet Loss (Application 1: Collision)	Avg. Packet Loss (Application 2: Lane Changing)
Status 1	100	40	24
Status 2	200	27	10
Status 3	400	11	12
Status 4	600	23	9
Status 5	800	21	13
Status 6	1000	16	6

Table 6.1: Packet Loss for varying network statuses

Through table 6.1, a dynamic variation can be noticed concerning transmitting packet loss as bandwidth increases. Bandwidth plays a critical role in determining network performance, impacting both latency and packet loss. We adopt six bandwidth statuses in which we evaluate our system.

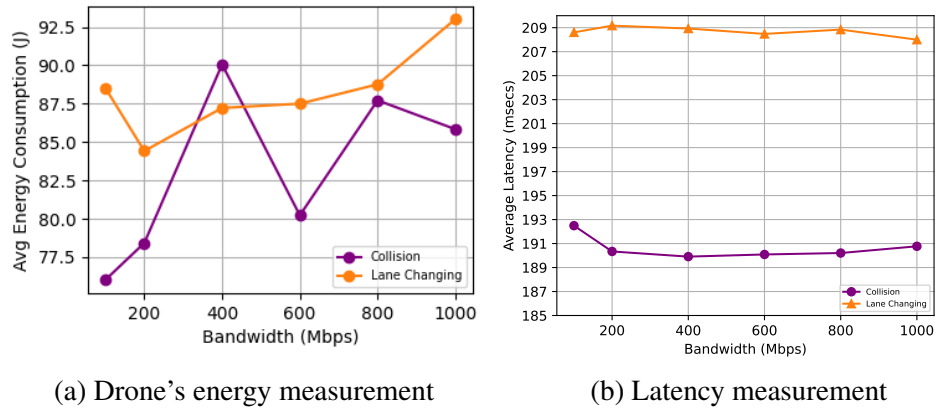


Figure 6.1: Evaluation on varying Network Performance on two event applications: Collision and Lane Changing

We first evaluate the GRIFFIN by randomly distributing the partitions to the drone and car. Figure 6.1 depicts the drone's average energy consumption and latency that is recorded to detect events during different network conditions. The corresponding graphs show that with a variation in network bandwidth of 450 Mbps, there is around 2% variation (2.11% and 1.93%) in average latency for both the applications, while the drone expends 0.078 KWs (Kilo-Watt-sec) energy and 0.0260 KW/frame (at 30 FPS) in average, with a mean variation of 12.5%. This evaluation shows controlled fluctuations of energy and latency measurements during implementation in realistic

scenarios. We can see bandwidth increases the packet throughput. As throughput ascends, there is a decrease in latency, showing ideal behavior. An increase in bandwidth often correlates with a decrease in packet loss, indicating improved reliability in data transmission. However, in a few cases when throughput increases, there is a spike in latency. This typically should be due to other processing latencies post-transmission.

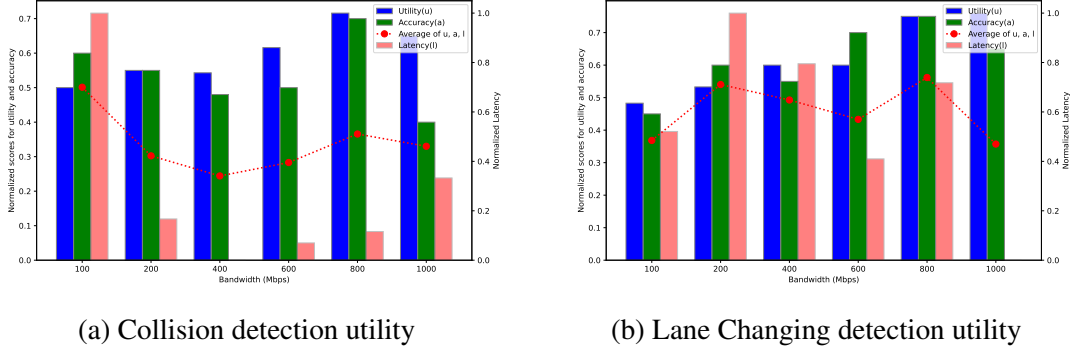


Figure 6.2: Trade-off between Utility, Accuracy and Latency while detecting the two events

Fig. 6.2 shows the QoS-wise utility and accuracy that is achieved while events are detected in image frames from the video data. It can be seen in both cases that median utility is achieved with minimal variations in accuracy. We adopt multiple frames with similar vehicle scenes from each video. Hence, this behavior can be seen for the accuracy achieved. For collision detection, we set l as 0.8, a as 0.3 and α as 0.6 (latency sensitive), with Utility-threshold set to 0.5 (all out of 1). For lane changing detection, we set l as 0.55, a as 0.8 and α as 0.70 (accuracy sensitive), with Utility-threshold set to 0.45 (all out of 1). For this work, we chose the utility threshold based on manual verification of the image frames. However, we aim to employ and provide a more robust mechanism (or an oracle) for CV developers to deciding on the utility-threshold in the future. From our empirical study in chapter 3, we notice that the battery drain rate increases from 2% per minute to 3-4% when the battery capacity is between 60% and 75%. Hence, we set the battery-drain rate threshold, r , as 0.7 for goal calculation. For both cases, we observe median goals that have been achieved for different network conditions.

6.3 Evaluation of unit modules

We thoroughly test the individual units to evaluate the efficacy of GRIFFIN's components and their reliability.

6.3.1 Fine-tuning CV modules for enhanced detections

As discussed in section 5.2, we fine-tune the existing Computer Vision (CV) modules to be suitable for detecting vehicles in BEV images. We utilize the Visdrone dataset and our own collected data to improve the accuracy of these modules. Post-fine-tuning, YOLOv5's and YOLOv8's detection performances have improved by 24.11% and 22.25% respectively.

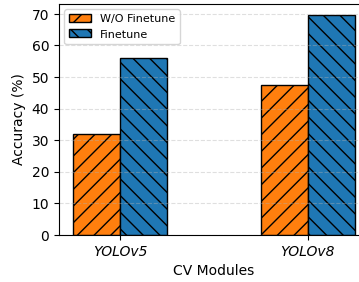


Figure 6.3: CV Modules Performance

6.3.1.1 Partitions distribution solver

Figures 6.4 and 6.5 provide adequate results to prove the efficacy of partitions distribution algorithm. GRIFFIN shows that drone's energy consumption reduces by over 1.25 x (25 %) when partitions distribution is employed against processing all partitions on drone for events. Surprisingly, we notice that the drone's energy consumption does not proportionally vary with the number of partitions. Energy is saved due to offloading equivalent tasks to cars, making car-based processing essential. The latter figure demonstrates the latency while the distribution algorithm is employed. For both applications, the latency is reduced by approximately 20-25%, giving optimal results in most of the cases. However, we see comparable latencies in the case of collision avoidance ($\downarrow$ 5% latency decrease). We suspect this behavior is due to the allocation of a similar(or more) quantity of partitions to the drone. This may also be attributed to the processing overload on the drone's resources. We aim to scrutinize such cases more deeply by conducting evaluations on more data and accordingly performing any modifications required to the algorithm.

6.3.2 Programming modules and toolchain

Extensibility of toolchain. Furthermore, we evaluated the efficiency of GRIFFIN's toolchain in generating executable files for drone and car operations (drone.exe and car.exe). Table 6.2 illustrates LoC in the backend of GRIFFIN toolchain, depicting compiling latency and generating executable files. The existing generation evaluations have been carried out on an Intel core i5 processing unit. Ideally, we expect the latency to be in a few 100s of milliseconds in an advanced graphical

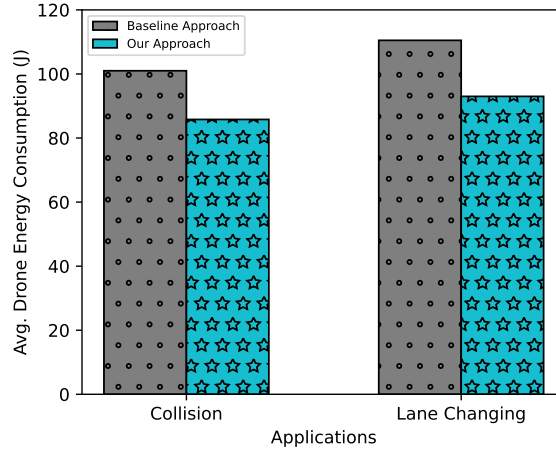


Figure 6.4: Drone’s energy w.r.t Partitions Distribution

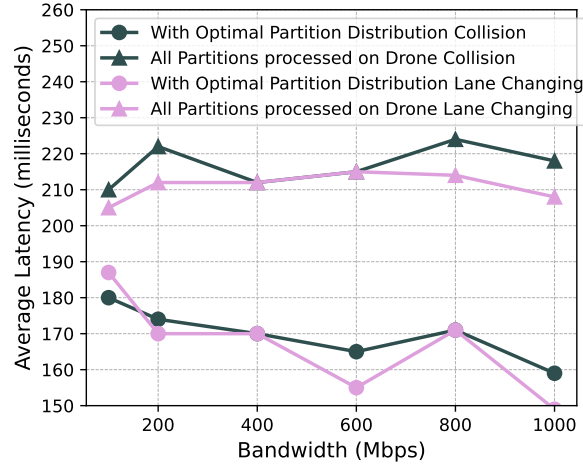


Figure 6.5: Latency w.r.t Partitions Distribution

processing unit. Nevertheless, the current results also demonstrate comparable outputs on a single processing platform.

The toolchain reduces LoC by over 1.12 x against the original codebase removing code redundancy by over 13%. This demonstrates that toolchain enables developers to quickly (re-)compile and build applications using single-time generated specs, thus supporting rapid Configure and Deploy cycles. The existing executables have been generated using one-file compiler and without modifying the actual size of finetuned CV modules. These results may be further improved through compiler and CV module optimizations. Also, our toolchain enables users to deploy complex CV processes during runtime, and how it is easily extensible with new features that can be reused across use cases.

Debugging. Debugging compiled libraries with workflow modules can be a challenging task.

	Orig.	With toolchain		Reduction
		Drone Exe	Car Exe	
LoC (approx.)	900	450	350	1.12 x
Compilation Latency (sec)	12.5	4.83	6.89	1.06 x
Memory Usage (MB)	600	300	240	1.11 x

Table 6.2: Evaluation on toolchain

Inorder to aid developers in debugging for other applications, the toolchain provides default executable implementations of a few applications with environment requirements in which they were compiled. These implementations provide a basic specification in the generated spec and can be iteratively reconfigured according to developer choice. This way, the toolchain avoids the compiling memory and time overload compared to the original compilation procedure.

Language Heterogeneity. Currently, the toolchain compiles libraries and generates executable services only in Python. Generating these in other languages is purely an implementation challenge, and we believe that the current design can be extended to generate code in other languages by utilizing compiler plugins.

6.3.3 Abstractable Components

GRIFFIN’s runtime abstracts several of the ingrained processes to make the runtime flexible and bridge the gap between GRIFFIN’s architecture and its users. In this regard, we assess how well the abstractions achieve their goals while maintaining simplicity, flexibility, and usability. For this, we utilize some key criteria (*Clarity*, *Encapsulation*, . . . , *User Feedback*) for evaluating our abstraction implementations.

Criteria (0-1)	Object Referencing and Hierarchical Inheritance	Relative notations and dimensional matching	API Operations
<i>Abstraction Level</i>	Medium	Low	High
<i>Clarity</i> ↑	0.9	0.7	0.8
<i>Encapsulation</i> ↑	0.85	0.75	1
<i>Cohesion and Coupling</i> ↑	0.8	0.65	0.95
<i>Usability and Effectiveness</i> (<i>User Feedback</i>) ↑	0.9	0.9	1
<i>Error probability</i> ↓	0.35	0.5	0.2
<i>Reconfigurations</i> ↑	0.7	0.6	0.05

Table 6.3: Evaluation metrics for abstractions implemented for GRIFFIN

Effectiveness. The abstractions provide adequate encapsulations, especially in the case of implementable API handlers. Each instance is loosely coupled, providing low inter-dependability and adequately cohesive to interact with system components and other abstracting instances. However, in certain cases, inconsistent implementations can be sorted out by providing clearly defined regulations during runtime routines. While GRIFFIN prototype visualizes two event applications, each provides unique components that arise during code realization. We believe that the implementation and production of more application instances would lead to a wider scope of reproducible abstractions among the applications.

Cost of abstractions. For each type of application, GRIFFIN interacts with the backend through a generalized programming interface. The interface is selected so that CV developers can opaquely reconfigure the abstraction instances. However, certain specialized and complex components, such as API configurations, need to be carried out by specialized operations by GRIFFIN developers.

Summary. Overall, these results illustrate that GRIFFIN supports optimal QoS-based event detection performance while ensuring a smooth user experience for efficiently building such applications.

CHAPTER 7

Discussion

In this chapter, we present the issues that we observed while designing GRIFFIN and implementing its prototype using the testbed setups.

Partitioning basis: We adopt uniform partitioning as the basis for GRIFFIN’s partitioning approach, where the image is split into certain equal parts. This static approach can be attributed to reduced processing latency during runtime. On the other hand, non-uniform partitioning generally splits the image into unequal partitions dynamically or focuses on certain specific regions each time based on prior results. We believe that this would incur additional latency for running extra processing steps and decision-making. Certain works that are based on non-uniform and dynamic partitioning [56, 57] argue that non-uniform partitioning with padding options achieves better object identification accuracy by capturing partial pixels that would normally get cut and not considered. We believe that since event detection is latency-sensitive, uniform partitioning is best suited to our case. Moreover, taking insights from detection results from the above-cited works, we implement IoU thresholding, which may alleviate any accuracy drops due to partial detections in partitions.

Deploying CV modules in drone and car We utilize YOLOv5 and YOLOv8 as CV modules for model inference in drone-captured images. We see that YOLOv5 generally takes more latency to detect objects (vehicles in our case). Conversely, YOLOv8 takes less time for model inference due to its optimized architecture. We choose to deploy YOLOv5 on drone-based Jetson resources and YOLOv8 on car-based superior resources due to the following reasons. 1) For edge computing scenarios with constraints on power and size, Jetson with YOLOv5 is highly suitable. For maximum performance and scalability, and where power consumption is less of a concern, a GPU-based computer with YOLOv8 is the optimal choice, [58, 59]. 2) Additionally, to satisfy QoS requirements using a drone that is mobile and a car that is stable and possesses robust hardware to achieve balanced latency and accuracy in real-time, the lightweight YOLOv5 is suitable for Jetson, and similarly, the heavyweight YOLOv8 for car-based processing.

Prerequisites to implement sensitive applications: GRIFFIN relies heavily on application-specific BEV data and CV modules for detections. For instance, there are a plethora of open-sourced CV algorithms for pothole detection. However, there is a lack of BEV pothole images and their

labels in the open-source community. Consequently, we see that CV modules pertinent to pothole detection are not trained for BEV data. Conversely, in the case of icy-road ahead detection, there is a lack of relevant CV algorithms and modules in the development community. And gathering icy-road BEV data is not a tedious task. In this regard, we can observe 1) Evaluating GRIFFIN for data and algorithm-sensitive applications requires adequate prerequisites and finetuning. 2) Certain plugin modifications can be carried out, such as employing learning techniques such as data-wise Active Learning and algorithm-wise Agile Training in GRIFFIN’s middleware framework and runtime. The following techniques can help alleviate data and algorithm-building issues.

More evaluations: We implement GRIFFIN as a prototype, primarily demonstrating its efficacy for event detections. Furthermore, its evaluations were carried out using small-scale and selective data. To make GRIFFIN’s runtime components robust and scale its programming support, we are of the opinion that more evaluations with data in diverse road scenes have to be carried out. In this regard, as we mentioned in the previous point, we highlight the need for research in the direction of capturing, collecting, and refactoring high-quality datasets and annotations, making them suitable to mimic realistic road event conditions.

CHAPTER 8

Future Directions

In the realm of drone-car collaboration, future research holds promise for technological advancement and broader application. As we look forward, addressing challenges and bridging technical gaps becomes crucial. In this chapter, we outline our roadmap, focusing on improving datasets, customizing algorithms, developing testbeds, and creating middleware support to facilitate seamless integration of drones and cars in collaborative applications.

8.1 Dataset and Testbed

Datasets are crucial for training and refining drone functionalities like navigation and obstacle detection, using varied data such as aerial images and sensor outputs. Testbeds are essential for providing controlled environments to evaluate drone performance with the necessary hardware for realistic applications, particularly in drone-car collaboration scenarios. This subsection discusses the necessity of datasets for training drone-car collaboration functionalities and the essentials of testbeds for evaluating performance and application viability.

8.1.0.1 BEV Datasets

CV-based tasks such as lane segmentation and detection, pedestrian and vehicle detection, speed estimation, and roadside unit (RSU) identification are crucial for drone functionalities, requiring extensive data collection and training under varied conditions to evaluate advanced modules [60].

- **BEV Data Gathering:** We see there is a lack of public datasets for many tasks listed above, and moreover, existing datasets for vehicle detection, lane detection, and pothole detection [61, 62] are typically captured from a human viewpoint rather than a bird’s-eye view. This gap highlights the need for research and development in collecting and generating BEV databanks, each task accompanied by high-quality meta-information in terms of resolution, object distribution, road-type climate-scene category, vehicle movement type, etc.

- **Gen-AI Based BEV Homographs:** Instead of manually recording and validating data, generative AI [63] can create near real-world BEV data from a few collected samples for training purposes. Further, these methods can also generate BEV-transformed homographs from manually recorded non-BEV video sets [64].
- **Advanced Annotations and Labeling:** Current BEV datasets like Mid-Air [65] provide trajectory data in different climatic conditions; they lack sufficient labeling for training real-time detections such as vehicles, RSUs, etc. Manually processing these proves to be time-consuming and exhaustive. Future work should use advanced AI techniques such as few-shot inference and multi-modal supervision [66] to streamline and automate the annotation and validation process for BEV data.
- **Drone-capacity & Scenario Based Datasets:** Training vision algorithms on data suited to drones' limited image-capture and computational abilities, like varied resolutions and pruned features, can enhance performance [67]. This method tests algorithms' adaptability to the drone's communicational and computational changes. Scenarios like lane changing and collision avoidance need BEV data from various heights and angles. However, existing datasets like VisDrone [51] and FloodNet [68] are limited in height and object size, making them unsuitable. Future work should focus on creating meticulously annotated BEV data for diverse scenarios.
- **Drone Sensor Outputs and Environmental Variables:** Modern drone systems are increasingly used to record sensor data and environmental variables collected from delivery and mock flights, such as images, battery status, altitude, velocity, location, signal strength, time of flight, etc. [69]. These datasets are crucial for training early decision-making and warning mechanisms in heterogeneous drone systems. Capturing sensory data from both drones and cars is essential for events like snow detection and speeding, as well as other collaborative applications.

8.1.0.2 Testbeds

The integration of drones into various sectors drives the demand for robust testbeds. These testbeds should enable performance testing, safety assurance, and algorithm refinement, particularly for applications like drone-car collaboration. Future efforts should prioritize adaptable testbeds that reflect diverse conditions to advance drone technology. Below, we discuss two such key areas for testbed development.

- **Real-World Testbed Development:** Real-world testbeds are crucial for advancing drone and car integration. They provide practical environments where developers can assess their

collaboration. For instance, drone fleets can be remotely deployed in designated areas via applications or web interfaces. These testbeds incorporate safety measures like predefined airspace limits and safety nets. Also, prototype testbeds can be used to simulate real-world scenarios, enabling iterative testing and refinement of hardware and software components. Recent advancements in autonomous drones utilizing affordable computing resources such as NVIDIA Jetson, Intel Myriad, and Snapdragon have significantly enhanced real-time computations and communication capabilities [70, 71, 72]. Existing programmable drone systems embedded with computing devices like Raspberry Pi require constant safety monitoring assistance and are heavy in size [73]. Therefore, lightweight drones (250-400 grams), equipped with real-time programming support and computational capabilities like Jetson devices [74], are essential for seamless operations under dynamic conditions.

- **Simulation Testbed:** Augmenting simulation testbeds with advanced features enables researchers to effectively test drone-car technologies in virtual environments before real-world deployment. As we have discussed previously, due to lack of 1) suitable hardware, 2) road conditions, 3) deployment infrastructure, and public safety concerns, simulations allow one to mimic real-time settings without compromising on performances in QoS metrics. Moreover, users can input or provide customized control commands such as frame selection, weather conditions, and event details to evaluate for extreme cases. Simulated testbeds can be orchestrated involving small-scale devices such as Jetson for drones and GPU-powered computer systems as car servers. Video sets of road events and road trajectories can be utilized to mimic capturing real road conditions. In such cases, applications can be tested by having data transmitted in varying network conditions established by TCP connections between the devices [75]. On the other hand, drone-based flight emulators (as drones) can be employed to test input videos by connecting to virtual servers (as cars). More such testbeds can be ideated and deployed to provide seamless testing environments to event application developers.

By addressing these areas, future research can significantly enhance the development and deployment of drones in realistic conditions for collaborative applications in real time.

8.2 CV algorithms for drone computations

Customizing computer vision (CV) algorithms is crucial for meeting specific drone application needs, such as accident tracking and surveillance. Additionally, optimizing the interaction between drones and cars through APIs and abstracting low-level details can enhance overall performance. Addressing these areas is essential for achieving efficient and effective drone-car collaboration across diverse applications. Below, we outline key aspects for realizing these advancements.

- **Customized CV Algorithms:** Existing vision algorithms released in public forums require significant codebase customization to be suitable for drone-based event detection applications, such as accident tracking in surveillance, and be integrated with drone software. Similarly, standalone modules are to be created and released as runtime packages for different applications. Additionally, standardizing inter-dependencies between algorithms, like “speed detection” from “vehicle-tracking” in multiple frames, will enable seamless implementation of multiple algorithms with minimal changes.
- **Pre-processing & Frame Augmentations:** As a drone camera captures image frames, pre-processing steps like partitioning sensitive parts and offloading them to a car are necessary, depending on battery power and network status. This approach enhances computational latency and is complemented by optimal detection accuracy. Such frame-level pre-processing & augmentation can be explored to achieve the goals of drone-car collaborative applications.
- **Multi-module Inference at the Edge:** Deploying multiple ML models at the edges (drone and car) can enhance performance and utility, unlike existing distributed paradigms [67] that implement limited models for inference.
- **Integration with APIs:** We discussed earlier the need to create particular CV algorithms for applications based on their requirements. These algorithms interact with commonly used components such as Storage cache, communication interfaces, network links, etc. Developers should encapsulate these processes into APIs for storage/retrieval, communication, status monitoring, etc respectively. Utilizing such pipelines enhances inter-algorithm implementation and system robustness.
- **Abstracting Low-level Details:** The complexity of unnecessary processing logics in algorithmic implementations can be alleviated by abstracting them into high-level concepts, allowing users to seamlessly interact with system modules and tune specific parameters at compile-time.

In summary, refactoring CV algorithms for application-specific needs and enabling smooth user interactions with system processing logic can be explored, paving the way for enhanced drone-car collaboration.

8.3 Advanced Sensing and Efficient Communication

The emergence of autonomous vehicles has led to advancements in various sensing technologies, including LiDAR, Radar, and wireless sensing [76]. We anticipate that in the future, more drones will come equipped with additional sensors by default. This enhancement can help overcome the

limitations of computer vision and provide more detailed information for ground vehicles. With the integration of new wireless communication technologies that combine sensing with communication, drones can better support ground vehicles in various scenarios, including sensing and tracing non-line-of-sight(non LoS) objects. Moreover, current peer-to-peer communication methods experience a significant drop in performance when both objects are moving quickly. This issue has long been a challenge for vehicle-to-vehicle communications. Addressing this issue would not only benefit the drone-car collaboration system but also offer more flexibility in making offloading decisions.

CHAPTER 9

Conclusion

This thesis presents a comprehensive approach to enhancing road event detection through the collaboration of drones and ground vehicles. By leveraging the strengths of both technologies, we aim to address the limitations of current detection systems, such as restricted sensing ranges and high operational costs. The proposed framework, designed as a distributed middleware system GRIFFIN, demonstrates significant potential for improving the speed, accuracy, and energy efficiency of real-time event detection.

Our work involved a thorough review of existing event detection technologies, highlighting their respective strengths and weaknesses. Motivated by these findings, we designed and implemented GRIFFIN to facilitate seamless drone-car collaboration. The empirical studies conducted showcase the superior performance of our approach in processing and detecting road events in real time.

The results from our evaluations indicate that GRIFFIN not only enhances the detection accuracy but also optimizes resource utilization, thereby extending the operational time of drones. This research contributes to the field by providing a scalable and efficient solution for real-time road event detection, paving the way for safer and more intelligent transportation systems.

Looking forward, the challenges and insights discussed throughout this thesis offer valuable directions for future work. Further advancements in drone technology, improved computational algorithms, and enhanced communication protocols between drones and GVs are essential to fully realize the potential of this collaborative approach. The ongoing development and integration of these technologies promise to significantly impact the future domain of smart road systems and autonomous driving.

APPENDIX

Relevant Paper Publications

A.1 Griffin: Facilitating Real-time Road Event Detection on Collaborative UAV and Ground Vehicle

Citation: K. Davuluri, S. Tamboli, N. Kothari, Z. Song, and M. Allison, “Griffin: Facilitating Real-time Road Event Detection on Collaborative UAV and Ground Vehicle. (to be submitted to Transactions on Vehicular Technology)

Abstract: Ground Vehicles (GVs) achieve micro-level driving decisions by sensing their environments with car-mounted sensors, limiting their ability to efficiently sense and react to road events. The new generation of vehicles, especially special-purpose vehicles, can launch, land, and charge unmanned aerial vehicles (UAVs), offering a more flexible and comprehensive view to guide GV in real time. However, UAVs have limited computing and energy resources, making them unsuitable for computationally intensive image processing tasks. To address this challenge, we introduce Griffin, a programming and runtime framework enabling real-time, drone-based road event detection by distributing image processing between a collaborative GV and UAV. The key novelty of Griffin lies in partitioning each real-time captured image frame and dynamically deciding where to process each partition based on its criticalness to the event, the real-time status of the drone-car system, and previous processing results. Griffin provides computer vision developers with an intuitive programming interface for calculating the criticalness of each partition and the per-partition processing logic on GV/UAV, along with well-packed computer vision modules fine-tuned for bird’s-eye view road event detection. Additionally, Griffin’s runtime features an optimizer that solves the partition distribution problem to minimize UAV energy consumption while meeting application requirements for latency and accuracy. We implemented three applications on Griffin for detecting road events, including collisions, speeding, and lane changes. By evaluating these applications’ performance on a testbed with a Jetson Tx2 as the drone and an edge server as the GV, we demonstrate that Griffin can achieve a better trade-off between energy efficiency, latency, and accuracy, making it a useful tool for developing real-time drone-car collaboration applications.

Commentary: This paper introduces Griffin as a middleware and distributed framework for

facilitating real-time road event detections.

A.2 Real-time Drone-Car Collaboration for Future Mobility: Vision and Challenge

Citation: S. Tamboli*, K. Davuluri*, K. Mottakin, Z. Song, "Real-time Drone-Car Collaboration for Future Mobility: Vision and Challenge" in 2024 IEEE Internet of Things Journal. (Under Review as of date 3rd August, 2024)

Abstract: Since the invention of automobiles, engineers have spent endless efforts to enhance their safety, operability, and fuel efficiency. The rapid development of electric vehicles (EVs) has transformed them into movable computing hubs, equipped with ample sensing, computing capabilities, and battery power. However, the placement of all the vehicle's sensors within its structure poses limitations to its sensing capability, consequently restricting potential applications. Meanwhile, unmanned aerial vehicles (UAVs or drones) offer aerial-perspective sensing capability from flexible viewpoints, processing which in a real time can complement cars' sensing and enable novel applications. In this paper, we introduce our vision of how the real-time collaboration of ground and aerial vehicles can enable innovative mobility solutions. We discuss the system architectures, potential usage scenarios, and enabling technologies for supporting drone-car collaboration. Furthermore, we examine the latest technical advancements in the field and shed light on the promising opportunities that lie ahead in the future. Overall, this article provides a comprehensive understanding of the potential and significance of real-time collaboration between drones and cars in revolutionizing future mobility, improving fuel efficiency, and enhancing driving safety.

Commentary: This paper presents our vision and existing challenges to realize the concept of Drone Car collaboration for future mobility.

A.3 BMEye: Public Health-Oriented Body Mass Index Monitoring using Commodity WiFi

Citation: K. Davuluri, K. Mottakin, Z. Song, J. Lu, M. Allison, "BMEye: Public Health-Oriented Body Mass Index Monitoring using Commodity WiFi", IEEE 9th World Forum on Internet of Things (WF-IoT), Aveiro, Portugal, 2023

Abstract: Body Mass Index (BMI) is a commonly used measure of body fat based on an individual's height and weight. Public health authorities and organizations monitor BMI at a population level to assess the prevalence and trends of under-weight and obesity within communities. This helps identify at-risk populations, evaluate the effectiveness of interventions, and develop targeted strategies to promote healthy weight and prevent associated health problems. However, there is a lack of BMI monitoring methods that do not require active cooperation or cause privacy concerns to human subjects. This paper introduces a BMI classification approach based on passive WiFi sensing. Our proposed approach is inspired by the observation that different body shapes cause dissimilar impacts on WiFi signal propagation, which can be reflected by Channel State Information (CSI) readings. Leveraging state-of-the-art machine learning algorithms, we developed a BMI sensing system that classifies the BMI of human subjects when they move in a WiFi range. Our evaluation results show that our approach could be an effective tool for public-health-oriented BMI monitoring, yielding a precision rate of 70.86%, a recall rate of 71.21 %, and an F1 score of 73.66%.

Commentary: This paper presents our work on estimating BMI of a person using WiFi and small-scale edge devices such as Raspberri Pi.

A.4 SWiDir: Enhancing Smartphone-based Walking Direction Estimation with Passive WiFi Sensing

Citation: K. Mottakin, K. Davuluri, Z. Song, "SWiDir: Enhancing Smartphone-based Walking Direction Estimation with Passive WiFi Sensing", 2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)

Abstract: Dead reckoning is a promising yet overlooked smart-phone based indoor localization technology. It relies on phone-mounted sensors for counting steps and estimating walking directions, requiring no massive deployment of additional sensors or landmarks. However, it suffers from the misalignment between phone's direction and human's movement direction, which makes the estimated walking direction unreliable and eventually leads to inaccurate location estimation. To solve this problem, this paper introduces SWiDir, an approach that calibrates walking direction by integrating active smartphone sensing with passive WiFi sensing. SWiDir deploys a few WiFi devices to form a correction zone, and use their WiFi Channel State Information (CSI) to infer human's movement inside the zone. We adopt the training-free WiFi Fresnel Zone model, and introduce an accurate and robust direction estimation model by exploring the geometrical relationship between the user's movement and its impact on the Fresnel zones. We built our testbed with 4 Raspberry Pis, forming a large correction zone and evaluated SWiDir across 5 participants in 2 different real environments. Our extensive experiments show that SWiDir achieves 8.89 degrees of average 75 percentile error in walking direction estimation, which is 64% lower than the state-of-the-art existing approaches.

Commentary: This paper focuses on enhancing smartphone based walking direction estimation methods using passive WiFi sensing techniques.

References

- [1] K. Buchholz, “Commercial drones - projected growth,” 2024, accessed: 2024-07-28. [Online]. Available: <https://www.statista.com/chart/17201/commercial-drones-projected-growth/>
- [2] M.-Y. Yu, “Tackling limited sensing capabilities for autonomous driving,” Ph.D. dissertation, 2022.
- [3] DJI, “Dji mini2 se drone,” 2024, accessed: May 13, 2024. [Online]. Available: <https://www.dji.com/mini-2-se>
- [4] X. Fu, G. Wei, X. Yuan, Y. Liang, and Y. Bo, “Efficient yolov7-drone: an enhanced object detection approach for drone aerial imagery,” *Drones*, vol. 7, no. 10, p. 616, 2023.
- [5] ChatGPT, “Ford model t,” <https://corporate.ford.com/>, 2024, accessed: 2024-08-01. [Online]. Available: <https://corporate.ford.com/>
- [6] J. Zhang and K. B. Letaief, “Mobile edge intelligence and computing for the internet of vehicles,” *Proceedings of the IEEE*, vol. 108, no. 2, pp. 246–261, 2019.
- [7] J. Nidamanuri, C. Nibhanupudi, R. Assfalg, and H. Venkataraman, “A progressive review: Emerging technologies for adas driven solutions,” *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 2, pp. 326–341, 2021.
- [8] M. Quarterly, “Recap: Toward carbon neutral mobility,” 2023.
- [9] “What is vision zero?” 2022.
- [10] “Recap: Toward carbon neutral mobility,” 2021.
- [11] Drone Brands, “Drone statistics: Unveiling the latest trends and insights,” 2024, accessed: 2024-05-28.
- [12] “Custom dcap vehicle-mounted autonomous drone charging dock,” 2024, accessed: 2024-05-28.
- [13] P. B. O. et al., “Vehicle-mounted aerial drone container,” USA Patent 11 465 740, Oct 11, 2022.
- [14] O. R. et al., “Systems and methods for operating drones in proximity to objects,” USA Patent 20 230 013 444, Jan 19, 2023.

- [15] J. Yoon, I. Kim, W. Chung, and D. Kim, "Fast and accurate car detection in drone-view," in *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 2016, pp. 1–3.
- [16] J. Fontaine, T. De Waele, A. Shahid, E. Tanghe, P. Suanet, W. Joseph, J. Hoebeke, and E. De Poorter, "Drone-mounted rfid-based rack localization for assets in warehouses using deep learning," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 1–4.
- [17] M. Hamza, A. Jehangir, T. Ahmad, A. Sohail, and M. Naeem, "Design of surveillance drone with x-ray camera, ir camera and metal detector," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 111–114.
- [18] B. McNally and G. L. Bradley, "Re-conceptualising the reckless driving behaviour of young drivers," *Accident Analysis & Prevention*, vol. 70, pp. 245–257, 2014.
- [19] J. Gao, H. Chen, K. Dave, J. Chen, and D. Jia, "Fuel economy and exhaust emissions of a diesel vehicle under real traffic conditions," *Energy Science & Engineering*, vol. 8, no. 5, pp. 1781–1792, 2020.
- [20] E. Tzirakis, F. Zannikos, and S. Stournas, "Impact of driving style on fuel consumption and exhaust emissions: defensive and aggressive driving style," in *Proceedings of the 10th international conference on environmental science and technology*. Global Network for Environmental Science and Technology (Global-NEST), 2007, pp. 1497–1504.
- [21] P. Kitjacharoenchai, B.-C. Min, and S. Lee, "Two echelon vehicle routing problem with drones in last mile delivery," *International Journal of Production Economics*, vol. 225, p. 107598, 2020.
- [22] H. Y. Jeong, B. D. Song, and S. Lee, "Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones," *International Journal of Production Economics*, vol. 214, pp. 220–233, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925527319300106>
- [23] W. Gao, J. Luo, W. Zhang, W. Yuan, and Z. Liao, "Commanding cooperative ugv-uav with nested vehicle routing for emergency resource delivery," *IEEE Access*, vol. 8, pp. 215 691–215 704, 2020.
- [24] N. M. Imran, S. Mishra, and M. Won, "A-vrpd: Automating drone-based last-mile delivery using self-driving cars," *IEEE Transactions on Intelligent Transportation Systems*, 2023.
- [25] A. Beg, A. R. Qureshi, T. Sheltami, and A. Yasar, "Uav-enabled intelligent traffic policing and emergency response handling system for the smart city," *Personal and Ubiquitous Computing*, vol. 25, no. 1, pp. 33–50, 2021.
- [26] L. B. López, N. Van Manen, E. Van der Zee, and S. Bos, "Dronealert: Autonomous drones for emergency response," *Multi-Technology Positioning*, pp. 303–321, 2017.

- [27] A. Albanese, M. Nardello, and D. Brunelli, “Low-power deep learning edge computing platform for resource constrained lightweight compact uavs,” *Sustainable Computing: Informatics and Systems*, vol. 34, p. 100725, 2022.
- [28] “DJI Mini 2 Intelligent Flight Battery,” <https://store.dji.com/product/mini-2-intelligent-flight-battery>, 2024, accessed: 2024-03-04.
- [29] H. Wang, H. Zhao, J. Zhang, D. Ma, J. Li, and J. Wei, “Survey on unmanned aerial vehicle networks: A cyber physical system perspective,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1027–1070, 2019.
- [30] C. Peng, K. Zhao, A. Wiliem, T. Zhang, P. Hobson, A. Jennings, and B. C. Lovell, “To what extent does downsampling, compression, and data scarcity impact renal image analysis?” in *2019 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2019, pp. 1–8.
- [31] H. Hsiang, K.-C. Chen, P.-Y. Li, and Y.-Y. Chen, “Analysis of the effect of automotive ethernet camera image quality on object detection models,” in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)*. IEEE, 2020, pp. 021–026.
- [32] N. Bhowmik, J. W. Barker, Y. F. A. Gaus, and T. P. Breckon, “Lost in compression: the impact of lossy image compression on variable size object detection within infrared imagery,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 369–378.
- [33] J. Zhou, C.-M. Vong, Q. Liu, and Z. Wang, “Scale adaptive image cropping for uav object detection,” *Neurocomputing*, vol. 366, pp. 305–313, 2019.
- [34] C.-E. Ko, P.-H. Chen, W.-M. Liao, C.-K. Lu, C.-H. Lin, and J.-W. Liang, “Using a cropping technique or not: Impacts on svm-based amd detection on oct images,” in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 199–200.
- [35] S. O’Gara and K. McGuinness, “Comparing data augmentation strategies for deep image classification,” 2019.
- [36] X. Yuan, Z. Zhang, C. Feng, Y. Cui, S. Garg, G. Kaddoum, and K. Yu, “A dqn-based frame aggregation and task offloading approach for edge-enabled iomt,” *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 3, pp. 1339–1351, 2022.
- [37] C.-F. Wang, Y.-K. Lin, and J.-C. Chen, “A cooperative image object recognition framework and task offloading optimization in edge computing,” *Journal of Network and Computer Applications*, vol. 204, p. 103404, 2022.
- [38] A. Coletta, F. Giorgi, G. Maselli, M. Prata, D. Silvestri, J. Ashdown, and F. Restuccia, “A 2-uav: Application-aware content and network optimization of edge-assisted uav systems,” in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

- [39] T. Li, J. Sun, Y. Liu, X. Zhang, D. Zhu, Z. Guo, and L. Geng, “Esmo: Joint frame scheduling and model caching for edge video analytics,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 8, pp. 2295–2310, 2023.
- [40] N. Aldahoul, H. A. Karim, A. Q. M. Sabri, M. J. T. Tan, M. A. Momo, and J. L. Fermin, “A comparison between various human detectors and cnn-based feature extractors for human activity recognition via aerial captured video sequences,” *IEEE Access*, vol. 10, pp. 63 532–63 553, 2022.
- [41] P. Labenski, M. Ewald, S. Schmidtlein, and F. E. Fassnacht, “Classifying surface fuel types based on forest stand photographs and satellite time series using deep learning,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 109, p. 102799, 2022.
- [42] U. Azmat, S. S. Alotaibi, M. Abdelhaq, N. Alsufyani, M. Shorfuzzaman, A. Jalal, and J. Park, “Aerial insights: Deep learning-based human action recognition in drone imagery,” *IEEE Access*, 2023.
- [43] D. Avola and D. Pannone, “Magi: multistream aerial segmentation of ground images with small-scale drones,” *Drones*, vol. 5, no. 4, p. 111, 2021.
- [44] A. Purwanto, D. H. Budiarti, F. N. Purnamastuti, I. Y. Tanasa, Y. Guno, A. S. Yunata, M. Wibowo, A. Hidayat, and D. Dirgahayu, “Image segmentation in aerial imagery: A review,” *SINERGI*, vol. 27, no. 8, pp. 343–360, 2023.
- [45] A. S. Chakravarthy, S. Sinha, P. Narang, M. Mandal, V. Chamola, and F. R. Yu, “Dronesegnet: robust aerial semantic segmentation for uav-based iot applications,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4277–4286, 2022.
- [46] W. Rong, Z. Li, W. Zhang, and L. Sun, “An improved canny edge detection algorithm,” in *2014 IEEE International Conference on Mechatronics and Automation*, 2014, pp. 577–582.
- [47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 8024–8035.
- [48] Ultralytics, “YOLOv5,” <https://github.com/ultralytics/yolov5>, 2020.
- [49] G. Jocher, A. Chaurasia, and J. Qiu, “Yolo by ultralytics. github,” *GitHub*, January, 2023.
- [50] J. Zhang, Z. Chen, G. Yan, Y. Wang, and B. Hu, “Faster and lightweight: An improved yolov5 object detector for remote sensing images,” *Remote Sensing*, vol. 15, no. 20, p. 4974, 2023.
- [51] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, “Detection and tracking meet drones challenge,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2021.

- [52] Roboflow, Inc., “Roboflow: Annotate and label objects for object detection,” <https://roboflow.com>, 2024, accessed: 2024-05-15.
- [53] DJI, *DJI Drones Pro: Advanced Technology for Professional Aerial Photography*. Shenzhen, China: DJI Innovations, 2024, accessed: 2024-08-02. [Online]. Available: <https://www.dji.com/products/professional-drones>
- [54] S. Sanfilippo and P. Noordhuis, “Redis: In-memory data structure store,” <https://redis.io>, 2024, accessed: 2024-05-15.
- [55] OpenDR EU, “Jetson power,” 2024, accessed: 2024-07-15. [Online]. Available: https://github.com/opendr-eu/jetson_power/
- [56] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, “Flexible high-resolution object detection on edge devices with tunable latency,” in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 559–572.
- [57] Y. Yan, Y. Jin, S. Zhang, F. Cheng, Z. Qian, and S. Lu, “Focus! provisioning attention-aware detection for real-time on-device video analytics,” in *2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2022, pp. 271–279.
- [58] Roboflow, “Deploy yolov8 object detection models to nvidia jetson,” 2024, accessed: 2024-08-02. [Online]. Available: <https://roboflow.com/deploy-yolov8-jetson>
- [59] Ultralytics, “Yolov5 training and deployment on nvidia jetson platforms,” 2024, accessed: 2024-08-02. [Online]. Available: <https://github.com/ultralytics/yolov5/discussions/8386>
- [60] F. Safavi, I. Ali, V. Dasari, G. Song, T. Zhu, and M. Rahnemoonfar, “Efficient semantic segmentation on edge devices,” *arXiv preprint arXiv:2212.13691*, 2022.
- [61] P. A. Chitale, K. Y. Kekre, H. R. Shenai, R. Karani, and J. P. Gala, “Pothole detection and dimension estimation system using deep learning (yolo) and image processing,” in *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 2020, pp. 1–6.
- [62] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.
- [63] E. V. P. Beyan, A. G. C. Rossy *et al.*, “A review of ai image generator: influences, challenges, and future prospects for architectural field,” *Journal of Artificial Intelligence in Architecture*, vol. 2, no. 1, pp. 53–65, 2023.
- [64] S.-Y. Cao, R. Zhang, L. Luo, B. Yu, Z. Sheng, J. Li, and H.-L. Shen, “Recurrent homography estimation using homography-guided image warping and focus transformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 9833–9842.

- [65] M. Fonder and M. V. Droogenbroeck, “Mid-air: A multi-modal dataset for extremely low altitude drone flights,” in *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, June 2019.
- [66] T. Liang, W. Pan, H. Bao, X. Fan, and H. Li, “Bird’s eye view semantic segmentation based on improved transformer for automatic annotation,” *KSII Transactions on Internet and Information Systems (TIIIS)*, vol. 17, no. 8, pp. 1996–2015, 2023.
- [67] K. Huang and W. Gao, “Real-time neural network inference on extremely weak devices: agile offloading with explainable ai,” in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 200–213.
- [68] M. Rahnemoonfar, T. Chowdhury, A. Sarkar, D. Varshney, M. Yari, and R. Murphy, “Floodnet: A high resolution aerial imagery dataset for post flood scene understanding,” *arXiv preprint arXiv:2012.02951*, 2020.
- [69] T. A. Rodrigues, J. Patrikar, A. Choudhry, J. Feldgoise, V. Arcot, A. Gahlaut, S. Lau, B. Moon, B. Wagner, H. S. Matthews, S. Scherer, and C. Samaras, “Data Collected with Package Delivery Quadcopter Drone,” 7 2020.
- [70] jetsonSeries, “jetsonseries,” 2024, accessed: 2024-06-05. [Online]. Available: <https://developer.nvidia.com/embedded-computing>
- [71] intelMyriad, “intelmyriad,” 2024, accessed: 2024-06-05. [Online]. Available: <https://www.intel.com/content/www/us/en/products/sku/204770/intel-movidius-myriad-x-vision-processing-unit-0gb/specifications.html>
- [72] snapdragon, “snapdragon,” 2024, accessed: 2024-06-05. [Online]. Available: <https://www.modalai.com/pages/snapdragon-flight>
- [73] C. Tech, “Clover,” 2024, accessed: 2024-06-04. [Online]. Available: <https://clover.coex.tech/en/>
- [74] jetsonTx2, “jetson,” 2024, accessed: 2024-06-05. [Online]. Available: <https://developer.nvidia.com/embedded/jetson-tx2>
- [75] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-peer communication across network address translators.” in *USENIX Annual Technical Conference, General Track*, 2005, pp. 179–192.
- [76] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, “Sensor and sensor fusion technology in autonomous vehicles: A review,” *Sensors*, vol. 21, no. 6, p. 2140, 2021.