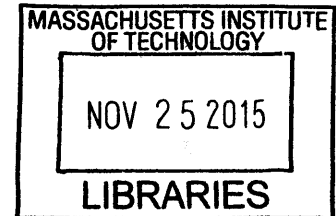


Quadrasense

Immersive UAV-based Cross-Reality Navigation of Environmental Sensor Networks

Vasant Ramasubramanian
BSEE, University of Texas Dallas, 2002

ARCHIVES



Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements of the degree of
MASTER OF SCIENCE
at the **MASSACHUSETTS INSTITUTE OF TECHNOLOGY**

September 2015

© 2015 Massachusetts Institute of Technology. All rights reserved.

Signature redacted

Author: **Vasant Ramasubramanian**
Program in Media Arts and Science
27-Aug-2015

Signature redacted

Certified by: **Dr. Joseph A. Paradiso**
Professor of Media Arts and Sciences
MIT Program in Media Arts and Sciences

Signature redacted

Accepted by: **Pattie Maes**
Academic Head
MIT Program in Media Arts and Sciences

Quadrasense

Immersive UAV-based Cross-Reality Navigation of Environmental Sensor Networks

Vasant Ramasubramanian

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning
on August 27th 2015,
in partial fulfillment of the requirements for the degree of Master of Science at the
Massachusetts Institute of Technology.

Abstract

Quadrasense explores futuristic applications in environmental sensing by integrating ideas of cross-reality with semi-autonomous sensor-aware vehicles. The cross-reality principals of telepresence, augmented reality, and virtual reality are enabled through an Unmanned-Aerial-Vehicle, a specialized imaging system, a Head-Mounted-Display, a video game engine and a commodity computer. Users may move between any of the three modes of interaction, in real-time, through a singular visual interface.

Utilizing an environment built with video game technology, a system was developed that can track and move a UAV in the physical world, towards goals of sensing, exploration and visualization. This application expands on the use of video games engines for simulation by directly joining the virtual and real worlds.

Thesis Supervisor:

Dr. Joseph A. Paradiso
Professor of Media Arts and Sciences

Quadrasense

Immersive UAV-based Cross-Reality Navigation of
Environmental Sensor Networks

Vasant Ramasubramanian

The following served as a reader for this thesis

Signature redacted

Dr. James W. Bales
Associate Director, Edgerton Center
Massachusetts Institute of Technology

Quadrasense

Immersive UAV-based Cross-Reality Navigation of
Environmental Sensor Networks

Vasant Ramasubramanian

The following served as a reader for this thesis

Signature redacted

Dr. Srinivas Ravela
Principal Scientist, E.A.P.S.
Massachusetts Institute of Technology

Acknowledgements

To my parents and family for their support and encouragement.

Joe Paradiso, my advisor, for your patience, flexibility, understanding, encouragement and enthusiasm. To my readers Jim Bales and Sai Ravela for being gracious with your time, for stimulating conversation and technical feedback on this thesis.

To Amna Carreiro, Keira Horowitz and Linda Peterson for countless amounts of help, keeping me on track and helping me to navigate many difficult situations.

Brenda Doss and Rick Richard for always being there, for your unending support, word of encouragement and for your help polishing this work.

Ermal Dreshaj for being there through tough times, sagely advice and direct contributions to this project.

Poras Balsara and Bob Hunt for your recommendations and help, even under extremely short notice.

Vijay, Dan Baruzinni, Mukesh Cooblal, Kun-Pei Chen, Jack Lavier, Bob Lowell, Gregg Mack and Mike Sly for encouraging me to take the plunge and go back to graduate school.

Jim Barabas, Kamal and Jennifer Broutin Farah, Pragun Goyal, Sunny Jolly for illuminating conversations, advice and sincere friendship.

To J. Turner Whited for your words of wisdom, perspective, advice about people and research.

To everyone in Responsive Environments, especially Gershon Dublon, Donald Haddad, Brian Mayton and Spencer Russell for Tidmarsh, the underpinnings of this project.

To Glorianna Davenport and everyone involved with the Tidmarsh restoration project, thank you for giving me an opportunity to contribute and for your support.

Table of Contents

Introduction	7
1.0 Sensors: Past, Present and Future.....	8
1.1 Terrestrial Sensing.....	9
1.2 Sensor Data as Agents for Action.....	10
1.3 UAVs as Semi-Autonomous Sensing Agents.....	12
1.4 UAVs as Vehicles for Collaborative Exploration.....	15
1.5 Video game technology for UAV control.....	15
Quadrasense Concept	16
2.1 Overview.....	17
Motivation	21
3.1 Limits of Existing Sensor Visualizations.....	22
3.2 Applications of Real-time Sensor Visualizations using UAV agents.....	23
Related Work	26
4.1 Overview.....	27
4.2 Telepresence / Augmented Reality / CAVE.....	27
4.3 Omnidirectional Imaging.....	29
4.4 Omnidirectional Imaging with UAVs.....	31
4.5 Multi-rotor UAVs and mixed-domain sensing.....	33
4.6 Video game engine use for immersive visualization.....	34
Quadrasense High-Level Design	36
5.1 Overview.....	37
5.2 The UAV.....	38
5.3 Camera Complex.....	39
5.4 Syphon GPU texture sharing.....	41
5.5 UniMAV: UAV Telemetry and Guidance Control.....	42
5.6 HMD and Unity environment.....	43
Quadrasense Implementation Details	44
6.1 Camera.....	45
6.1 Low-latency Video.....	46
6.2 Oculus Head Position and Dewarp communication.....	49
6.3 OpenWRT Router.....	50
6.4 Syphon Video.....	50
6.5 UniMAV and Software-In-The-Loop Simulator.....	52
6.6 Unity.....	55
Quadrasense Results	61
7.1 Quadrasense Unity Views.....	62
7.2 Data captured from Live Flight.....	65
Conclusions and Future Work	70
8.1 Contributions.....	71
8.2 Future Work.....	72
References	74

Chapter 1

Introduction

1.0 Sensors: Past, Present and Future

We are continually evolving our understanding of the world through a variety of means including the scientific process of hypothesis, observation, data collection and analysis. In this pursuit we have used a number of sensing mechanisms, often in novel ways, to record phenomena of interest. With the progression of technology, the once simplistic and limited view of sensors as data collection instruments is shifting. The domain of sensors and the informed world are becoming more intertwined. Ubiquitous sensing also ushers in the challenges of information collection and meaningful data extraction, previously problems of big science.

A predominant use of sensors is in terrestrial sensing. This application utilizes sensors to record some limited number of parameters in geographies such as oceans, volcanos, jungles and so on. Historically this usage has relied on sensors as data logging devices. However, with advances in electronics, power storage, and signal processing capability, sensor nodes can now autonomously inform users about events of interest as they occur.

The concept of a sensor network moving from a passive informant to an active participant in the world is becoming a recognized perspective. If sensors can interact with and exert control of physical instruments in response to measured data, we can view this network as possessing the concept of agency. In looking towards the future, integrating the extensive informative capability of modern sensors and emerging capabilities of autonomous vehicles will result in combined agency, blending the best of both disciplines. Quadrasense aims to explore this unique and largely unrealized combination. Figure 1-1 illustrates a conceptual view of problem solving ability vs. increasing capability of sensor networks.

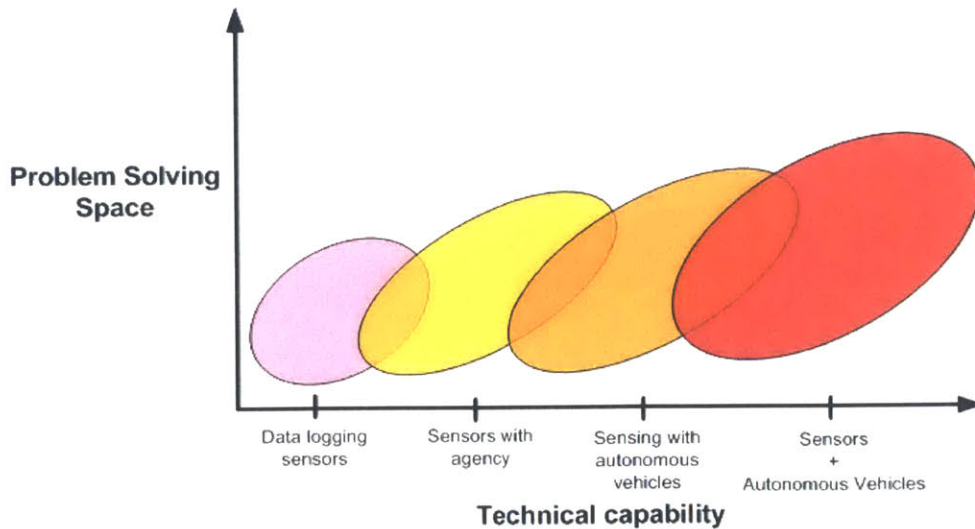


FIGURE 1-1: CONCEPTUAL VIEW OF PROBLEM SOLVING ABILITY OF SENSOR NETWORKS VS. SENSOR NETWORK CAPABILITY

1.1 Terrestrial Sensing

Individuals from scientists, researchers and engineers to farmers, environmental agencies, industrialists and energy producers are relying on environmental sensors to provide key insight to their respective needs. With low-power semiconductors, advanced wireless communication and large non-volatile storage, sensor nodes now have the ability to measure, store and transmit tremendous amounts of information, all in real-time. Early environmental research work at Great Duck Island, conducted by UC Berkeley, College of Atlantic and Intel[1] helped explore key aspects of hardware, software and system-architecture necessarily for successful environmental sensing. This work helped pave the road for future low-power sensor node designs, and sensing applications.

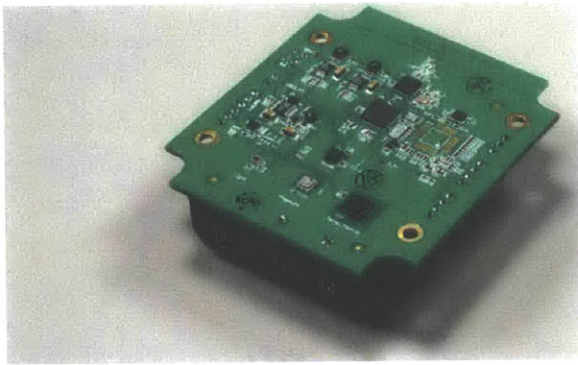


FIGURE 1-2: RESPONSIVE ENVIRONMENTS SENSOR NODE .

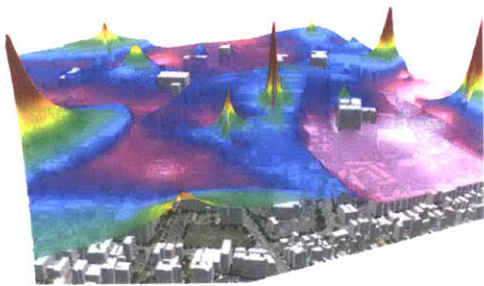


FIGURE 1-3: ACTUAL VOC MEASUREMENTS, CITY OF LOS ANGELES, 2014 [VALARM]

A typical, modern environmental sensor node, such as the one shown in Figure 1-2, developed by the Responsive Environments group at MIT MediaLab, has the ability to capture among other parameters, light levels, temperature, humidity, pressure, physical orientation (via an accelerometer) and voltage levels at very low electrical power levels. This particular node can communicate to a central network concentrator and also form a mesh network using custom software interfacing to the on-board 2.4 GHz Radio Frequency IC. While this node was designed with a fairly generic set of measurement capabilities, other node designs can measure a more specialized spectrum of data including toxic gasses, radiation, pH, soil moisture and so on.

According to a recent survey publication “Environmental Sensor Networks: A revolution in earth science system?”[2], sensor networks have morphed from simple data logging instruments to a collective system of intelligence which will become a

standard tool for all aspects of environmental science. Environmental sensing has in fact already expanded from beyond traditional remote sensing applications to include data collection for urban environments, through the use of networked nodes distributed in modern buildings. Figure 1-3 shows distribution of Volatile-Organic-Compounds (VOCs) taken from such a network, in the City of Los Angeles.

1.2 Sensor Data as Agents for Action

Advanced sensor networks, with increased intelligence, are now more tightly coupled into explorations, experiments and engineering than ever before. For example, in scientific research, sensors are often used at the outset of an experiment to provide data for informed decisions about necessary fieldwork, thus driving both future sensor usage and additional experiments. Other industries such as ecological engineering, agriculture, and water treatment often use environmental sensor data to validate that a process is working as expected.

In a farming application where deployment of fertilizer needs to be precisely distributed, ground-based phosphorous and nitrate sensors can indicate if there is unexpected run-off or improper dispersal. In a water processing facility, sensors can be used to verify that all harmful pathogens and toxic substances are removed or within established norms. Figure 1-4 depicts these two typical uses for sensor networks.

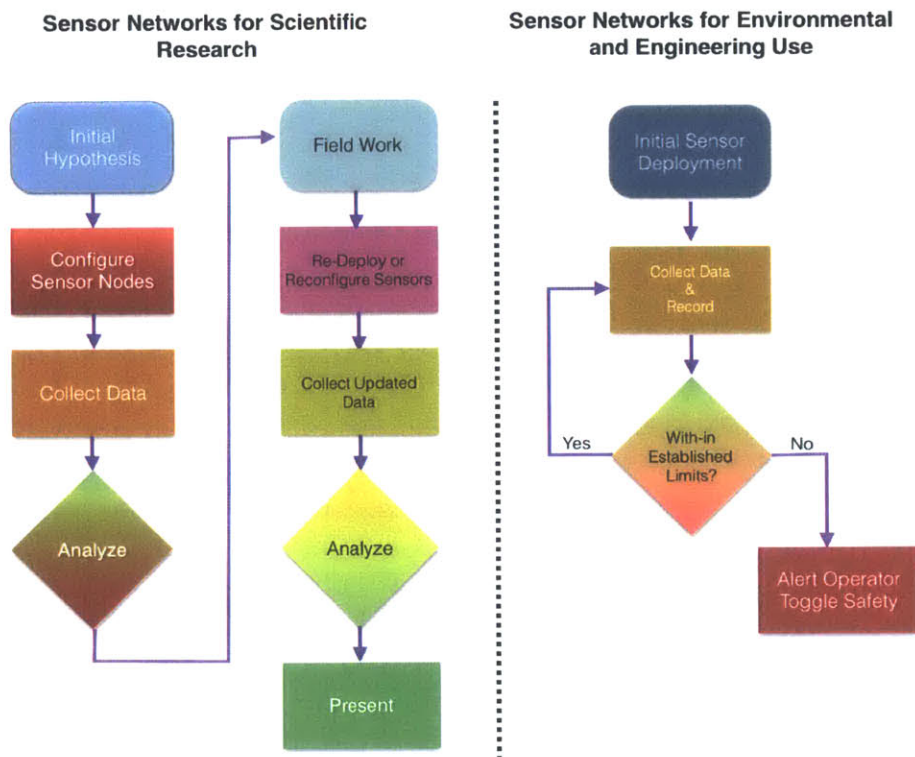


FIGURE 1-4; COMPARISON OF SENSOR NETWORK USAGE IN RESEARCH VS. ENGINEERING

Thus in these examples environmental sensor networks are used in a more engineering-oriented or process control sense verifying that measured data is not outside established standards and alerting users if these condition are violated.

However, observing both modern research and engineering aspects of environmental sensing, we can consider the wireless sensor network and the “informed data” as jointly active participants in the experiment or research process. This high-level concept of sensor networks as “actors” or “agents” as a proxy for action, be it informing humans for further investigation or commanding other semi-autonomous agents to carry out a task, is emerging. This field is called Wireless Sensor and Actor Networks or “WSAN” by researchers in the Electrical Engineering Department at Georgia Tech [3].

Wireless Sensor and Actor Networks (WSANs)

- Sensors**
 - Passive elements sensing from the environment
 - Limited energy, processing and communication capabilities
- Actors**
 - Active elements acting on the environment
 - Higher processing and communication capabilities
 - Less constrained energy resources (Longer battery life or constant power source)

Sensors + Actors → **WSANs**

IFA'2004 26

Components of Sensor & Actor Nodes

Sensor Node

Actor Node

- Actor Node Operation**
 - Actors receive digital data from sensors
 - Digital data is processed
 - Controller generates action commands based on data
 - Digital action command is converted to analog signal
 - Action is performed

IFA'2004 31

FIGURE 1-5: HIGH-LEVEL VIEW OF SENSOR AND ACTOR NODES, WSAN CONCEPT [AKYILDIZ & KASIMOGLU]

As sensor networks evolve “intelligence”, distribution of processing and control is yet another dimension to explore. Perhaps one of the more interesting aspects of the WSANs is how they can exert a physical influence by controlling machines such as robots in the sensed landscape. By introducing robotic agents under joint human and WSAN control, we can potentially expand exploration and problem solving from one under sole human command to a hybrid domain where human needs are translated into parameters for the WSAN to carry out. A multitude of autonomous robotic agents to aid in sensing have been already been explored as part of WSAN research. These actors include remote-controlled cars/trucks, Unmanned-Aerial-Vehicles (UAV) and even sub-miniature, limited functions robots.



FIGURE 1-6: ROBOTIC ACTORS TESTED IN WSAN RESEARCH, TOP ROW: RC HELICOPTER, RC ALL-TERRAIN-VEHICLE, BOTTOM ROW: MINIATURE ROBOTS [AKYILDIZ]

1.3 UAVs as Semi-Autonomous Sensing Agents

The use of UAVs, especially multi-rotor vehicles such as quad-copters, has attracted significant attention for use as agents in sensing networks. While aircraft such as fixed-wing planes and gliders offer significant benefit in terms of longevity and high-altitude observability, multi-rotor vehicles have tremendous maneuverability, geographical coverage, payload carrying capacity and flexible frame configuration. A typical multi-rotor can be launched from small platforms, and can carry sensor packages, cameras and radios for extended range based transmission. Combined with developments in battery storage technology, lower cost Brushless-DC motors and low-noise Inertial Measurement Units

(IMU), multi-rotors span from diminutive in size to formidable with commensurate capability.



**16-ROTOR AGRICULTURAL UAV
FIGURE 1-7**



NIXIE, WRIST-WEARABLE QUADROTOR W/CAM

Many innovative sensing platforms are now looking to incorporate UAVs as agents. Lockheed Martin has demonstrated a sophisticated wireless sensor network called “SPAN” – “Self Powered Ad-hoc Network” that dynamically routes sensor traffic without the need for a fixed central node and autonomously notifies users of potential threats [4]. In a newer configuration of SPAN, the intelligent sensor mesh informs a UAV agent to use its more precise instrumentation to investigate potential threats. All of the sensor and UAV coordination is done completely without human intervention. While this application is



FIGURE 1-8: LOCKHEED-MARTIN SPAN MOCK-UP [LMCO]

defense-oriented, the case towards integration of sensor technology and semi-autonomous UAVs for environmental research is equally strong.

Several projects underway such as [5,6,7,8] are leading efforts to integrate UAVs in innovative environmental sensing applications. These projects carry payloads that enable a diverse set of measurements from the monitoring of soil erosion, ice crystal formation in clouds and urban water quality to algae blooms in ocean environments. Figure 1-9 shows results from some of these experiments. Along with many of these sensing applications is the tight integration of high-resolution 2D and 3D imagery. If image capture and depth sensing is real-time and compelling enough, the agency concept of UAVs for sensing can now be extended to include the capabilities of telepresence and augmented reality.

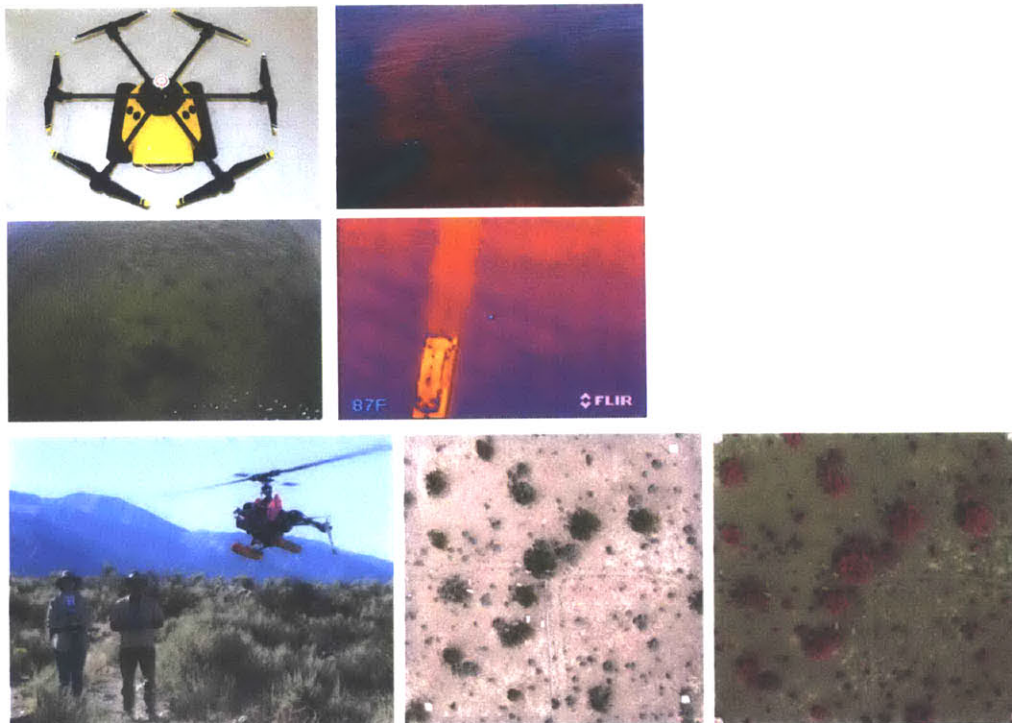


FIGURE 1-9: Standard color digital photo Multispectral CIR photo
TOP: UAV PROJECT MEASURING ALGAE BLOOMS W/COLOR+FLIR CAMERA [LEIGHTON, MIT]
BOTTOM: MULTISPECTRAL SENSING FOR CLIMATE CHANGE [DESERT RESEARCH INSTITUTE, FENSTERMAKER]

Current research also carried out at MIT by Ravela, et al [9] has already looked at the use of single and multiple UAV agents to image environmentally occurring phenomena such as volcanic ash plumes. This work examines real-time coordination of many UAVs to capture a given environmental event. Part of the developed system utilizes sensors and real-time imagery to plan trajectories for coordinating UAV-based sensing.

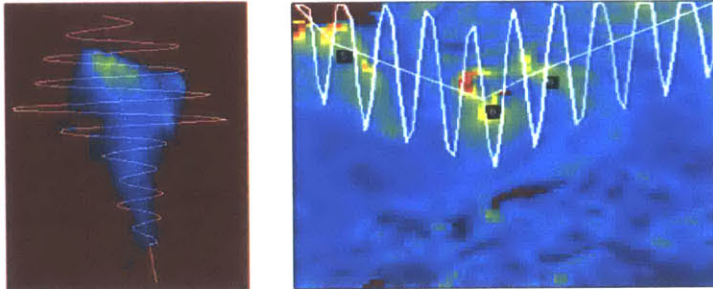
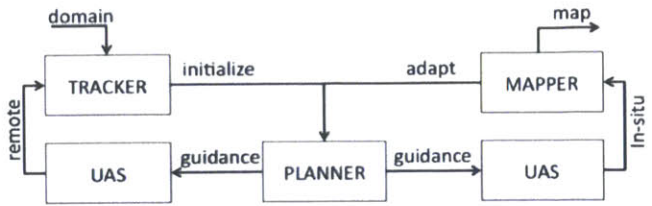


FIGURE 1-10 : FLIGHT PATH GENERATION FOR SENSING A VOLCANIC EVENT, FLIGHT PATH SHOWN IN WHITE [RAVELA]

1.4 UAVs as Vehicles for Collaborative Exploration

The union of semi-autonomous vehicles with advanced wireless sensor networks opens the door to interesting possibilities beyond purely sensing applications or dynamic imaging. The goal of this thesis is to explore and develop a system that merges sensed data with visual imagery, taken in real-time from a UAV agent. Ideally multiple variants will be realized including systems that enable individuals to interact with a single or many UAV agents simultaneously. Similar to video conferencing where many parties are unified despite differences in geographical location, enabling multiple people to see the world through a singular UAV agent, and even further, interact using multiple UAV agents opens the door to true collaborative exploration. In this thesis I present Quadrasense, a project that intersects people and sensors with UAVs and cross-reality[10] for the purposes of enabling new modalities of exploration.

1.5 Video game technology for UAV control

Quadrasense employs the use of a video game engine to unify the domains of sensor data, graphics visualization and UAV control. The concept of using such technology for sensor graphics rendering and visualization is perhaps not unique, but this approach to control a physical UAV agent is novel and previously unexplored. Modern video game engines excel at reconciling all aspects of real-time player positioning for thousands to millions players, simultaneously. Extending this concept to control a physical UAV leverages video game engine advantages while simultaneously forging new ground for real-time UAV applications.

Chapter 2

Quadransense Concept

2.1 Overview

Quadrasense aims to use UAVs and cross-reality concepts to connect people with environments through the conduit of sensors. By creating a context for interpreting the combined real-time stream of video and sensor data, Quadrasense expands and enhances our ability to explore using a semi-autonomous agent. Utilizing video game software to realize key aspects of the system, compelling graphics, animations and rich interactions are enabled without the challenges of creating these elements completely anew. After considering forward-looking aspects of sensor agency and semi-autonomous sensing in the previous sections, key objectives of Quadrasense, outlined in this section, are defined.

A mock-up of the desired primary User Interface is shown in Figure 2-1. This UI provides a snapshot of an “on UAV” perspective (telepresence), with sensor data (shown in white & black circles) alpha blended into the video stream (augmented reality).



FIGURE 2-1: MOCK-UP OF QUADRASENSE UAV VIEW: AUGMENTED REALITY + TELEPRESENCE UI [HADDAD]

High-level Overview of Quadrasense

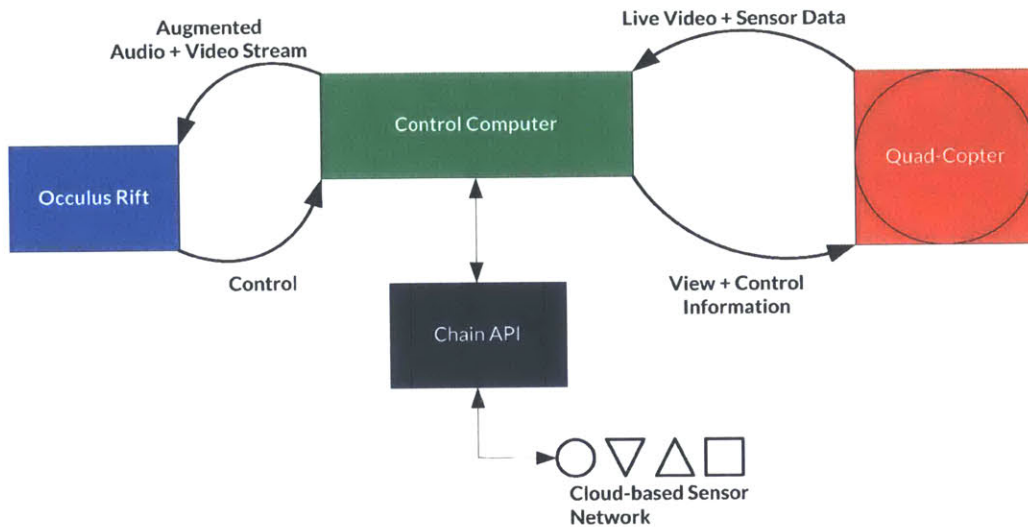


FIGURE 2-2: QUADRASENSE HIGH-LEVEL BLOCK DIAGRAM

A highly simplified proposed system architecture to enable a cross-reality UAV system for terrestrial sensing is shown in Figure 2-2.

1) The proposed system consists of an Oculus Rift Head-Mounted-Display which is the primary user interface. Users will be presented, via the HMD, with three modes of interaction. These three different modes, together, aspire to implement cross-reality concepts:

- a) An “On UAV” mode to experience an environment immersively with augmented reality capabilities
- b) A Birds Eye View for mission planning and selection of relevant sensors for investigation
- c) A virtual “On the Ground” view, which is a purely computer rendering of the environment a user would see if he were physically placed on the ground next to the sensors

2) A commodity computer, to which the HMD is attached, brings in real-time sensor data via a wireless connection using Responsive Environment’s ChainAPI[11] and software real-time blends rendered sensor data with the incoming video. Users head pose and gaze

are transmitted real-time, wirelessly to the quad-copter to enable mechanical or electronic pan-tilt-zoom functionality in the video stream.

3) The desired UAV needs autonomous flight modes (guided by GPS and waypoints), the ability to stay relatively stable in a fixed location (“hover” mode), and the capacity to autonomously land. Guidance and control of the UAV agent is governed by a user’s high-level input (i.e. a gaze or a mouse click on a particular sensor node) which is then translated into flight commands that result in locomotion. To communicate commands and retrieve telemetry information, the UAV needs a flight computer that provides open access to this key functionality.

4) The UAV agent should carry a digital camera imaging system with a very wide field of view (~180 deg horizontal ~180 deg vertical) that either mechanically or digitally tracks the users pose and provides a near normal (50 deg) rectified, flattened, distortion-free field-of-view. To minimize payload for equipment, the same 802.11 stream that carries command/control information is also desirable for carrying the video stream.

5) The software environment (running on the control computer) to coordinate all operations including User Interface, video blending and UAV agency uses the Responsive Environments Tidmarsh project, created in Unity, as a starting point. Unity is a video game design environment that provides rich OpenGL graphics including various shader models, scripted behavior and a physics engine.

6) To enable realistic telepresence and augmented reality experiences, the target end-to-end transmission of video and telemetry data was less than 10 milliseconds (ms). An average of 10 ms latency provides for system update rates of 100 Hz which is a reasonable trade-off between responsiveness and bandwidth for data transmission.

The Tidmarsh landscape[12] in Plymouth, MA will serve as the testing grounds for Quadrasense. Up until recent years this landscape has been used as a cranberry farm. Currently it is part of a large state sponsored restoration project. The environment has been instrumented with Responsive Environments sensor nodes and in conjunction with traditional field work, the space will be observed as it is restored to a more natural condition.

By leveraging existing wireless infrastructure at Tidmarsh, Quadrasense only needs to support WiFi connectivity and Chain API to query and pull data from sensors. The existing Tidmarsh Unity video-game based visualization provides a “First-Person” interaction with a simulated (virtual) version of the Tidmarsh environment. Figure 2-3 shows a sample of this existing visualization. The real environmental topology including vegetation, elevation & water channels have been very closely modeled in the game world. This allows users the

ability to experience the landscape virtually with reasonable accuracy. The Tidmarsh Unity project communicates with deployed sensor nodes and displays data such as temperature, light levels, humidity and so on as simple text on top of a virtualized sensor node.



FIGURE 2-3: EXISTING TIDMARSH UNITY EXPERIENCE, SENSOR DATA RENDERED INTO VIRTUAL WORLD

Chapter 3

Motivation

3.1 Limits of Existing Sensor Visualizations

An examination of traditional sensor visualizations and their applications reveals how they can benefit from the immersive capability that Quadrasense attempts to provide. Many of the existing sensor visualizations are based around either classical statistical analysis of data as a function of time or an overlay of sensor data on static terrain maps. These visualizations are very powerful and extremely useful for detecting trends, characterizing behavior and helping to model environments.

The sensor data shown in Figure 3-1 is measured by Responsive Environments environmental sensor nodes, with visualizations driven by a web browser. This data depicts temperature in the sensed environment and is displayed in a map view giving users an instant understanding of temperature distribution over the measured area for a snapshot in time.

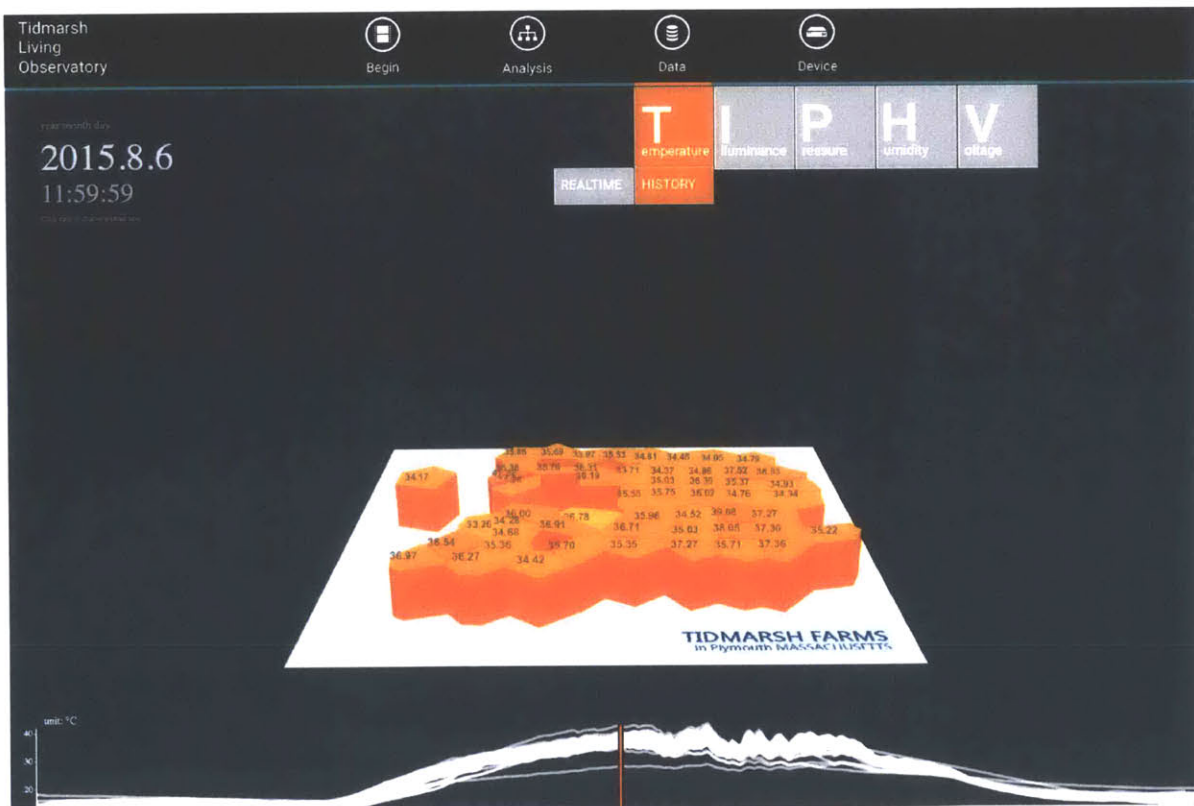


FIGURE 3-1: TEMPERATURE DATA OVERLAID ON A 3D MAP OF A CRANBERRY FARM INSTRUMENTED WITH ENVIRONMENTAL SENSORS [TIDMARSH]

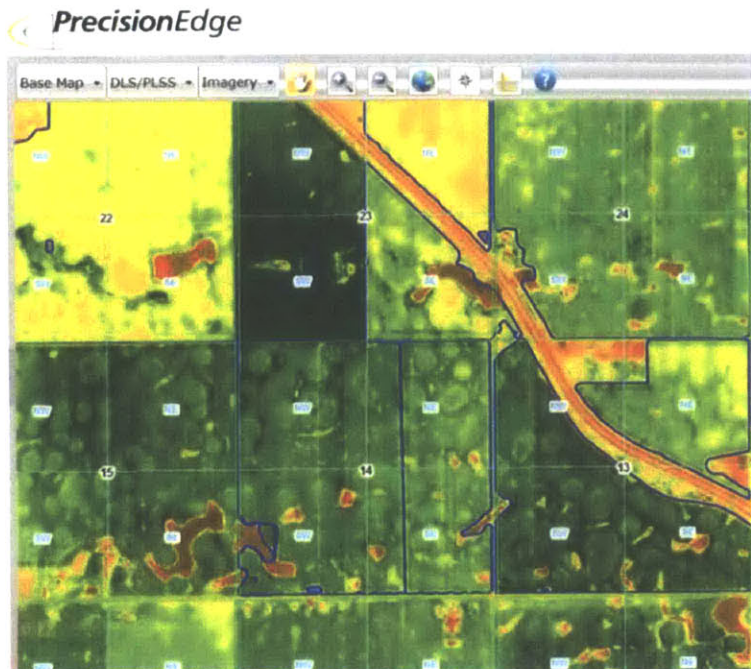


FIGURE 3-2: WATER DISTRIBUTION OVER A 2D MAP FOR AGRICULTURAL USE ; SENSOR DATA ACQUIRED VIA UAV [PRECISION EDGE]

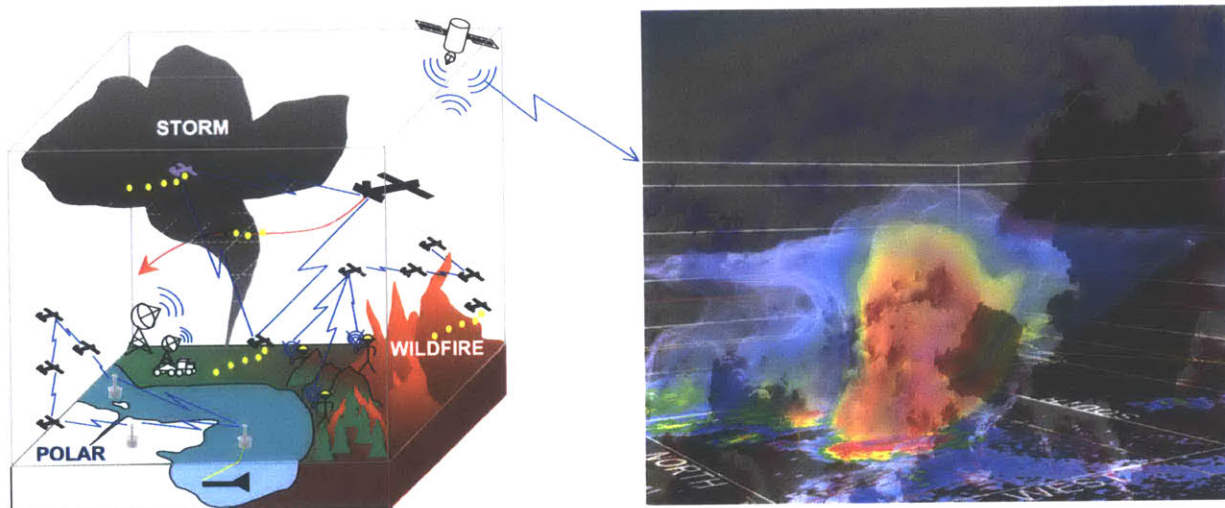
Another sample visualization in Figure 3-2 shows water sensor data for a farm, overlaid with a 2-D map of the sampled space. The water distribution data was acquired using a UAV, flying a prescribed flight plan. The different colors and their intensities specify water concentration, or lack thereof.

The previous Tidmarsh example and this agricultural visualization are highly suitable for offline use, as the time scales for data collection and expected analysis do not depend on real-time use. They are clear and straightforward representations of slowly varying data.

3.2 Applications of Real-time Sensor Visualizations using UAV agents

Some of the existing UAV and non-UAV based sensor visualizations demonstrated are very similar to the sensing flowcharts introduced in Figure 1-4. These applications are non-real-time and follow the scientific research model of capture, process and re-iterate. In contrast, applications that need real-time fusion between sensors and sensed landscapes require a new approach. A UAV possessing augmented reality and telepresence capabilities has the potential to serve this function.

It is beneficial to examine a few use cases that might directly benefit from UAVs possessing augmented reality and telepresence capability used in-conjunction with sensing applications. One set of target applications is proposed by CCMSS, Center of Cooperative Mobile Sensing Systems, a joint research effort between University of Colorado, University of Alaska, MIT and University of Oklahoma [13]. CCMSS aims to use sensors and UAV agents in three mainly identified use cases: “Wildfire”, to combat wildfires, “Polar” to support data collection in a variety of environments from oceanic to atmospheric and lastly “Storm”, collecting volumetric data in severe storm conditions. The CCMSS proposal in-fact specifically calls out the needs for an interactive, immersive technology environment with real-time display and visualizations coming from UAV sensing agents. The left image in figure 3-3, from the CCMSS proposal, illustrates the various environments suggested for UAV usage and the right image shows a potential real-time visual relayed by the UAV sensing agent network.



**FIGURE 3-3: UAVS IN CCMSS IMMERSIVE SENSING
LEFT: WILDFIRE/POLAR/STORM COMBINED[CCMSS], RIGHT: ARTIST RENDITION OF VISUALIZED DATA FROM UAV SENSING[HADDAD]**

For the Wildfire configuration, the authors are looking to use a network of UAVs to help in fire-fighting operations. The ground conditions around large fires is very dynamic and extremely unpredictable. In this situation bulk collection of data and offline processing is simply too slow for the turbulent environment. Fightfighters could instead make use of terrestrial sensors along with real-time imagery, guided by UAV agents, to help track and ultimately curtail the destruction. Quick, responsive, and effective collaboration among remote and ground personnel through enhanced hybrid visuals is clearly a benefit in this application

Real-time visualization of high energy transient weather patterns is another use case outlined by CCMSS. The "Storm" configuration proposes the use of sacrificial UAVs as direct sensing agents to help identify conditions leading to tornado formation. While the phenomenon is grossly predicted by tracking cold and humid air mixing, developing an understanding at a microlevel could help to detect and possibly prevent such events from occurring. As referenced in the CCMSS publication, prior efforts for storm visualization have utilized balloons, augmented with sensors, to limited success. The ability to measure and visualize various parameters within clouds in real-time, even under non-threatening conditions, is another area of interest[14]. Such imagery could return particulate matter and pollutions levels as a result of human activity.

Quadrasure is not intended to replace classical statistical analysis or rigor. Rather, Quadrasure gives users an easy to use interface, providing an intuitive understanding of data and environments. Other off-line techniques such as use of Machine Learning algorithms, Hidden-Markov-Modelling and other heuristic behaviors can reveal latent information in the plethora of data a sensor network collects, guiding future research. However, as demonstrated, there are a growing number of applications that can benefit from the new sensor interactions Quadrasure aims to enable.

Chapter 4

Related Work

4.1 Overview

In addition to exploring a number of the sensing projects outlined in the Introduction and Motivation sections, when contemplating the design of a telepresence, multi-user UAV system, a number of related research endeavors were examined to narrow the design space.

4.2 Telepresence / Augmented Reality / CAVE

The goal of realizing end-to-end 3D telepresence, i.e. blending augmented reality (AR) and virtual reality (VR) is perhaps mostly readily identified with research at University of North Carolina, Chapel Hill. The research group at UNC, headed by Henry Fuchs, has been investigating the concept of immersive technologies, 3D video conferencing and low-latency collaboration systems for almost two decades.

In “Enhanced Personal Autostereoscopic Telepresence System using Commodity Depth Cameras”[15], the authors’ work combines multiple remote users in a single 3D environment possessing the ability to interact with virtual objects. Maimone suggests for collaboration that motion tracking in mixed-domains (where virtual objects and real-world objects are synthetically blended) provides users with a tremendously increased sense of collaboration. The research emphasizes that the depth cues afforded by free head movement enhances the user experience, corroborating the design choice of a Head-Mounted-Display for Quadrasense.



FIGURE 4-1: TWO TELEPRESENCE USERS IN A SINGLE 3D ENVIRONMENT [2011 PAPER]

Prior work by Fuchs et al [16] combined head-mounted displays and augmented reality concepts to help with surgical biopsies. This project mixed imagery between off-line ultrasound and real-time video. Many of the AR lessons outlined in the paper on topics such as graphics re-projection, culling and camera/optics alignment suggest necessary approaches for convincing augmentation.

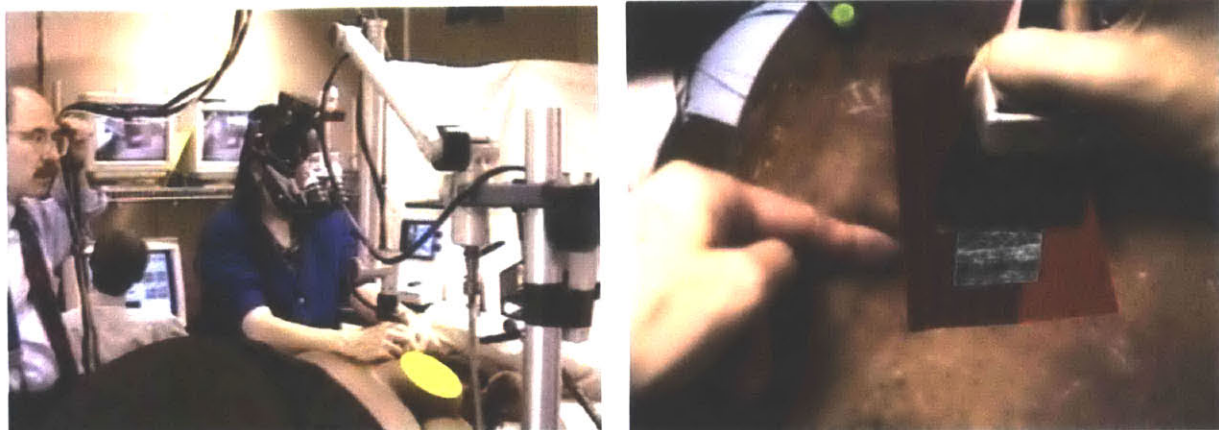


FIGURE 4-2: EARLY HMD+Augmented Reality Surgical Application [FUCHS]

A recent publication from Fuch’s group, “Minimizing Latency for Augmented Reality Displays”[17] demonstrates a custom augmented reality display using TI Digital Mirror Displays to project synthesized graphics on real-world objects. This implementation is highly novel, in that only part of the display is updated during the refresh cycle. By employing concepts of estimation between a perceived image and actual output, the system sacrifices some absolute image quality to minimize delays. The authors of the paper strive for a total system delay of 2ms. The concepts of estimation as it applies to head tracking and system rendering are highly relevant to Quadrasense since similar problems are faced in

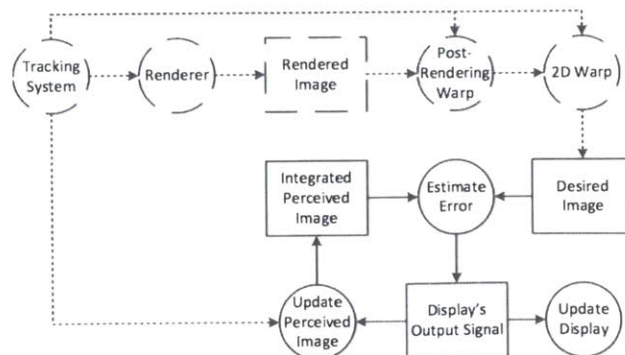
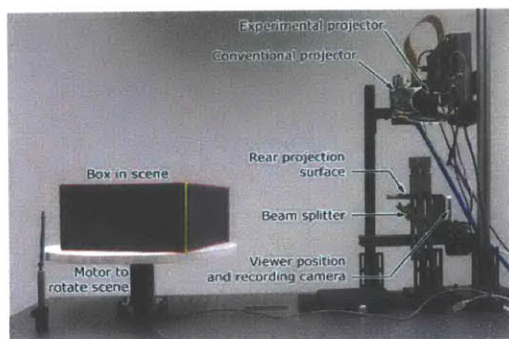


FIGURE 4-3: LOW-LATENCY AUGMENTED REALITY DISPLAY USING PERCEPTIVE RENDERING [FUCHS]

the reception of telemetry data via a low-bandwidth, high-latency lossy link. In augmented reality use cases, the latency of the telemetry link will manifest itself as misalignment between the real-time imagery and the rendered sensor data.

4.3 Omnidirectional Imaging

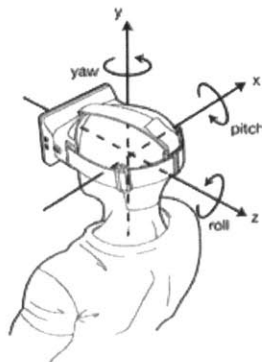


FIGURE 4-4: HMD MOVEMENTS SHOULD TRANSLATE TO CORRESPONDING CAMERA VIEW [OCULUS]

A clear goal of this project was to enable users to experience the actual environment the UAV is placed in. The intention is, by providing a video stream which allows users to see in any forward direction, the experience mimics physically being on the UAV. To enable this, the camera perspective should track in real-time the user’s HMD pose. Low-latency and omni-directionality are more highly desired than absolute resolution, so a camera that is either physically gimbaled or has super-wide angle of view was an objective. The ability to image in 3D, while not a strict requirement, was also desired, so all hardware selection was done with this long-term goal in mind.

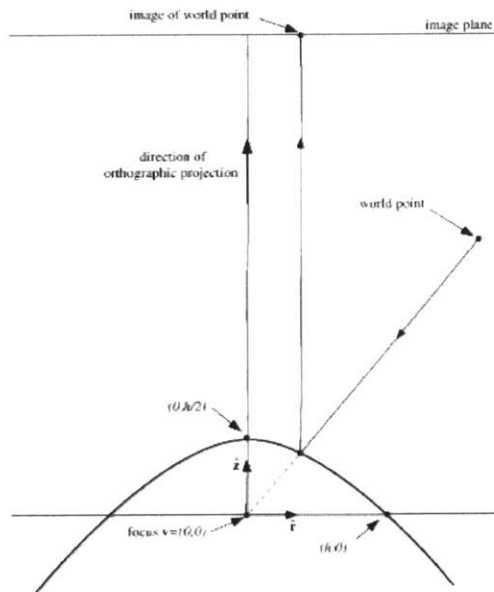


FIGURE 4-5: OPTIMAL PARABOLOID FOR CATADIOPTRIC-BASED 360 DEG IMAGE CAPTURE [BAKER, NAYAR]

Early research into omnidirectional imaging systems by Baker & S.K. Nayar and associated researchers at Columbia proved highly relevant for single-capture, omni-directional imaging. This work carried out in the late 90’s and early 2000’s used a catadioptric configuration of hemispherical mirrors and conventional CMOS/CCD imagers to capture images and video in 360 deg[18]. The image in Figure 4-5 shows an optimized configuration of a catadioptric reflector & sensor assembly for this application. Such a system has numerous advantages over a traditional mechanically gimbaled system. Work by Nayar and et. all was extended beyond stills to include concepts of immersive video. Demonstrated work used an early Head-Mounted-Display in-conjunction with a 360 deg catadioptric system to allow real-time seamless panning of captured video. This work was later expanded to include

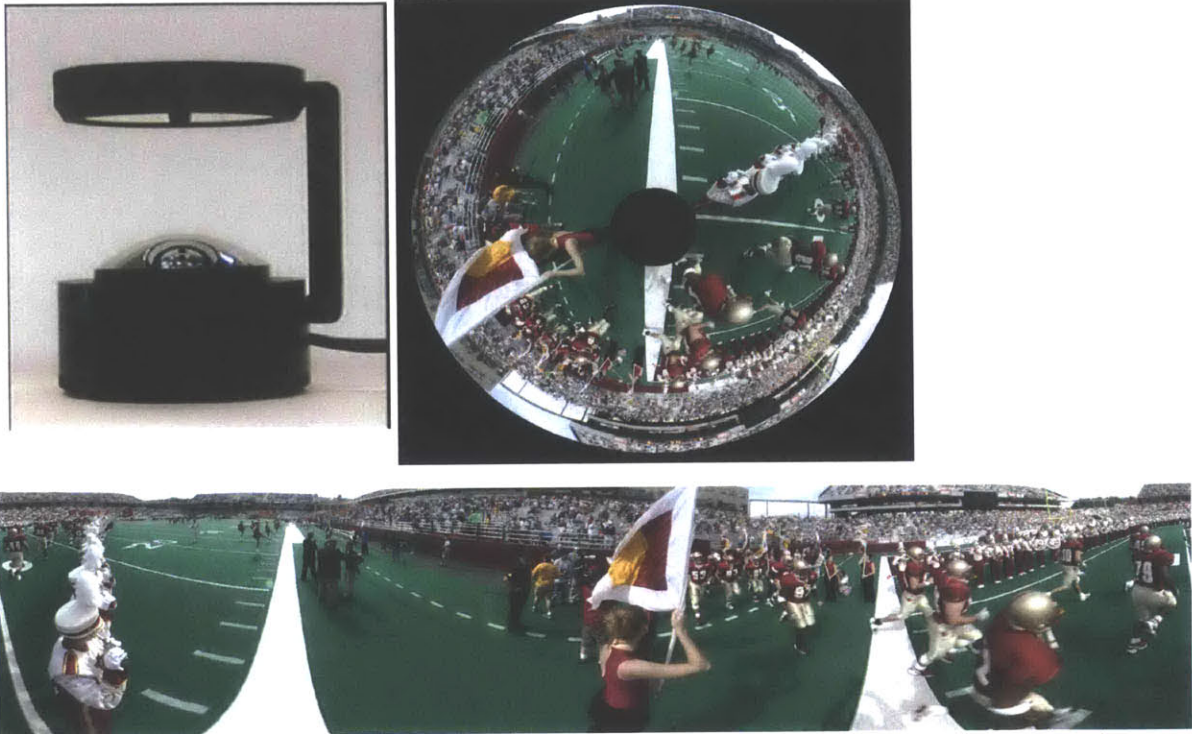


FIGURE 4-6: EXPERIMENTAL 360-DEG CATADIOPTRIC CAMERA, RAW IMAGE CAPTURE (TOP-RIGHT) AND UNWRAPPED IMAGE (BOTTOM) [2001, NAYAR]

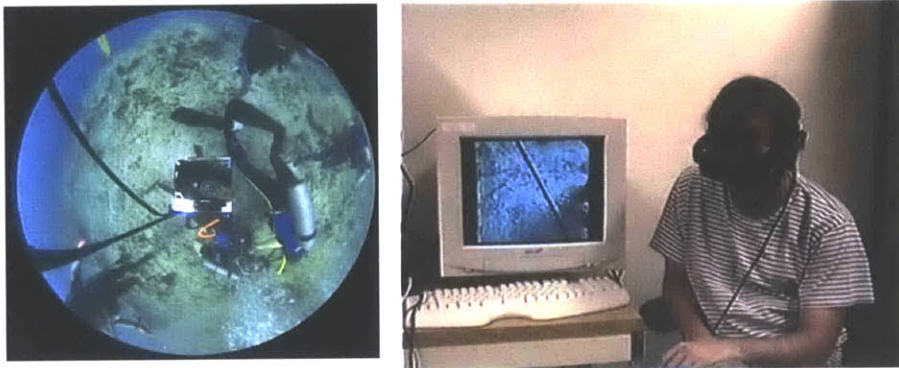


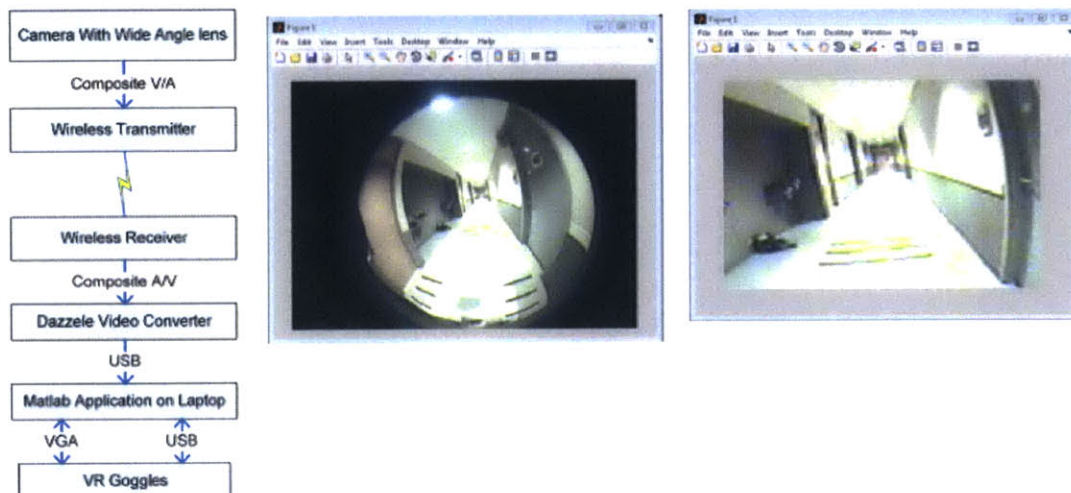
FIGURE 4-8: 360-DEG CATADIOPTRIC IMAGE CAPTURE AND REAL-TIME UNWRAPPED VIEWING USING HMD [2001, NAYAR]

360 imaging on remote control vehicles, such as remote controlled cars.

Similar work into the area of fisheye imaging, another mechanism for 360 deg simultaneous capture was also explored. Fisheye imaging, due to smaller implementation size and lack of mechanical obstructions, has become popular in robotic applications. Work in this domain includes calibration and algorithms for flat-field dewarp image reconstruction.

4.4 Omnidirectional Imaging with UAVs

In the 2011 paper “An Image-Processing-Based Gimbal System Using Fisheye Video“[19], the authors detail a fisheye imaging system for omnidirectional imaging using a Head-Mounted-Display. Their goal was to enable the functionality of pan-tilt-zoom without moving parts, targeting UAV applications. The system captures and transmits the entire fisheye image, and performs the image orientation/rectification from head-tracked data on a receive-side computer. While this demonstrated system is novel, there are a number of drawbacks. Their use of analog imaging and transmission required digitization on the receive side prior to dewarp, resulting in latency. Additionally their analog capture was limited to only NTSC/PAL resolution (approximately 480i). As noted by the authors, in recording the full fisheye image, de-warped imaging resolution was sacrificed, resulting in a rectified image only 180px x 144px in size .



**FIGURE 4-9: FISHEYE IMMERSIVE VIDEO, NO MOVING PARTS [2011, RAWASHDEH & SABABHA]
 TOP LEFT: EXPERIMENTAL SYSTEM, TOP RIGHT, USER USING HMD FOR NAVIGATION
 BOTTOM LEFT SYSTEM FLOW, BOTTOM MIDDLE WARPED IMAGE, BOTTOM RIGHT DE-WARPED IMAGE**

Work carried out in 2014 “Oculus FPV” [20] approaches the same problem of immersive

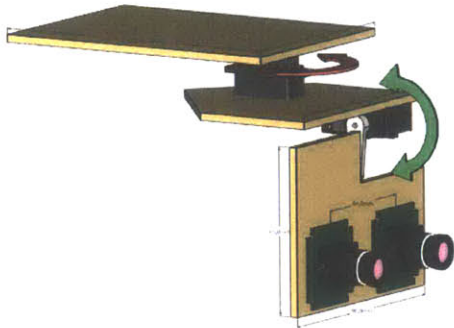


FIGURE 4-10: LOW-COST GIMBAL FOR STEREO IMAGE CAPTURE, TARGETING UAV +HMD [HALS ET. ALL]

video for UAVs by utilizing a pair of analog video cameras mounted on an inexpensive custom-made gimbaled system along with the Oculus Rift Head-Mounted-Display and a downstream computer for post-processing. Their low-cost gimbal utilizes two servo motors enabling pan (horizontal rotation) and tilt (vertical translation) in conjunction with a micro controller and wireless link which receives in real-time, user’s head pose and gaze information. Similar to the prior approach, to present the images to the user, the analog video is digitized by a commodity PC using twin video capture cards.

Due to the distortion of the Oculus Rift’s optical objectives, the authors “pre-distorted” the captured video so when viewed through the Rift, the images appear approximately flat-field. While the approach presented is low-cost, there are a few drawbacks. Without a carefully architected system that synchronizes analog video capture/transmission (genlocking) and the digitization process, as the authors note, slight miss-matches in timing occur when presenting the video stream to the end-user. As a result of poor synchronization, the authors observed eye strain, fatigue and even motion sickness.

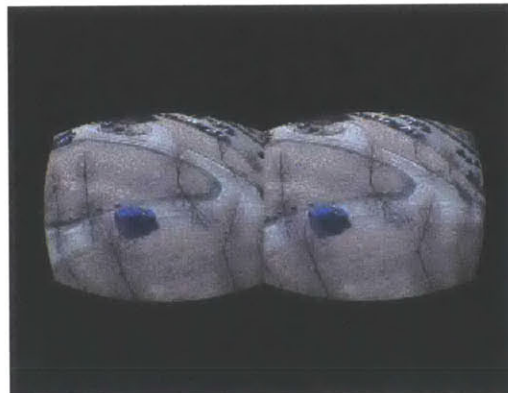


FIGURE 4-11: REALIZED OCULUS FPV, WITH CAPTURED VIDEO [HALS ET. ALL]

In comparing the two image capture systems, fisheye/catadioptric vs. “normal” field of view image capture, if omnidirectional imaging is possible, this approach provides a clear advantage in that users may examine and experience different viewpoints from recorded video via the HMD, post-image capture. With normal imaging, the real-time perspective is the only perspective ever recorded. Thus, for flying and safety applications normal imaging

is reasonable, for immersive and exploratory uses, with sufficient imaging resolution and transmission bandwidth, omnidirectional capture is a worthy technique.

4.5 Multi-rotor UAVs and mixed-domain sensing

Prior sections have outlined projects that either proposed the use of UAVs for immersive use cases, or UAVs acting under control of a sensor network. In “Environmental Sensing Using Land-Based Spectrally-Selective Cameras and a Quadcopter”[21] the authors have used a sparse array of ground sensors to measure temperature, with the UAV acting as an agent to increase sensing density by physically sampling in-between sensor positions. In this work the authors have used a FLIR (Forward Looking InfraRed) camera as part of sparse set of sensors to fuse with data acquired by a UAV+on-board temperature sensor. This project is very illuminating as a potential use case for Quadrasense. Temperature measurement data acquired by the UAV is a real-time process, as-is the capture by the thermal camera. However the outlined data fusion is currently non-real time. Visualizing the status of sensor interpolation, as the measurements are taken, can certainly benefit numerous remote sensing applications.

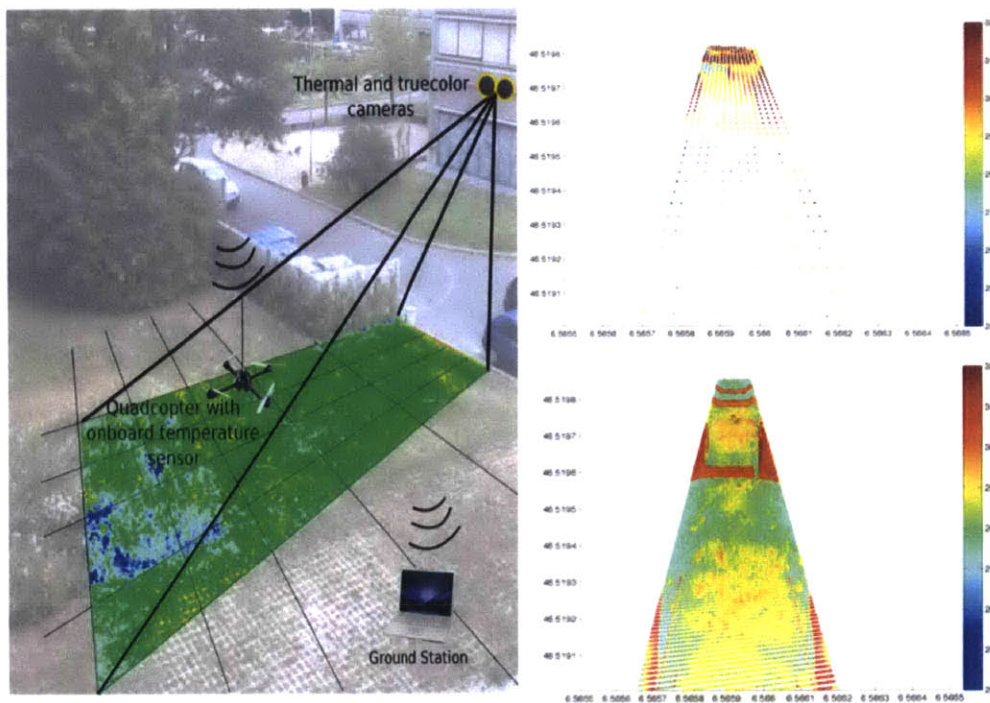


FIGURE 4-12: USING A UAV TO INTERPOLATE COARSELY SAMPLED THERMAL DATA [DAS, ET. ALL 2013]

**LEFT: EXPERIMENTAL SETUP,
RIGHT-TOP : SPARSE DATA FROM THERMAL CAMERA
RIGHT-BOTTOM : INTERPOLATED DATA AFTER SENSING WITH UAV**

4.6 Video game engine use for immersive visualization

Published work carried out by Paul Bourke at University of Western Australia in 2009 details iDome[22], an immersive experience concept using Unity for visualization generation. iDome utilizes a large 3m hemispherical “hull” as a surface for projection in conjunction with a commodity 1080p projector and spherical mirror. iDome’s aim is to visually surround a user and present a view that is all encompassing, similar to an IMAX theater experience ; by using a spherical mirror surface and curved dome, the end user does not see any projection apparatus and the system engages peripheral vision cues.

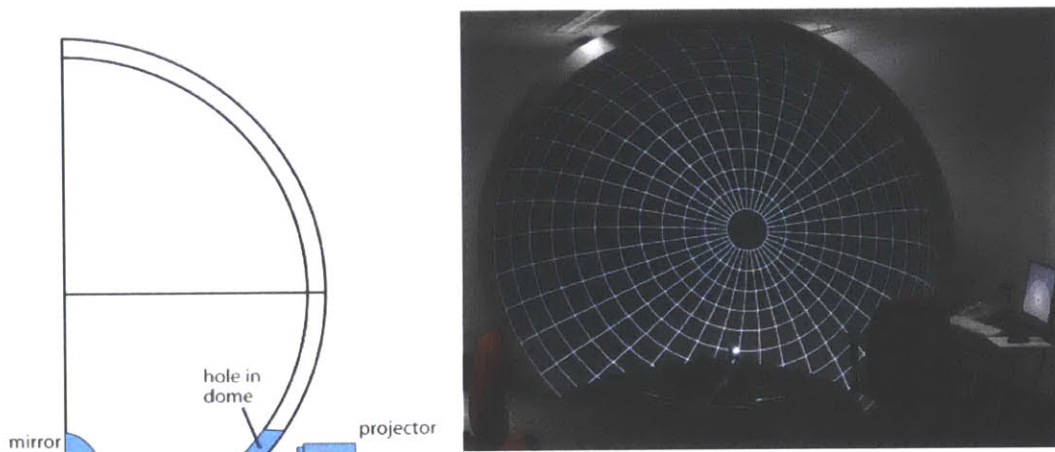


FIGURE 4-13: IDOME AN IMMERSIVE ENVIRONMENT THAT ENGAGES PERIPHERAL VISION [BOURKE]

Unique and relevant aspects of iDome are the use of Unity gaming environment to prototype and explore the graphics concepts of compositing multiple “normal” 50/60 deg rectilinear views into a curved image suitable for immersive viewing. While Qudrasense explores the dual, fisheye and catadioptric imaging with rectified views for HMDs, iDome’s approach has significant merit as catadioptric and fisheye image capture systems already provide a warped view. Additionally work outlined in iDome discusses the use of vertex shaders on commodity graphics hardware to perform all computations with low-latency, and at video frame rates (60 fps+) all from with-in Unity. Use of Unity for image processing takes advantage of a Graphics-Processing-Unit (GPU), in contrast to previous approaches in this section which rely on general purpose CPU cycles to handle all image processing, including rectification.

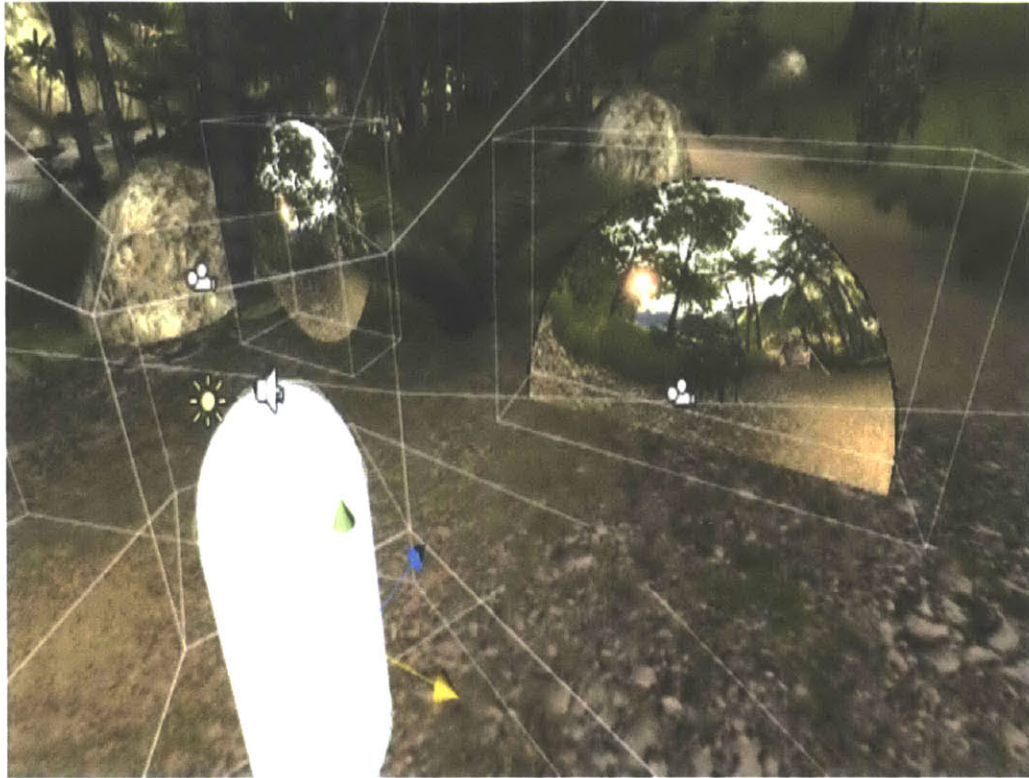


FIGURE 4-14: REAL-TIME IDOME VIDEO GENERATION USING UNITY [BOURKE]



FIGURE 4-15: UNITY-BASED RENDERED GAME CONTENT IN IDOME [BOURKE]

Chapter 5

Quadrasense High-Level Design

5.1 Overview

The following sections will briefly describe the design choices, the next chapter covers detailed implementation. Figure 5-1 highlights keys components of the realized system.

QuadraSense System Level View

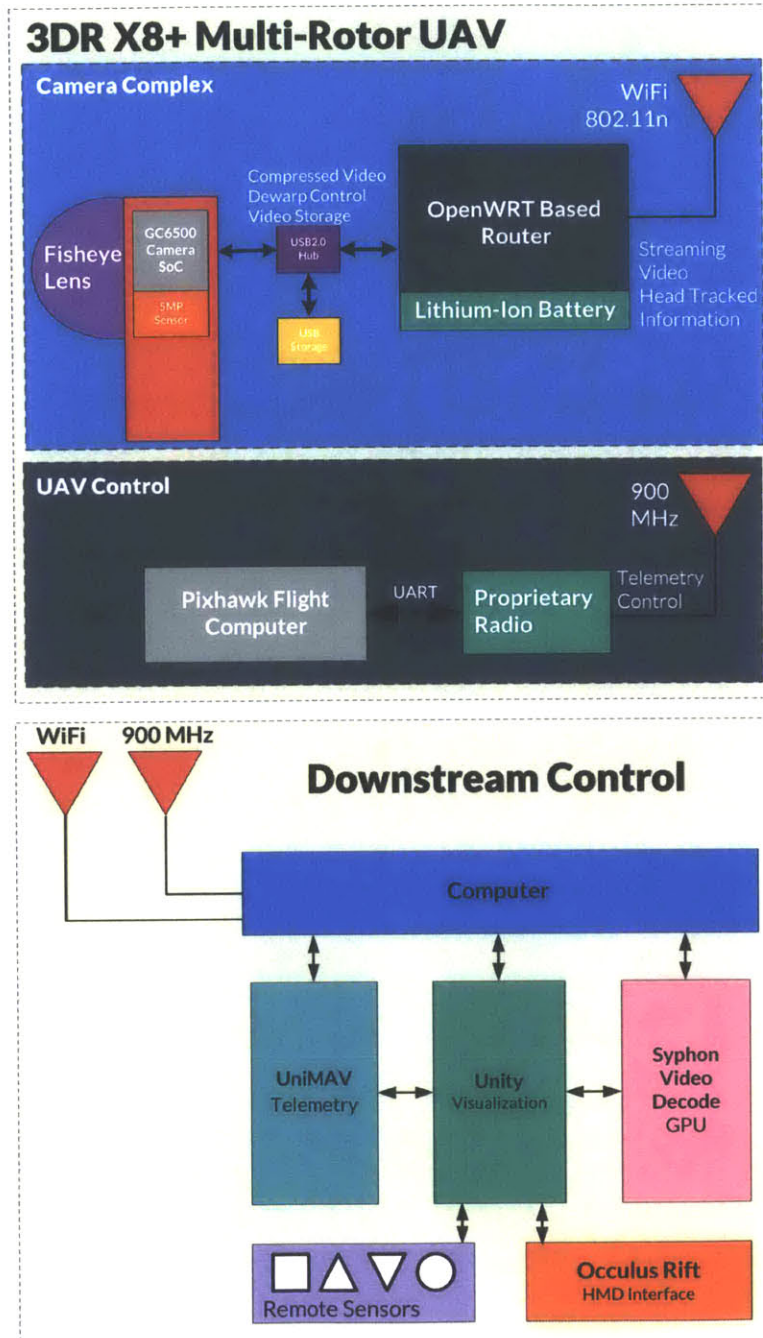


FIGURE 5-1: QUADRASENSE SYSTEM LEVEL BLOCK DIAGRAM

5.2 The UAV



FIGURE 5-2: EIGHT ROTOR X8+ UAV SELECTED FOR USE IN QUADRASENSE [3DR]

A 3D Robotics X8+ was selected as the UAV agent for several key reasons:

- 1) The multi-rotor configuration was selected over other vehicles such as planes and gliders due to the extreme maneuverability and payload capacity. The eight rotors on the X8 offer additional payload carrying capacity compared to a traditional quad-rotor. Excess capability was selected with the forethought of carrying and deploying sensor packages in addition to any imaging hardware.



**FIGURE 5-3:
TOP PIXHAWK
BOTTOM TELEMETRY
RADIOS [3DR]**

- 2) The 3DR UAVs make use of an open-source flight computer system called the Pixhawk. This control computer implements a well documented open-source control protocol called “MAVlink”, an underlying messaging system for sending telemetry and receiving control information. A key motivator in choosing the 3DR platform and the Pixhawk was clear, and documented access to the telemetry link.
- 3) The 3DR X8+ utilizes a discrete wireless telemetry/control link module. Two identical units are used, one is placed on the UAV and interfaces to the Pixhawk via an RS-232/UART and the second radio links to a downstream computer or tablet via standard UART-over-USB connection. Although there are numerous advantages to merging telemetry control with a video-link interface, such as the potential for higher update rates for

telemetry, having a separate control radio simplified the software development process, as no demultiplexing or additional complex protocols needed to be handled. The control plane and data plane (video) are cleanly separated.

5.3 Camera Complex

The camera complex in Quadrasense (figure 5-1) refers to the on-UAV components needed to realize wireless transmission of omnidirectional video. The complex is composed of the GC6500 camera sub-system and the OpenWRT wireless router.

Camera sub-system

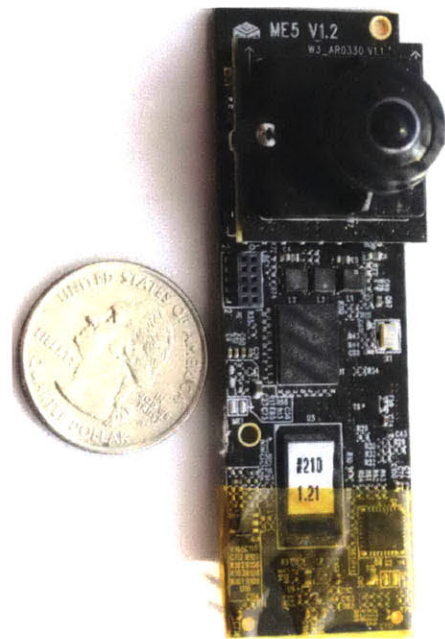


FIGURE 5-4: FISHEYE CAMERA USED IN QUADRASENSE PROJECT

After careful consideration of many potential solutions, including devices such as a GoPro and custom gimbaled cameras, an evaluation platform from GeoSemiconductor utilizing the GC6500 SoC was selected. This platform uses a 5 MegaPixel sensor in conjunction with a near 180-deg fisheye lens to enable real-time wide-angle capture. The GC6500 SoC platform implements several key features that were strong motivators for its use:

- 1) The SoC, in hardware and in real-time, can dewarp the fisheye (to provide an approximately normal field of view) while compressing the image. The dewarp engine is under user control so a new perspective can be communicated to the SoC which in turn provides an updated, rectified image stream. Virtual parameters of the GC6500 such as horizontal pan, vertical pan and tilt can be mapped to the yaw/roll/tilt angles from a traditional Head-Mounted-Display, easing implementation. The use of fisheye imaging and dewarp was highly appealing owing to (in theory) only a single frame of delay for all panning and movement operations.
- 2) The compression engine in the SoC is capable of an all "Intra coded" mode where each compressed image does not depend on a previous or next image (motion

estimation / motion prediction), as is normally done with video CODECs. Maximum latency for compression, using H.264 in this mode, is sub-10ms .

- 3) The GC6500 interfaces to a host processor via single USB2.0 High-Speed link and is bus powered, drawing less than 500 mA. Since the target platform is moving, a battery-based power delivery system was required and a low-power platform eased the burden of additional batteries

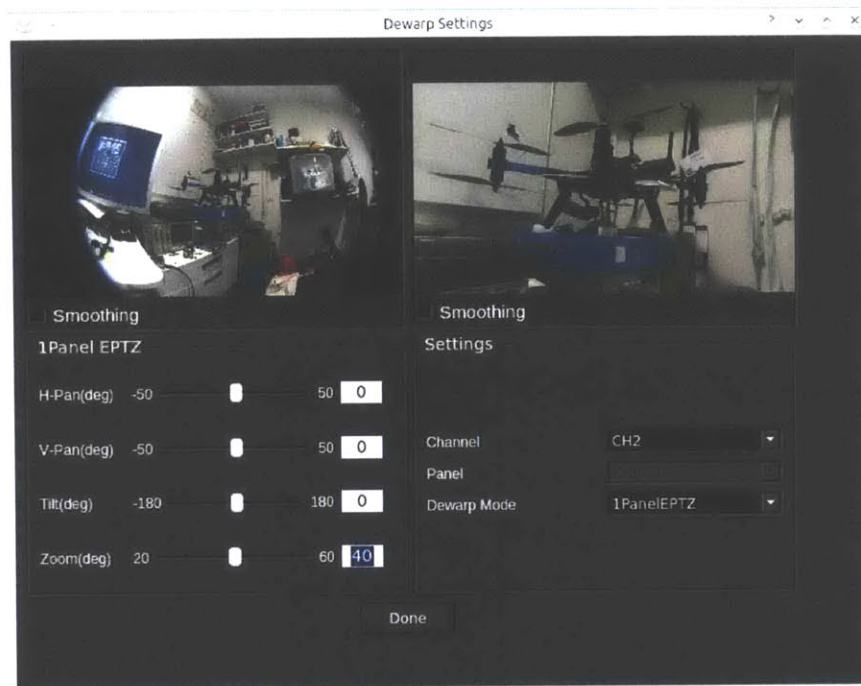


FIGURE 5-5: TOP DEWARP CONTROL IN GC6500, BOTTOM FISHEYE & DEWARPED IMAGE

Wireless video link and camera control



FIGURE 5-6: TPLINK ROUTER SELECTED FOR QUADRASENSE, USB2.0 HS + LITHIUM-ION BATTERY [TPLINK]

An off-the-shelf TPLink router was selected to interface to the GeoSemi camera system. This commercial router has several benefits including a built-in battery, a USB2.0 high-speed interface and 802.11n 108 Mbps WiFi. OpenWRT, a Linux-based wireless router distribution, is supported on this device and the complete OpenWRT Linux-based toolchain makes cross-compilation fairly straightforward.

In addition to streaming video, by adding an external USB 2.0 High-Speed hub and USB2.0 flash drive, the platform was extended to include local recording of the fisheye video (which is not sent wirelessly due to bandwidth limitations).

5.4 Syphon GPU texture sharing

The Syphon software framework[23] allows different programs running on the same Apple OSX machine to share GPU textures with next to zero overhead. Owing to its cross-platform capability, Unity supports a very limited number of native video CODECs and relies on the main CPU for all decoding. By decoding the incoming video in a separate application and making the decoded frames available to Unity via a Syphon GPU texture, video streams using arbitrary compression formats can be piped into the game environment.

Syphon video frames in Unity appear simply as a graphics texture, meaning all hardware accelerated data paths of the GPU including tessellation, shaders and so on supported by Unity and can be applied in real-time to the video. Figure 5-7 shows an example of live video, sent via Syphon texture to Unity, with translation and rotation applied in the Unity game environment.

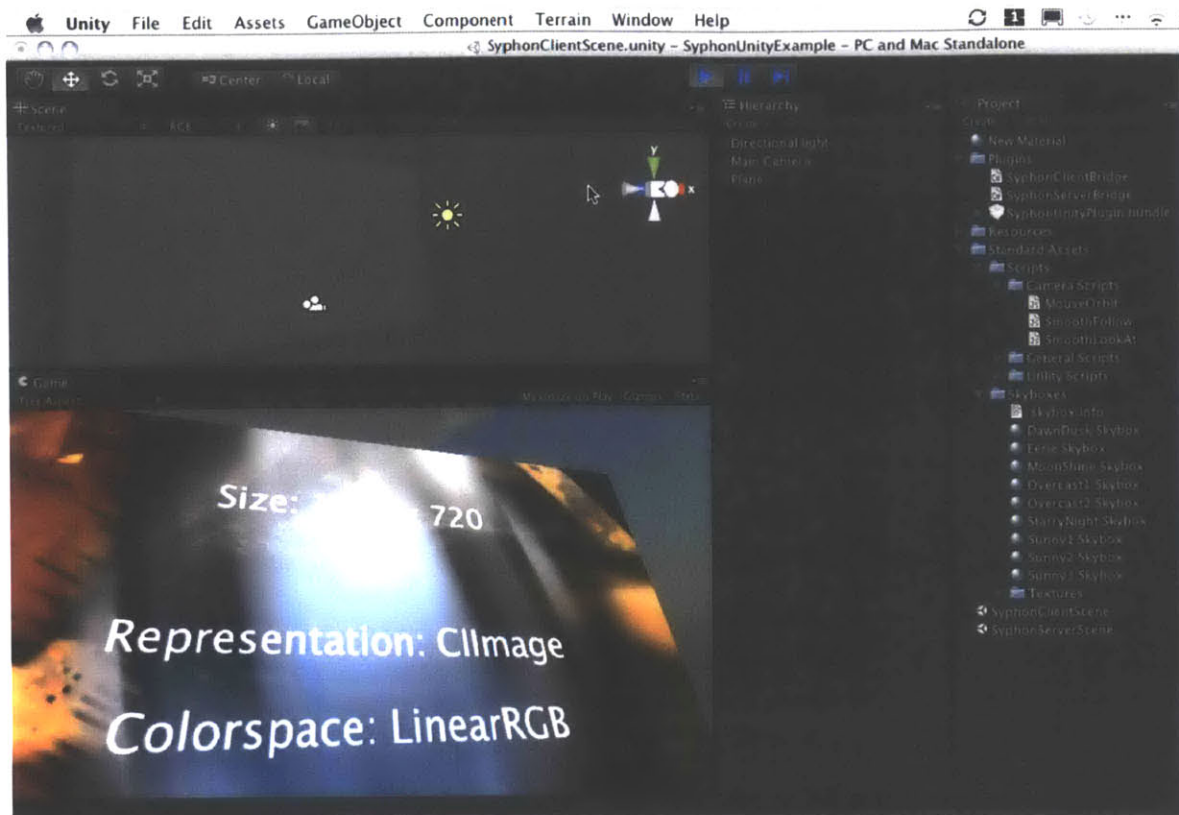


FIGURE 5-7: VIDEO SCALED, TRANSLATED AND COMPOSITED REAL-TIME IN UNITY FROM A SYPHON VIDEO FEED [SYPHON]

5.5 UniMAV: UAV Telemetry and Guidance Control

As mentioned previously, the Pixhawk flight computer uses a protocol called MAVLink for guidance commands, relying telemetry, battery status, and system status. To enable Unity the ability to access telemetry data and to relay guidance waypoints back to the UAV, a custom module named “UniMAV” was written for the open-source Python-based application called MAVProxy.

MAVProxy is a text-mode interface that translates high-level human readable commands to the lower level MAVLink messages which are ultimately sent via the wireless telemetry link to the Pixhawk. The processed MAVLink messages result in the UAV taking action. For example a MAVProxy human-readable request of “takeoff 10m” is translated into the necessary MAVLink messages that command the UAV to takeoff to an altitude of 10 meters.

5.6 HMD and Unity environment

MAV_CMD

Commands to be executed by the MAV. They can be executed on user request, or as part of a mission script. If the action is used in a mission, the parameter mapping to the waypoint/mission message is as follows: Param 1, Param 2, Param 3, Param 4, X: Param 5, Y:Param 6, Z:Param 7. This command list is similar what ARINC 424 is for commercial aircraft. A data format how to interpret waypoint/mission data.

CMD ID	Field Name	Description
16	MAV_CMD_NAV_WAYPOINT	Navigate to MISSION.
	Mission Param #1	Hold time in decimal seconds. (ignored by fixed wing, time to stay at MISSION for rotary wing)
	Mission Param #2	Acceptance radius in meters (if the sphere with this radius is hit, the MISSION counts as reached)
	Mission Param #3	0 to pass through the WP, if > 0 radius in meters to pass by WP. Positive value for clockwise orbit, negative value for counter-clockwise orbit. Allows trajectory control.
	Mission Param #4	Desired yaw angle at MISSION (rotary wing)
	Mission Param #5	Latitude
	Mission Param #6	Longitude
	Mission Param #7	Altitude
17	MAV_CMD_NAV_LOITER_UNLIM	Loiter around this MISSION an unlimited amount of time

FIGURE 5-8: SAMPLE MAVLINK COMMAND MESSAGE [MAVLINK WEBSITE]

As mentioned, the Responsive Environment's Tidmarsh Unity project was used as a starting point for the UI visualization and control aspects of Quadrasense. The existing Unity Tidmarsh application is able to communicate with the Responsive Environments Sensor nodes with ChainAPI. An Oculus Rift DK2 device was used as the Head-Mounted-Display. A Unity script was created to relay head position to Camera Complex via a UDP stream. Unity communicates commands to the UAV via UniMAV and also receives telemetry data via UniMAV.



FIGURE 5-9: LEFT: OCULUS RIFT DK2, RIGHT PRE-DISTORTED IMAGES FOR LEFT/RIGHT OBJECTIVES

Chapter 6

Quadrasense Implementation Details

The next sections will cover the hardware, embedded software and Unity software developed in the course of the project with associated details and trade-offs.

6.1 Camera

A small, cross-platform C++ program, running on the router device sent video frames and proxied head-mounted-angles to the GC6500. In the final implementation this program sends video frames, compressed in H.264 over a dedicated UDP port and receives angle position information from the Oculus Rift in a simplified 12-byte format. In addition to transmitting a dewarped video stream, the developed software also saves a local copy of the unwarped video stream to a file whose name is auto generated by the current day and time.

The GC6500 SoC utilizes a USB2.0 host interface and sends compressed video using the Universal-Video-Class of USB. Because the GC6500 can send up to eight compressed multiplexed video streams (e.g. dewarped video and regular video), GeoSemi supplies a custom driver that was cross-compiled for the Linux router platform. All video is transported to the host via a single Linux UVC end-point and the Geosemi driver/libraries provide a C-language API which allows user applications to demultiplex the many video streams completely in software. Users associate their application to a specific video stream by registering a C-callback function. For Quadrasense, two video callbacks were registered, one for the fisheye image stream and a second for the dewarped video. Use of callbacks enables psuedo-threading as the user's callback function is only activated on an event (new frame), but does not use any polling.

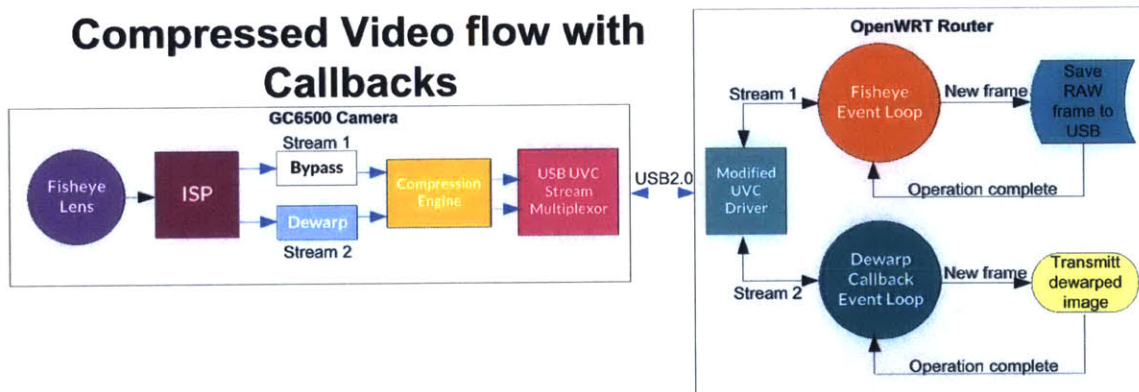


FIGURE 6-1: VIDEO FLOW THROUGH GC6500 + UVC DRIVER

6.1 Low-latency Video

To deliver an immersive experience, with conservative 802.11n bandwidth usage, the H.264 video CODEC was chosen, in an all Intra-coded mode with a Constant-Bandwidth-Rate inside the GC6500 set to 5 Mbps (~600 KBytes/second). To minimize end-to-end latency from capture to delivery, network delays had to be carefully considered. To transport video to the downstream computer a cross-platform approach was also required. TCP was ruled out due to packet acknowledgements adding significant delay ; wireless links are notoriously lossy and if acknowledge packets are lost, then video delivery halts until a backlog of packets are successfully transmitted. To meet the demanding <10 ms latency figures, two different network approaches were implemented and tested:

- 1) UDT[24], a TCP-like protocol that uses UDP for its underlying transport but includes the concept of retries for only lost packets. UDT API and member functions have close correspondence to Berkley sockets and can almost be used interchangeably. UDT is written in C++.
- 2) A pure UDP implementation which does not include any kind of re-tries, where each image frame is enqueued into a large buffer and the appropriate send function is called.

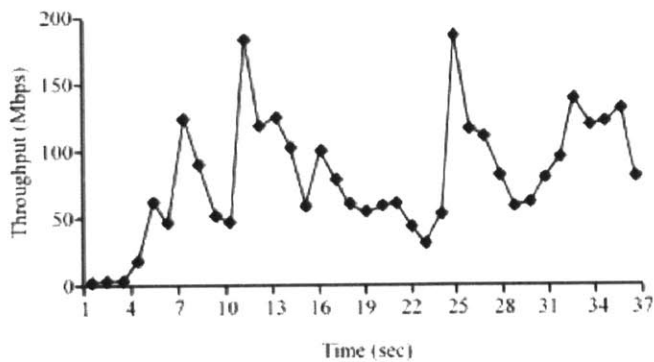
```
1 // Dewarped video callback
2 static void video_cb(unsigned char *buffer, unsigned int size, video_info_t info, void *user_data)
3 {
4     struct channel_info *vch_info = (struct channel_info*) user_data;
5
6     if (use_UDT == 1)
7     {
8         // Send frame out using UDT
9         UDT::sendmsg(UDT_client, buffer, size, -1, false);
10    }
11    else
12    {
13        // Send out the video frame via UDP
14        sendto(sndsock_fd, buffer, size, 0, (struct sockaddr *) &sndsock, socketlength);
15    }
16
17    printf("Sent frame: %d \n\r", frame_cnt);
18    frame_cnt++;
19
20    // Update dewarp engine on frame sync
21    mxuvc_video_set_dewarp_params(ch, dw_panel, dw_mode, &dw_params);
22
23    // Done with callback
24    mxuvc_video_cb_buf_done(vch_info->ch, info.buf_index);
25 }
```

FIGURE 6-2: UDT AND UDP CODE FOR SENDING DEWARPED VIDEO PACKETS

In UDT, packets sent and received are accounted for in a “bitmap” and the receiver compares bitmaps with the sender so that only lost packets are re-sent if the network experiences packet loss. Actual end-to-end testing revealed that under low-loss network conditions UDT performed quite well, however under heavy packet loss, UDT would take an unpredictable amount of time before new video frames would appear. Because UDT is

designed for reliable file transfer over UDP, the concept of “semi-reliable” transport is difficult to achieve.

With a wireless link that both experiences packet loss and possess unused capacity, a reasonable system-level implementation will utilize excess bandwidth to resend lost video frames with some expiry time before retries are stopped (as to not present user with stale or outdated video frames). While in principal UDT's retry/acknowledge scheme provides for this highly desirable functionality, in practice it did not deliver satisfactory results. UDT estimates the channel latency and bandwidth to most effectively transport data without causing excessive packet-loss and uses a software control-loop algorithm to adjust interpacket-delays, window size (packets per frame) and retries. Unfortunately wireless packet loss is often random and difficult to predict i.e. interference caused by microwave oven usage or a car passing by. Because UDT attempts to adapt to the dynamic network conditions, testing with random interference resulted in an oscillatory behavior in UDT's control-loop. The control-loop oscillations ultimately resulted in oscillatory bandwidth utilization, causing extraneous delays, loss of video and poor recovery . UDT's instability under random packet loss has been noted in other literature[25], shown in Figure 6-3.



UDT throughput changes with time

FIGURE 6-3: UDT BANDWIDTH OSCILLATIONS [REN, TANG, QIAN]

The UDP implementation of wireless video delivery was quite straightforward to implement and test. Each received video frame is enqueued into a recipient UDP buffer in the callback and the appropriate socket API to send is called to send the data. The TCP/IP stack automatically fragments the large packet into smaller frames (up to the MTU of 1500-bytes). This particular implementation of video-over-UDP does not implement any kind of acknowledge, re-try, or re-transmission. With the assumption of a 30 fps video stream and a new frame arriving every 33 ms, any partial loss of a fragmented UDP message will result in a complete loss of the video packet, causing a minimum of 33 ms of no video. In practice this scheme was found to be robust both in degrading and in recovering from interference

— highly desirable characteristics for a real-time system. Because no acknowledge or retransmission system is used, as soon as the wireless link is free of interference, new frames will be delivered with zero software synchronization delay. Additionally UDP is supported on numerous platforms, easing software implementation tremendously.

In examining details of 802.11n networks, there is a salient reason why a pure UDP over 802.11n approach is perhaps close to optimal. At the MAC-layers 802.11n actually implements a bitmap / re-try mechanism that automatically resends only lost Ethernet packets, similar to the mechanism UDT implements in software. Any software implementation of the same scheme will result in latencies and delays since the 802.11

Frame #	Type	Sequence #	Bitmap (64 bits)
4579	Block ACK	Start Sequence # 1381 + 64 = 1445	FF FF ... FF FF
4580	MPDU #1	1445	
4581	MPDU #2	1446	
4582	MPDU #3	1447	
4583	MPDU #4	1448	
4584	MPDU #5	1449 lost frame	
4585	MPDU #6	1450	
4586	MPDU #7	1451	
4587	MPDU #8	1452	
4588	Block ACK	Start Sequence # 1389 + 64 = 1453	FF FF ... FF EF
4589	MPDU #1	1449 retransmitted frame	
4590	MPDU #2	1453	
4591	MPDU #3	1454	
4592	MPDU #4	1455	
4593	MPDU #5	1456	
4594	MPDU #6	1457	
4595	MPDU #7	1458	
4596	MPDU #8	1459	
4597	MPDU #9	1460	
4598	MPDU #10	1461	
4599	Block ACK	Start Sequence # 1398 + 64 = 1462	FF FF ... FF FF

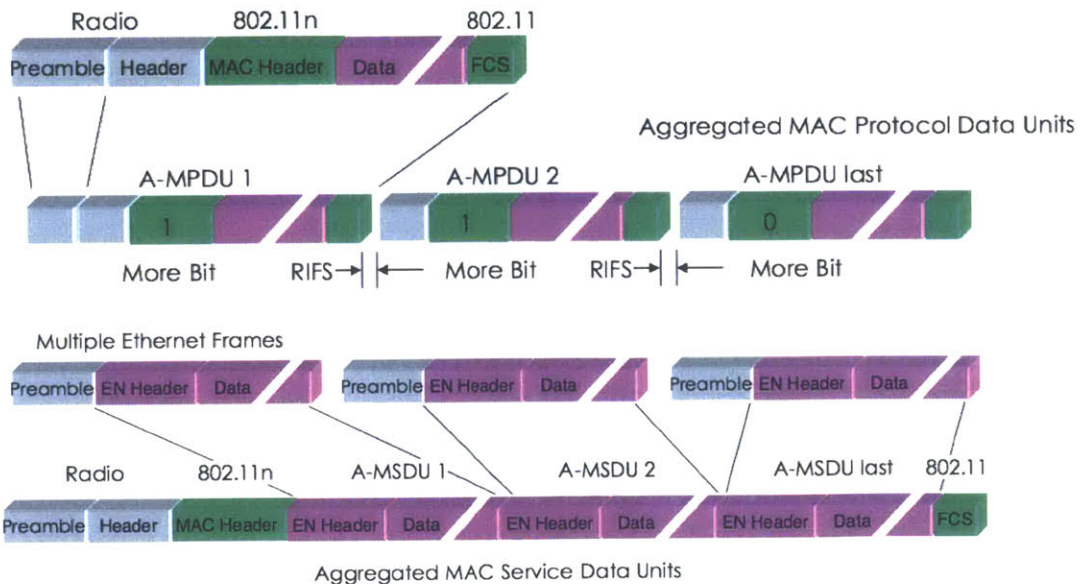


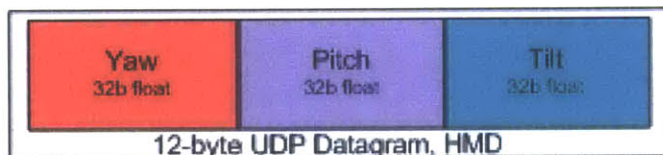
FIGURE 6-4:
TOP: BITMAP ACK/RETRY IN 802.11N, MIDDLE: A-MSDU PACKET COALESCING, BOTTOM A-MPDU PACKET COALESCING [LEUTART]

hardware is monitoring packet loss, signal strength and adjusting numerous other parameters all in real-time.

Secondly, 802.11n supports the ability to aggregate Ethernet frames into larger packets via two independent methods: A-MPDU or A-MSDU. Frame coalescing enables more throughput at the expense of larger packet loss during interference. Since a compressed video frame fragments into many 1500-byte MTU Ethernet frames, A-MPDU/A-MSDU aggregation will automatically combine the 1500-byte fragments and places them into larger messages. Owing to these key features, acknowledge-free transmission of video over UDP via 802.11n will most likely result in the best “reasonable” delivery of real-time streams.

6.2 Oculus Head Position and Dewarp communication

The GC6500 driver and APIs provide a very simple interface to control parameters of the dewarp engine. The calling parameters are the virtual horizontal position, the virtual vertical position, tilt, zoom (field of view) and a “divisor” which controls the effective granularity of panning movements. To allow cross-platform development and debugging of dewarp control, UDP again was selected as the transport mechanism for the head position information



```

28 // Dewarp parameters
29 dewarp_params_t dw_params;
30 dewarp_mode_t dw_mode = EMODE_WM_1PanelEPTZ;
31 STRUCT_Q_EPTZ_MODE_WM_1PANELEPTZ *eptz;
32 .
33 .
34 .
35
36 // Dewarp code
37 eptz = &dw_params.eptz_mode_wm_1paneleptz;
38 memset(eptz, 0, sizeof(STRUCT_Q_EPTZ_MODE_WM_1PANELEPTZ));
39 |
40 eptz->HPan = (int) occulus_yaw;
41 eptz->VPan = (int) occulus_pitch;
42 eptz->Tilt = (int) occulus_tilt;
43 eptz->Zoom = ZOOM*ANG_DIVISOR;
44 eptz->Divisor = (int) ANG_DIVISOR;
45
46 mxuvc_video_set_dewarp_params(ch, dw_panel, dw_mode, &dw_params);

```

FIGURE 6-5:
TOP: UDP DATAGRAM FOR CONTROLLING DEWARP ENGINE
BOTTOM: EXAMPLE OF UPDATING GC6500 DEWARP ENGINE

A very simplified packet structure of 12-bytes was defined, for roll, pitch, yaw. On the sender side the packet is formatted as several IEEE-754 32-bit floats, with bytes enqueued into the packet buffer in “network byte ordering” (big endian for Ethernet). Software on the router platform asynchronously receives these UDP packets and updates the dewarp engine on the next video frame sync. Because a user may change their head pose only slightly, a simple software filter was used to only communicate changes larger than a reasonable threshold, to minimize jitter in the dewarped video.

6.3 OpenWRT Router

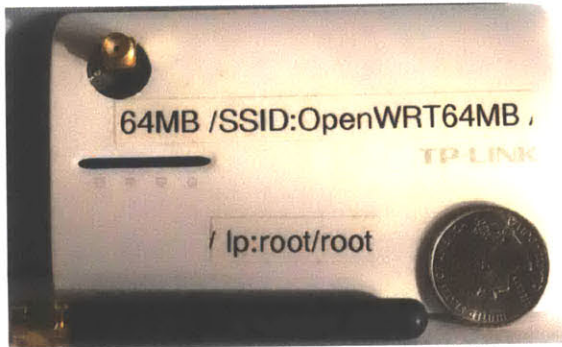


FIGURE 6-6: ROUTER USED FOR WIRELESS VIDEO TRANSMISSION

OpenWRT, an open-source Linux based router software platform was utilized due to ease of access, well defined tool chains and support for a broad number of router SoCs. The chosen router was also physically modified to upgrade the SDRAM from the standard 32 MBytes to 64 MBytes. An SMA connector was added to enable the use of an external antenna, bypassing the internal high-loss, low-sensitivity PCB antenna.

The wireless router is configured as a “client” to connect to the Tidmarsh Access Point, meaning the Quadrasense wireless video, control computer and sensor nodes are all accessible via the same, single network. The software written for the GC6500 platform is cross-compiled and stored on an SDXC memory card, attached via a USB reader/hub to the router. Since onboard non-volatile storage is limited to 4 Mbytes, external USB flash memory stores all executables, GC6500 drivers and modules, in addition to the raw fisheye image stream.

6.4 Syphon Video

An open-source program called SyphonNetCamera[26] was used as a starting point for the Syphon-based video sharing aspect of this project. SyphonNetCam is an OSX, Objective-C program which connects to web cameras over TCP and streams MJPEG data, decodes the data and presents the decoded video as a GPU texture to downstream Syphon applications. For this project SyphonNetCam was modified to:

- Support video reception carried by UDP & UDT transports

- Decode video in MPEG-2 and H.264 video formats to Syphon/GPU buffers using the x264 libraries
- Record any incoming video stream to a local file for offline use, with file naming based on time and date stamps

Router Sending Dewarped Video

```

vrt35001~/projects/Audi_dewarp_send/Debug$ ./rudi_dewarp_send.bin 192.168.64.110
Dewarped image to be created./net/pen/
Waiting for data on port UDP 41234
Initializing the video
Using dev_offset = 0
Using dev_offset_secondary = 1
Using vbl_buffers = 8
Initializing CH1 channel using /dev/video0
Getting resolution on CH1 channel
Getting the framerate on CH1 channel
Initializing CH2 channel using /dev/video0
Getting resolution on CH2 channel
Getting the framerate on CH2 channel
Initializing CH3 channel using /dev/video0
Getting resolution on CH3 channel
Getting the framerate on CH3 channel
Initializing CH4 channel using /dev/video0
Getting resolution on CH4 channel
Getting the framerate on CH4 channel
Registering callback function for CH1 channel
Starting the video on CH1 channel
Starting video
Entering camera event loop
Registering callback function for CH1 channel
Starting the video on CH1 channel
Starting video
sent frame: 0

```

OSX Control Computer

The screenshot displays the SyphonNetCamera application interface. A 'New NetCam' dialog is open, showing fields for Name, URL (http://192.168.2.3/mjpg/vi...), User, and Pass, with an 'Enabled' checkbox checked. Below the dialog, a 'Simple Client' window shows a video feed with a resolution of 1280 x 720 at 15 FPS. The background shows a code editor with C++ code and a terminal window displaying network-related logs, including UDP receive statistics.

FIGURE 6-7: END-TO-END VIDEO TRANSMISSION AND DECODING
TOP: ROUTER PLATFORM SENDING DEWARPED VIDEO VIA UDP
BOTTOM: OSX COMPUTER RUNNING MODIFIED SYPHONNETCAM, DECODING VIDEO

6.5 UniMAV and Software-In-The-Loop Simulator

The multi-rotor's Pixhawk flight computer receives control and transmits telemetry(i.e. latitude/longitude, yaw, pitch, roll) using a low-level, non-human readable protocol called MAVLink. MAVLink messages can command the multi-rotor to perform a number of semi-autonomous operations such as fly to a way point, takeoff or land. Creating low-level MAVLink messages directly to control the UAV is error prone and dangerous, thus a third-party Python program called MAVProxy was utilized. MAVProxy translates "human readable" commands into a succession of correctly formatted MAVLink messages which are transmitted to the UAV via the dedicated telemetry link. In reverse, MAVProxy can parse received MAVLink telemetry information into more human readable data such as yaw, pitch and roll angles. Unfortunately MAVProxy only offers a command-line interface or Python API for interaction.

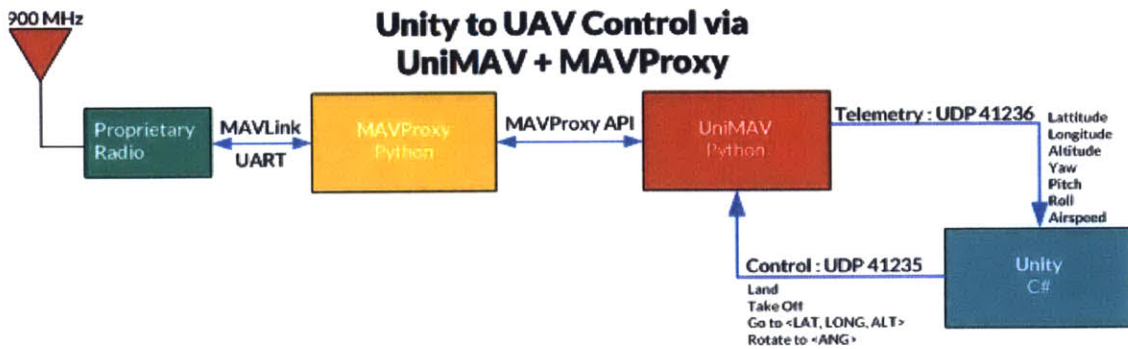


FIGURE 6-8: SYSTEM VIEW OF UNITY, UNIMAV AND MAVPROXY

To enable flow of telemetry and control information between Unity and the UAV, a custom program called UniMAV was created. UniMAV translates Unity high-level commands (i.e. fly to destination) into MAVProxy commands, which are ultimately formatted as MAVLink messages and sent via the telemetry link to the UAV. UniMAV also receives telemetry data from MAVProxy as soon as the data is parsed, and updates Unity with the latest information.

UniMAV uses two UDP sockets to communicate with Unity ; one socket (41235) listens for commands, while telemetry data is relayed to Unity via destination socket 41236. Figure 6-8 shows a diagram of data flow between the telemetry radio, MAVProxy and Unity. UniMAV listens for a stream of four 32-bit IEEE-754 floating-point values and makes the appropriate MAVProxy function calls after parsing the data. The first value UniMAV receives specifies a desired command for the UAV to execute with the following values acting as arguments. The implemented command-set is a very small subset of operations

the Pixhawk supports. Figure 6-9 shows key sections of UniMAV used to send telemetry and to process Unity requests.

```
19 # Commands:
20 # 0 = LAND, 1 = TAKEOFF, <alt> , 2= WAYPT <long,lat,alt>, 3 = YAW <yaw_angle> , 4 = RTL
21
22 # Commands come in on any IP, and this port via UDP
23 UDP_IP_IN = "0.0.0.0"
24 UDP_PORT_IN = 41235
25
26 # Telemetry data is send to this IP (Unity) via UDP
27 UDP_IP_OUT = "192.168.64.110"
28 UDP_PORT_OUT = 41236
29
30 # Create UDP socket and bind to address
31 control_socket = socket.socket( socket.AF_INET,socket.SOCK_DGRAM )
32 control_socket.bind( (UDP_IP_IN,UDP_PORT_IN) )
33 telemetry_socket = socket.socket( socket.AF_INET,socket.SOCK_DGRAM )
34
35 # First get an instance of the API endpoint
36 api = local_connect()
37 # Get the connected vehicle (currently only one vehicle can be returned).
38 vehicle = api.get_vehicles()[0]
39
40 def output_vehicle_state(attribute):
41
42     latitude = vehicle.location.lat
43     longitude = vehicle.location.lon
44     altitude = vehicle.location.alt
45
46     # Get vehicle parameters, convert from radians to degrees
47     yaw = vehicle.attitude.yaw * (360/(2*math.pi))
48     pitch = vehicle.attitude.pitch * (360/(2*math.pi))
49     roll = vehicle.attitude.roll * (360/(2*math.pi))
50     airspeed = vehicle.airspeed
51
52     telemetry_data = ""
53
54     telemetry_data += struct.pack( "f",latitude)
55     telemetry_data += struct.pack( "f",longitude)
56     telemetry_data += struct.pack( "f",altitude)
57     telemetry_data += struct.pack( "f",yaw)
58     telemetry_data += struct.pack( "f",pitch)
59     telemetry_data += struct.pack( "f",roll)
60     telemetry_data += struct.pack( "f",airspeed)
61
62 # Push the packet out
63 telemetry_socket.sendto(telemetry_data, (UDP_IP_OUT,UDP_PORT_OUT) )
64
65
```

FIGURE 6-9: KEY SECTION OF UNIMAV PYTHON CODE

To safely debug Unity/UAV interaction prior to actual flights, a software-based simulator was used to emulate the UAV. The Pixhawk software codebase offers an accurate simulator called “Software-In-the-Loop” (or SITL). The SITL program enables testing of MAVLink commands and flight plans even without a flight computer or physical UAV. SITL runs on standard desktop Linux and Windows computers and faithfully emulates operation of the physical Pixhawk-based UAV.

Calibration data from the physical UAV’s Pixhawk was extracted, so the SITL used same rates of movement (“time constants”) and control capability as the actual UAV. Access to a software simulator for critical UAV operations proved immensely useful, Unity and UniMAV interactions were debugged fairly quickly and confidence was built that the physical UAV would behave properly under Unity control.

An open-source software package called “APM” (Advanced Plight Mission) was used in conjunction with Unity and UniMAV/MAVProxy to observe simulated flights at a virtual

Tidmarsh. The upper image in figure 6-11 shows successful communication between UniMAV & MavProxy, while the bottom image shows the virtual quadcopter is correctly placed at Tidmarsh.

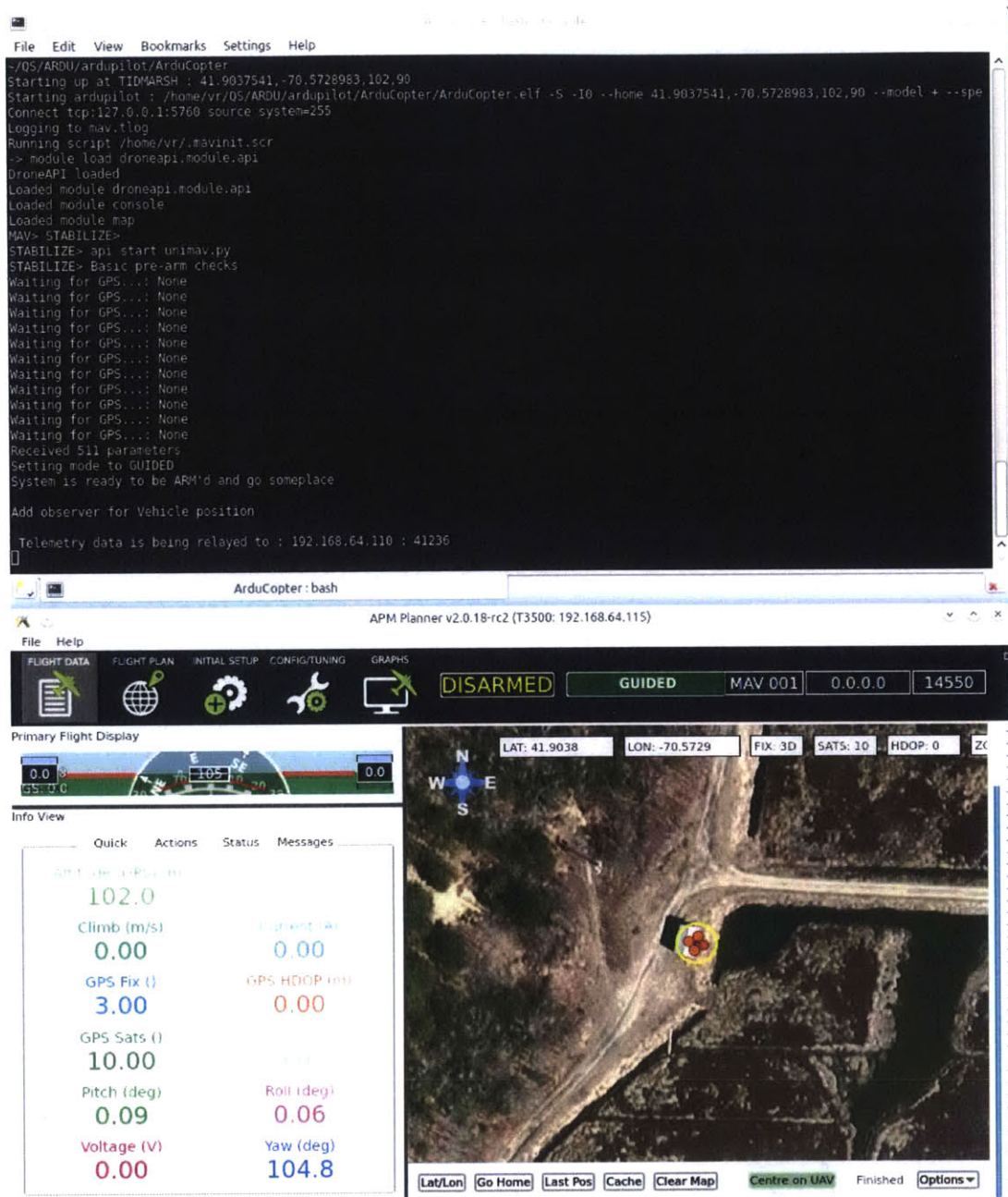


FIGURE 6-11 : VERIFYING UNITY+SITL IS FUNCTIONING, QUADCOPTER IS CORRECTLY PLACED AT TIDMARSH

TOP: CONSOLE-BASED ARDUICOPTER SITL
BOTTOM: APM PLANER SOFTWARE

6.6 Unity

An object-oriented approach was taken in modifying the existing Tidmarsh Unity project for Quadrasense. Unity virtual objects such as the UAV, virtual fisheye camera and so on are modeled after real-world elements, with interactions and behaviors attached to the relevant Unity objects. This made software design more straightforward as there was a one-to-one correspondence between physical objects, virtual objects and their respective behaviors.

6.6a UAV Agent and on UAV Virtual Cameras

A UAV agent called Drone was introduced into the Unity environment. To adjust the position of the virtual UAV in-tandem with movements from the physical UAV, a script called "UDPReceive.cs" was created ; this script subscribed to UDP packets sent by UniMAV. Received yaw, pitch, roll and latitude/longitude are used to re-position the virtual Unity UAV on every UDP update.

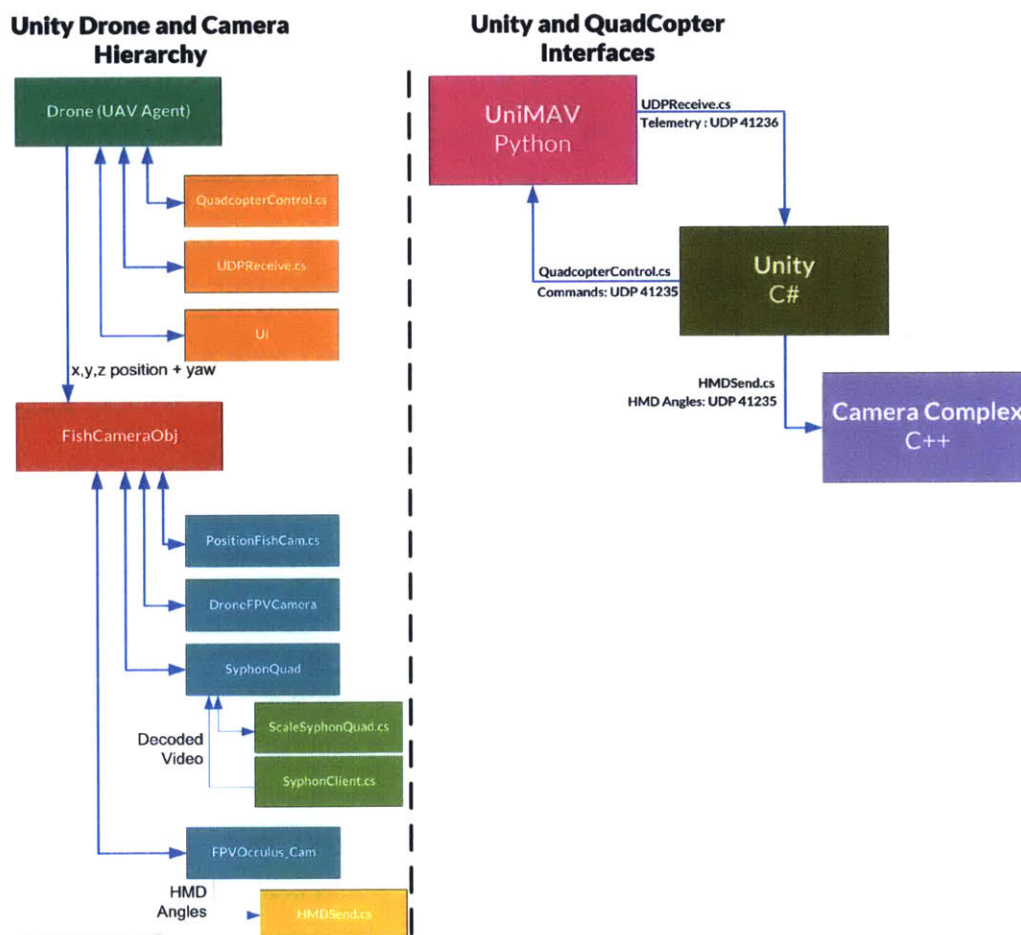


FIGURE 6-12:
LEFT: ORGANIZATION OF QUADRASENSE UNITY MODULES
RIGHT: SYSTEM-LEVEL INTEGRATION OF UNITY IN QUADRASENSE

A script called QuadcopterControl.cs was also attached to the Unity UAV object. This script, when in the mission planning mode, translates a mouse click in the Unity game coordinate system to standard latitude/longitude values and sends this via a UDP datagram to UniMAV, which results in the UAV flying to the desired (clicked) location.

A virtual camera with field-of-view modeled after the actual dewarped GeoSemi camera was attached to the UAV agent. The physical camera platform was attached to a two-axis gimbal in order to enable smooth video during UAV movements. To enable image registration adjustment between the physical camera and the virtual world, the Unity UAV camera behavior takes Unity x,y,z + yaw coordinates from the UAV, but tilt and pitch are set to zero assuming the gimbal will counteract any movement. This approach, in software, stabilizes the rendered camera. In addition, via Unity, offsets are provided to enable alignment and corrections for “open-loop” image registration between the rendered world and the real-time video. To enable the Oculus Rift interface, an additional Unity camera object named “FPVOculus_CAM” was added. The Oculus Rift camera object has access to the real-time head gaze and pose angles from the physical Rift hardware. A script called “HMDSend.cs” was attached to the Unity Rift object and real-time transmits the Oculus head angles, via UDP, to the router/camera sub-system.

6.6b Sensor markers and simple sensor visualizations

The original Tidmarsh Unity project simply overlaid data readings in the terrain where the sensor was located. To make the information more similar to a map with “call outs” for sensor location and sensor values, several 3D objects were modeled and created in Rhino. First a “sensor marker” depicted as a series of concentric circles, similar to the circles in the original UI mock-up, was created. This 3D object was imported into Unity and is auto center-positioned with each sensor node displayed in the Unity Tidmarsh environment. A shader was used to light the sensor indicator with some transparency as to not be visually overwhelming or distracting when alpha blending with real-time video.

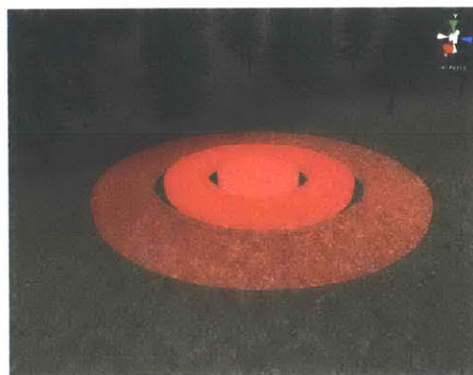
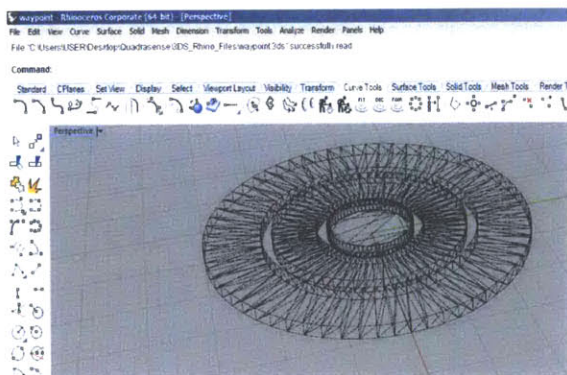


FIGURE 6-13: NEW SENSOR MARKER FOR UNITY
LEFT: DESIGN OF MARKER IN RHINO
RIGHT: MARKER IN UNITY AFTER IMPORT AND APPLYING SHADERS

A second item was modeled and created in Rhino as well, a “tag” for which sensor data appears inside. The choice of a chevron-like symbol was chosen for its ability to inform users of data values with 3D relief from the surface, similar to markers used in mapping programs and GPS navigation applications.

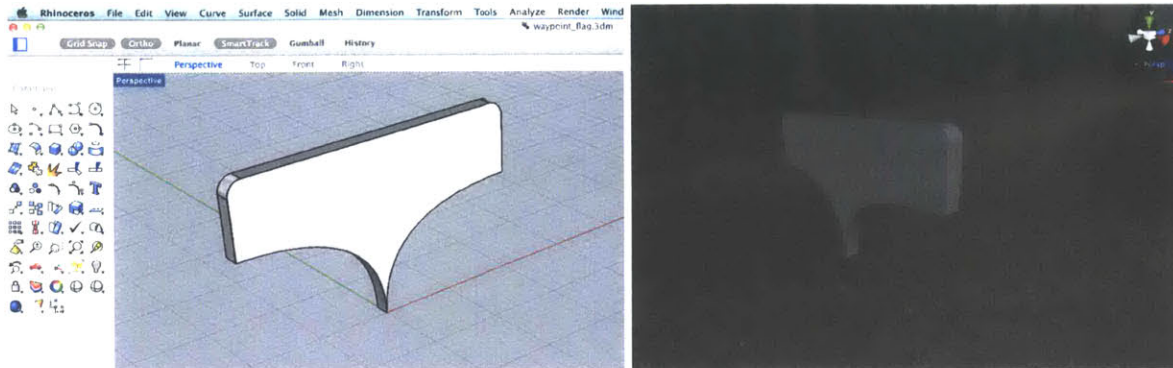


FIGURE 6-14: NEW DATA CONTAINER FOR SENSOR INFORMATION
LEFT: DESIGN OF CONTAINER IN RHINO
RIGHT: MARKER AFTER IMPORTING INTO UNITY, BEFORE SHADER-BASED LIGHTING

As the user moves around in Unity (in any view) the tags re-orient themselves automatically to face the camera, so the user always sees the sensor data (updated in real-time). While this visualization / rendering of sensor data is fairly basic, the approach was taken as a first step towards a more sophisticated augmented reality view. Any sensor data that Unity has access to via ChainAPI (i.e. temperature, humidity, lux levels, etc) can be displayed in the tags

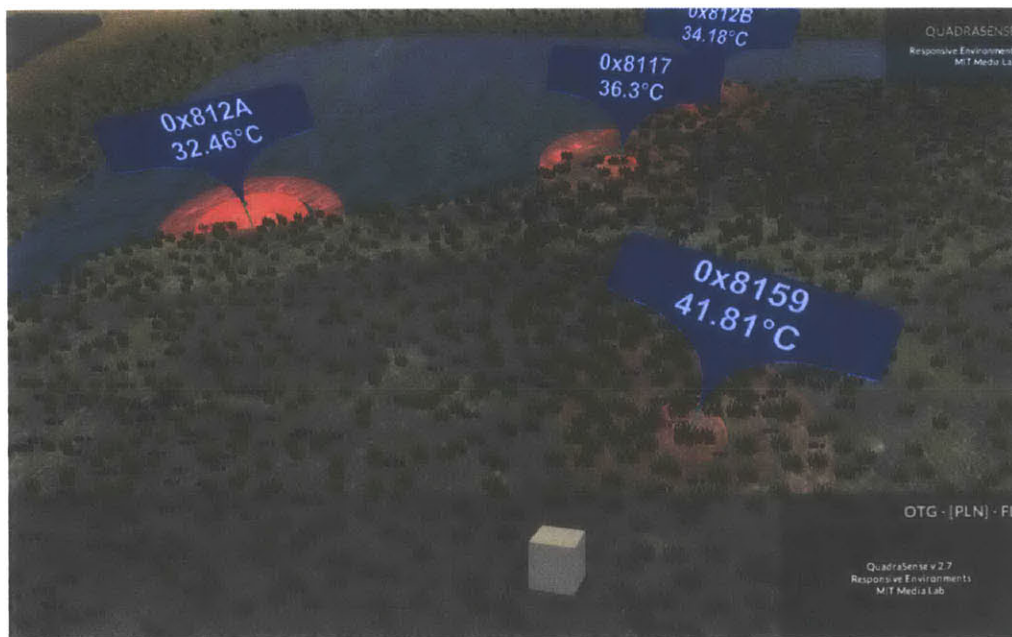


FIGURE 6-15: COMPLETED SENSOR VISUALIZATION IN UNITY, WITH REAL SENSOR DATA DISPLAYED

6.6c Syphon real-time video, alpha blending and scaling

As mentioned, Syphon GPU sharing technology was used to take decoded video frames from the SyphonNetCam application and bring them into Unity. To display the Syphon video in Unity, a quad primitive was used. In the graphics domain, a quad resembles a plane, but has only one rendering surface composed of four vertices and two triangles. In graphics, it is common to render simple 2D images (i.e. a billboard or sprite) into a quad, as other graphics primitives such as a plane or cube are more complex. In Unity the Syphon scripts were attached to a quad object which is a child of the FishEyeCameraObject

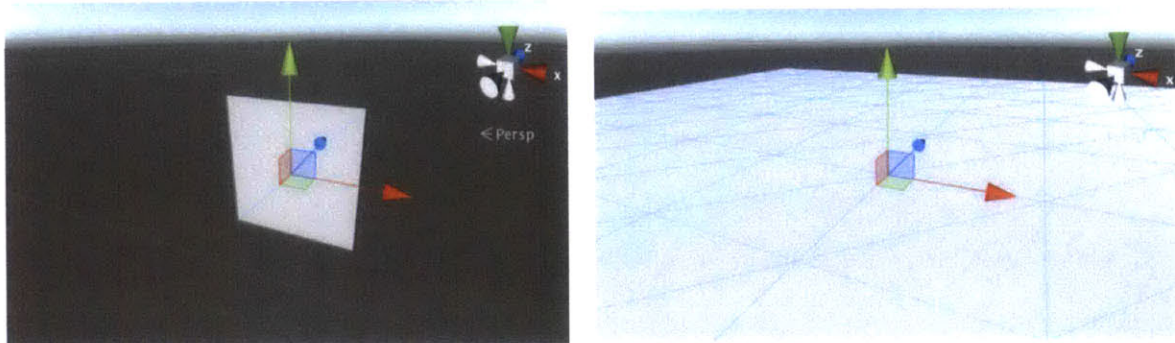


FIGURE 6-16 [UNITY]
LEFT: QUAD IN UNITY, TWO TRIANGLES, FOUR VERTICES
RIGHT: PLANE IN UNITY, TEN EDGES

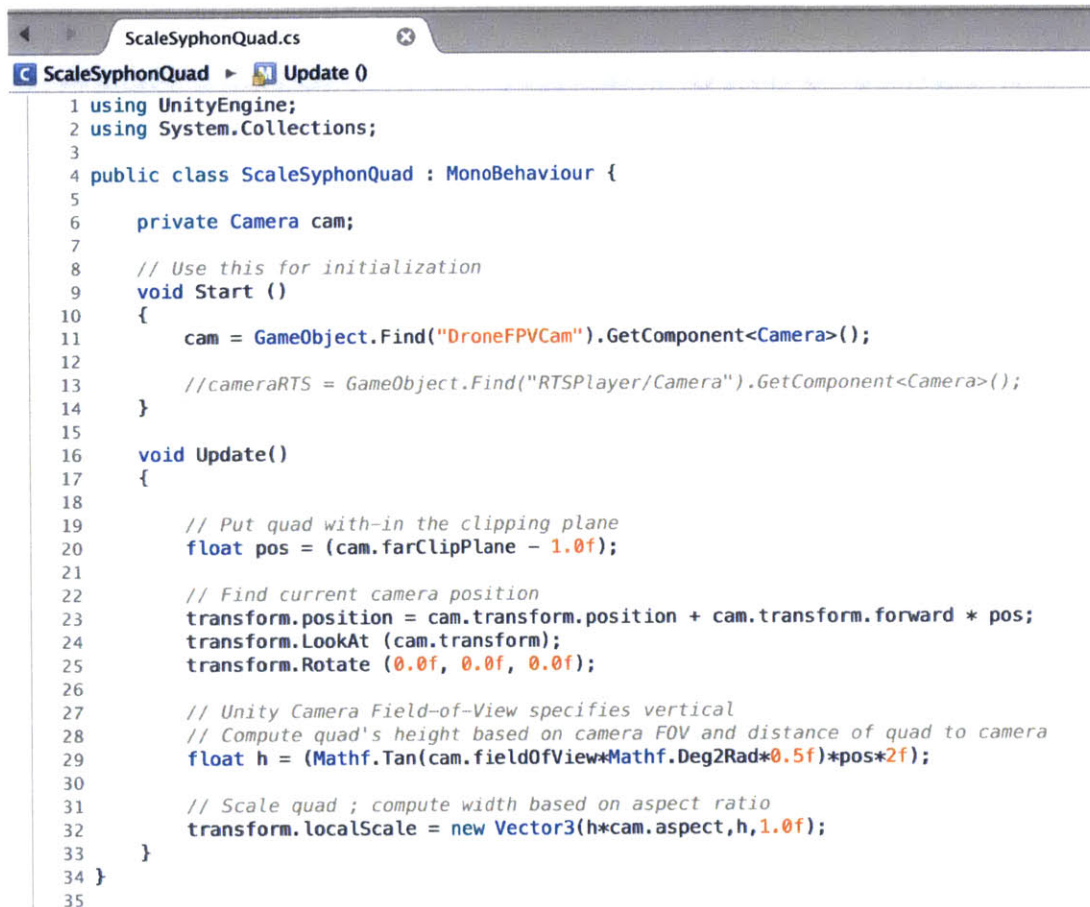
However, using the Syphon video with the quad posed an interesting challenge. The Unity rendered world is a 3D, and the real-time video was a 2D capture. In order for the augmented reality metaphor to function and to have clean registration between the virtual and physical worlds, it was important that the Unity 3D objects render “on top” of the real-time video stream. To ensure the virtual world is rendered on-top of the video stream, the quad was placed at a location near the far limit of the Z clipping plane. The Z clipping plane is the furthest distance from the camera that Unity will render. This approach ensures the Unity 3D objects are always rendered into the scene and will appear on top of the Syphon video.

Because the quad is placed in the Unity scene far away, it needs to be resized to fill the entire field of view of the Unity camera. Additionally several different Unity cameras are available (real-time video, virtual player on the ground) so the quad not only had to be placed and scaled appropriately, it also had to be oriented normal to the current camera's surface. To solve these issues, a script was written to always to auto-scale the quad to fill the screen and simultaneously guarantee it to be within the clipping plane of the active camera.

The following formula[27] is used for scaling the quad:

$$quad_height = \tan\left(cam_FOV * \frac{1}{2}\right) * quad_position * 2$$

In Unity, the FOV angle related to the vertical field of view, so the vertical quad size is computed first, and the screen's aspect ratio is used to determine the horizontal size.



```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ScaleSyphonQuad : MonoBehaviour {
5
6     private Camera cam;
7
8     // Use this for initialization
9     void Start ()
10    {
11        cam = GameObject.Find("DroneFPVCam").GetComponent<Camera>();
12
13        //cameraRTS = GameObject.Find("RTSPlayer/Camera").GetComponent<Camera>();
14    }
15
16    void Update()
17    {
18
19        // Put quad with-in the clipping plane
20        float pos = (cam.farClipPlane - 1.0f);
21
22        // Find current camera position
23        transform.position = cam.transform.position + cam.transform.forward * pos;
24        transform.LookAt (cam.transform);
25        transform.Rotate (0.0f, 0.0f, 0.0f);
26
27        // Unity Camera Field-of-View specifies vertical
28        // Compute quad's height based on camera FOV and distance of quad to camera
29        float h = (Mathf.Tan(cam.fieldOfView*Mathf.Deg2Rad*0.5f)*pos*2f);
30
31        // Scale quad ; compute width based on aspect ratio
32        transform.localScale = new Vector3(h*cam.aspect,h,1.0f);
33    }
34 }
35
```

FIGURE 6-17: SCALESYPHONQUAD.CS , APPROACH TO AUTO-SCALING AND PLACING SYPHON QUAD IN CAMERA FIELD OF VIEW

Using the Arducopter SITL to simulate UAV movements, the alpha-blending, object distance rendering and visualizations were all carefully examined. Figure 6-18 shows testing of ScaleQuad, Syphon and alpha-blending interaction. This scene is taken from a UAV agent above the ground with the top image capture showing no Syphon video blending and the bottom showing a test Syphon video stream. The image demonstrates key Unity sensor objects are not culled or occluded when using the scaled quad.

On-UAV view without video blending



On-UAV with video blending



FIGURE 6-18: TESTING OF SYPHON AND QUAD SCALING FOR AUGMENTED REALITY MODE

TOP: VIEW FROM UAV IN UNITY WITHOUT SYPHON VIDEO BLENDING
BOTTOM: SAME VIEW FROM UAV WITH SYPHON VIDEO ENABLED

Chapter 7

Quadrasense Results

7.1 Quadrasense Unity Views

There are three key modes of interaction with Quadrasense/Unity: “OTG” or On-the-Ground, “PLN” or Planning and “FLY” or Flying. In OTG users explore the Tidmarsh environment as if they were physically walking in the terrain ; this mode is nearly the same as the initial Tidmarsh Unity interaction. In PLN users explore Tidmarsh with a Bird’s Eye 2D view, and choose destinations for the UAV. FLY mode enables a virtual reality, augmented reality or telepresence interaction, taken from the perspective of the UAV.

7.1a) Mission Planning

Users initially interact with Quadrasense using the Planning or “PLN” view (shown in lower right of Unity screen). Users have the ability to pan in this view and zoom into sensor nodes of interest using standard keyboard arrow keys and mouse movements. Figure 7-1 shows a starting point for planning (top) and zooming into an area of interest (bottom). Note that the mode has not changed between images.

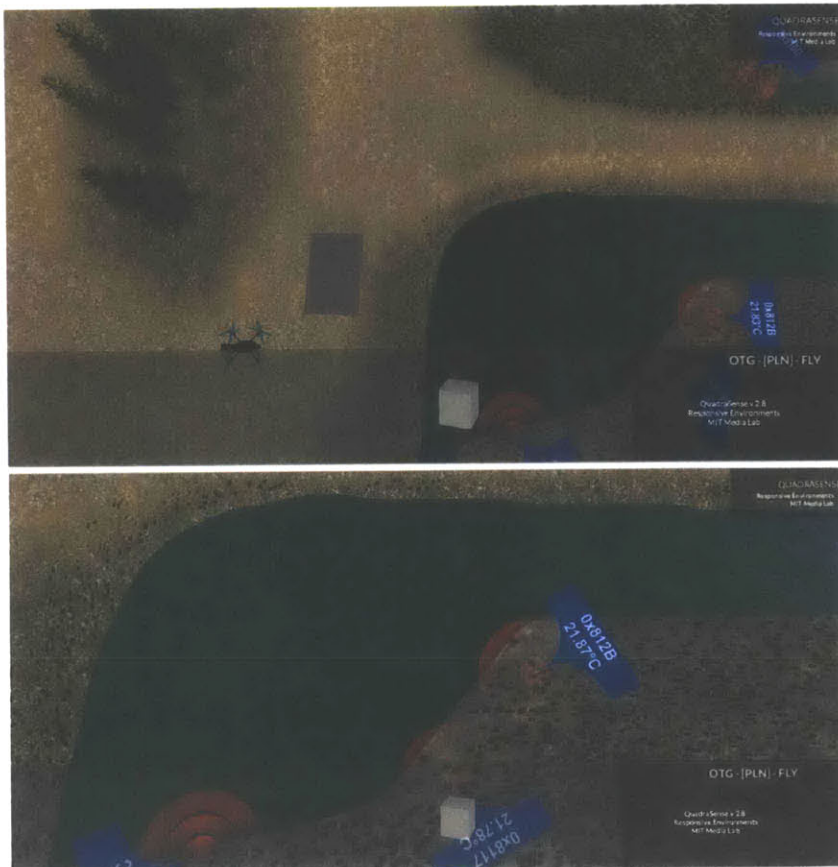
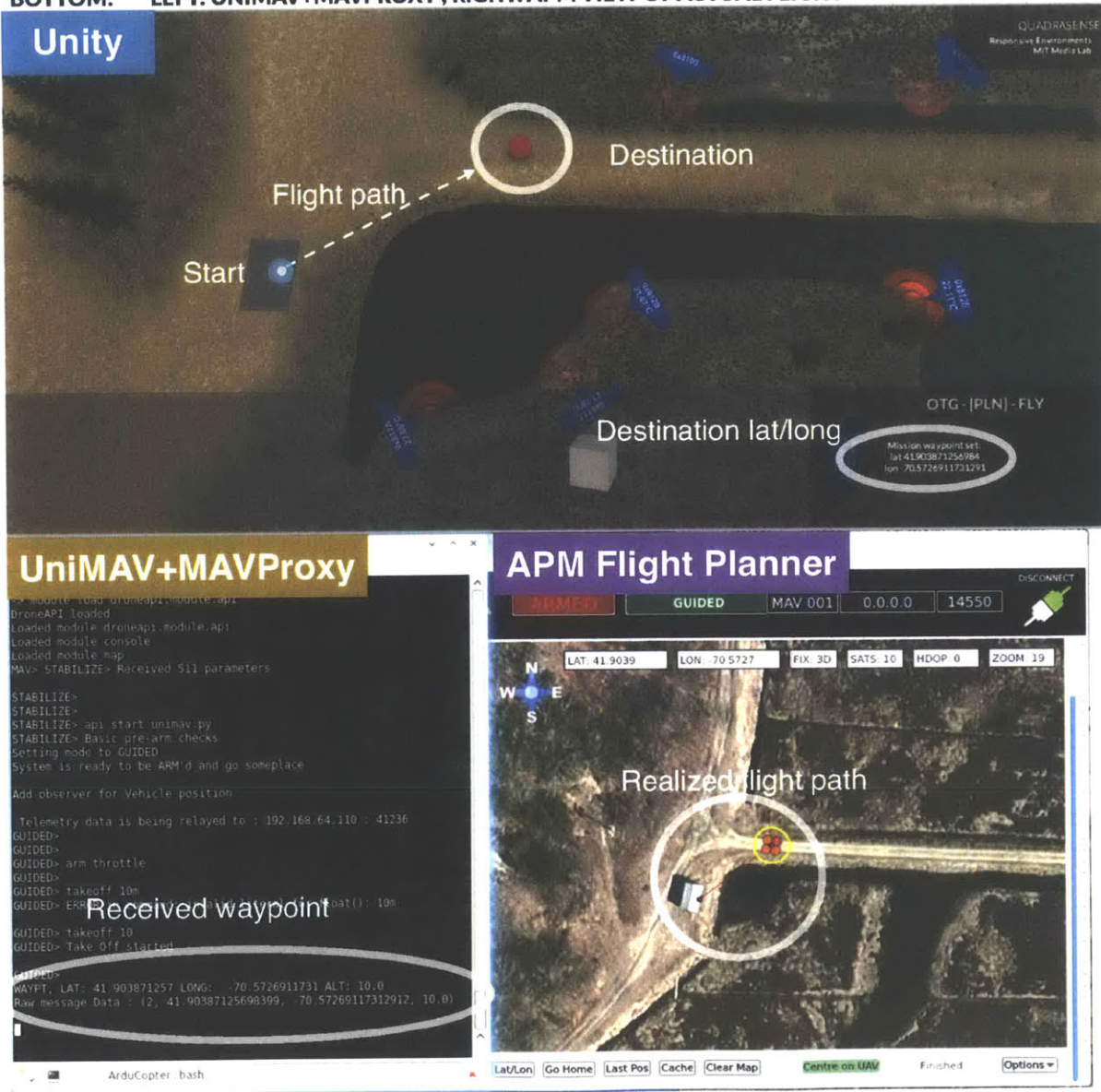


FIGURE 7-1: MISSION PLANNING VIEW IN QUADRASENSE/UNITY
TOP: INITIAL VIEW IN MISSION PLANNING
BOTTOM: ZOOMED INTO AREA OF INTEREST

Once an area of interest has been determined, users hold down a special activation key and mouse-click on the map to transmit the destination to the UAV. In the Unity screen capture, Figure 7-2(Unity), we can see a small red circle is placed on the clicked location and the computed “Mission waypoint” longitude/latitude is shown in the lower right of the top image. This lat/long pair is immediately sent to UniMAV via UDP and the UAV moves to the destination as a result. Taken from an end-to-end simulation, the circle in 7-2(UniMAV) shows the received destination latitude/longitude and the image in 7-2(APM) shows the traveled flight path of the UAV. This simulation successfully demonstrated Quadrasense/Unity control over the UAV, and synchronization between Unity and the real UAV.

FIGURE 7-2: SIMULATED MISSION IN QUADRASENSE/UNITY ; ANNOTATIONS IN WHITE
TOP: UNITY MISSION PLANNING
BOTTOM: LEFT: UNIMAV+MAVPROXY ; RIGHT: APM VIEW OF ACTUAL FLIGHT



7.1b) On UAV View: Telepresence and Augmented Reality Use Cases

Once a user has issued a destination, the main mode of interaction can be switched from PLN to FLY which changes the Unity camera perspective to match the actual on-UAV fisheye camera. In FLY mode there are three possible views, with the notion of enabling virtual reality, telepresence and augmented reality during a flight. Images in figures 7-3 through 7-5 are generated by Unity with terrain footage from Tidmarsh farms piped in through Syphon.

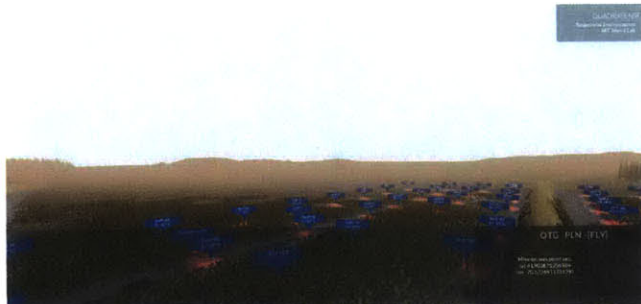


FIGURE 7-3: PURE UNITY ON UAV VIEW

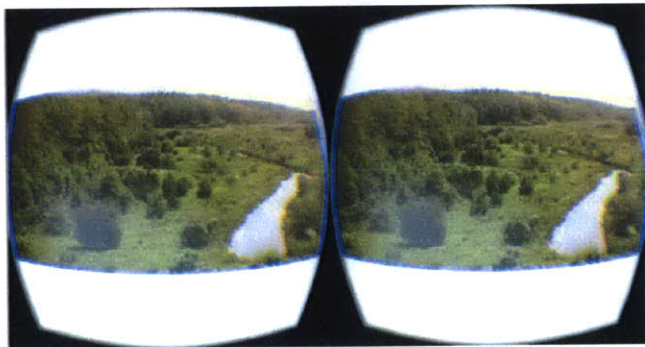


FIGURE 7-4: TELEPRESENCE ON UAV VIEW



FIGURE 7-5: AUGMENTED REALITY ON UAV VIEW

1. Users see a purely Unity generated view from the UAV ; the Unity UAV position is synchronized to the actual UAV, so this is a virtualized view of what the real UAV sees.

2. The user may hit a key which brings in the actual UAV's real-time video stream via Syphon into Unity (full-screen). As the user moves their head, correspondingly the camera complex sends a new view. In this mode there are no Unity graphics overlaid into the video. This mode implements the concepts of immersive remote telepresence.

3. By pressing the same toggle key again the video stream is blended with Unity generated sensor visualizations. The UAV's real-time position is continually tracked so the generated sensor graphics are correctly registered to the real world. This third-mode enables augmented reality interactions.

Not shown is the "OTG" or On-the-Ground mode. This is nearly the same as the existing Tidmarsh Unity interaction except for the updated sensor markers and virtual UAV agent.

7.2 Data captured from Live Flight

Actual results taken from flying Quadrasense at Tidmarsh Farms are presented in this section. The sequence of images shown follow the UAV on a simple point-to-point flight, demonstrating the interaction between Unity and the UAV.

Due to damage of the main fisheye camera prior to final test, these results are from an earlier version of the Unity project without updated map visuals featuring the circular sensor indicators and newer sensor tags. Only the mission planning and augmented reality use case is illustrated, the other (fully-functional) modes of operation such as telepresence (using the HMD), Oculus integration and mode transitions are not shown here.

Figure 7-6 shows the control computer used to drive the system, the UAV on the launching point at Tidmarsh and the UAV directly after autonomous take-off.



FIGURE 7-6: TEST FLIGHT AT TIDMARSH

The top image in Figure 7-7 shows the mission planning view in Quadrasense/Unity while the bottom image is a capture of the actual UAV at the same instant in time. The top image of figure 7-7 also shows real-time video (lower left) composited into the planning view with a requested destination depicted by a white circle. Lower-left video shows the expected water channel features based on the location, orientation and altitude of the UAV.

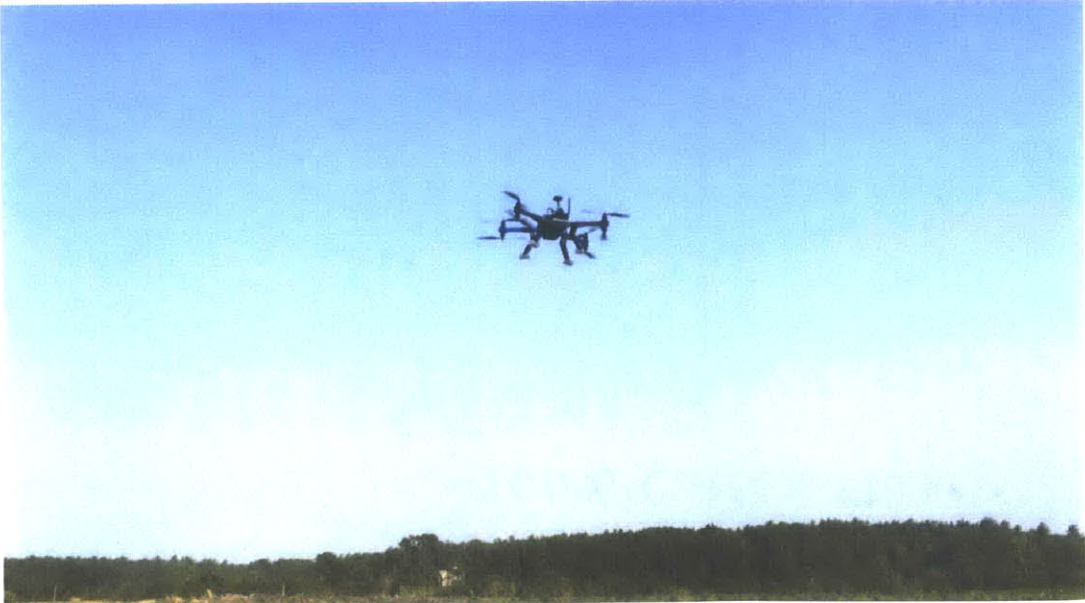
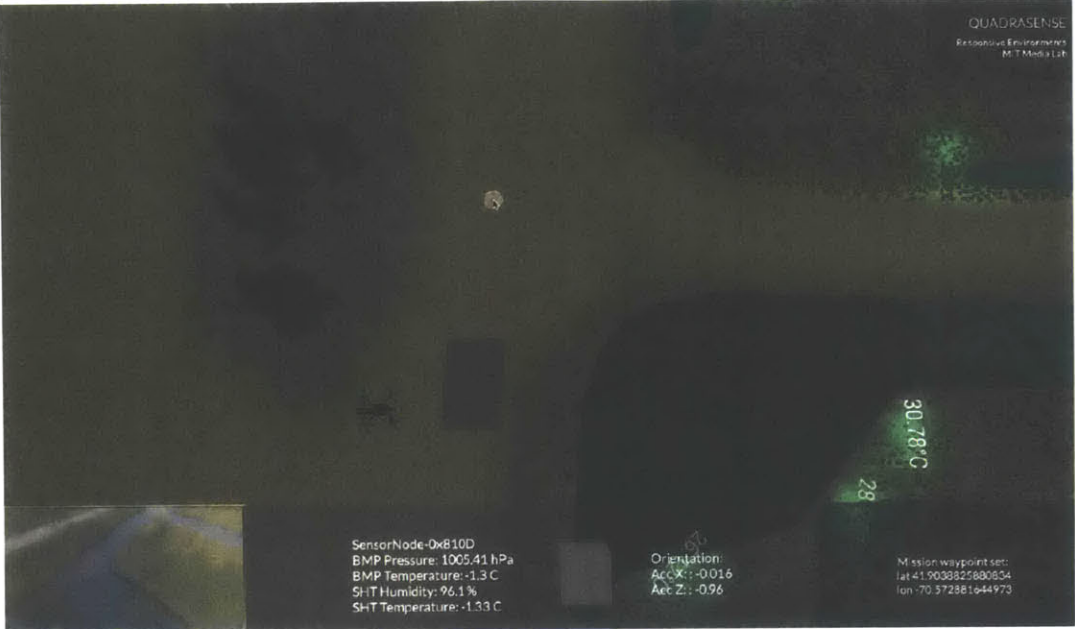


FIGURE 7-7
TOP: MISSION PLANNING IN UNITY
BOTTOM: REAL UAV POISED FOR ACTION

The images in Figure 7-8 are taken some seconds after the images from Figure 7-7. We can see the UAV has physically advanced, moving towards the mission destination. The top image is taken from the mission planning view in Unity, the bottom image is a time synchronized photo of the UAV. The Unity visualization lacks some foliage detail, there are in fact a greater number of trees in the vicinity of the UAV, as seen in the actual photo.

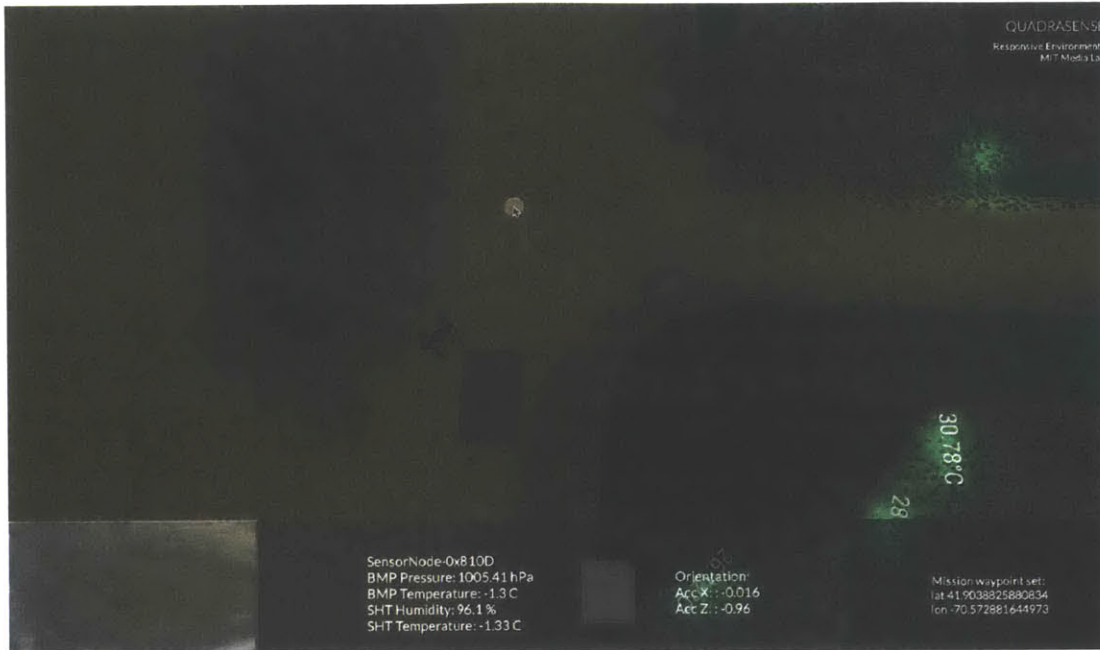


FIGURE 7-8
TOP: UNITY UAV IS MOVING TOWARDS DESTINATION
BOTTOM: SNAPSHOT OF REAL UAV MOVING IN TANDEM

Figure 7-9 (top) shows the UAV at the destination point, as rendered by Unity. The bottom image shows the augmented reality mode of Quadrasense, the live video stream is alpha-blended with sensor readings rendered by Unity. Due to use of an early implementation of the software, this augmented-reality view retains synthetic terrain in addition to the sensor renderings. In current versions the augmented reality mode disables all graphics except sensor data and associated markers, as seen in Figure 6-18 (bottom).

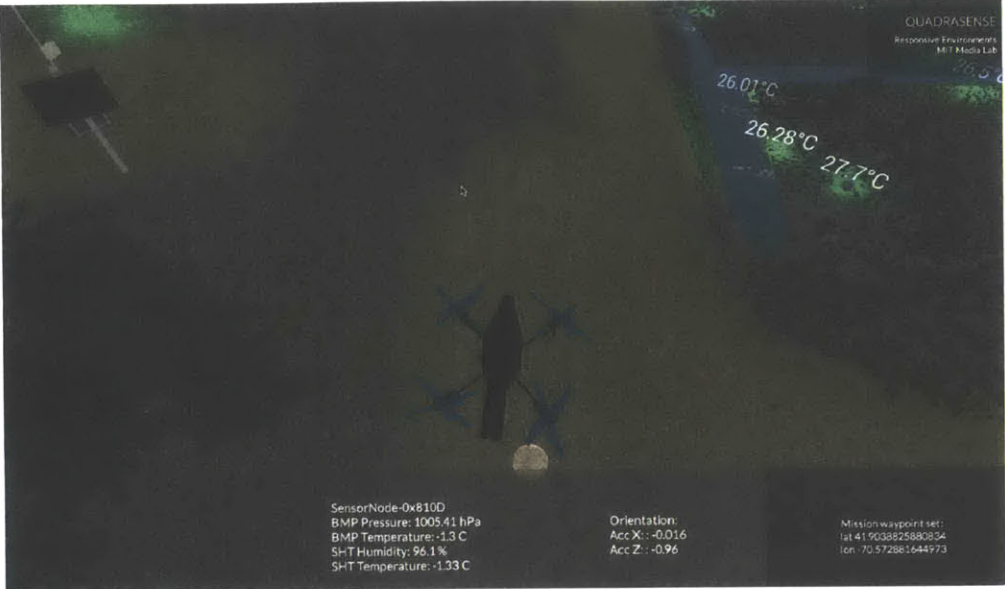


FIGURE 7-9
TOP: UNITY UAV HAS REACHED DESTINATION
BOTTOM : AUGMENTED REALITY VIEW GENERATED FROM UNITY AT DESTINATION

Figure 7-10 shows UniMAV/MAVProxy interface (top left console window) and the APM Planner software view (top right) captured during the entire Quadrasense UAV mission. APM recorded flight path, circled in white, confirms the UAV has physically flown to the correct location requested by Quadrasense. Other trajectories in APM (in red) are from earlier testing sessions.

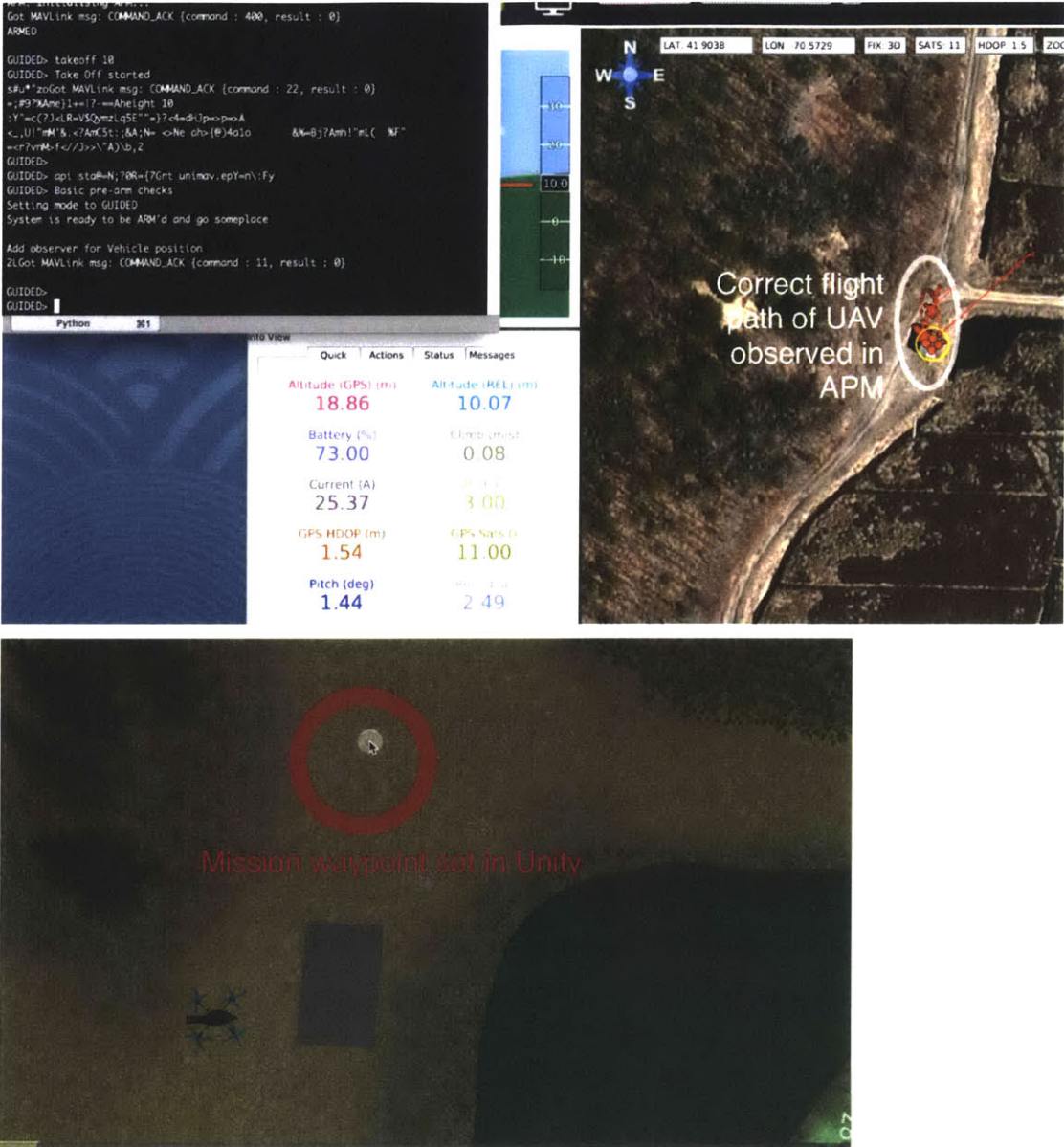


FIGURE 7-10:
TOP: APM TELEMETRY CAPTURE OF UAV DURING QUADRASENSE TEST MISSION
BOTTOM: TARGET DESTINATION FROM QUADRASENSE/UNITY

Chapter 8

Conclusions and Future Work

8.1 Contributions

Quadrasense makes three specific contributions. First, it provides UAV users with real-time telepresence, providing the experience of riding on a remote UAV sensing agent. Secondly, it blends data from in-field environmental sensor networks to augment that telepresence. Third, it enables users to seamlessly switch between a telepresence/augmented reality view to a virtual model of the environment. Actuation and control of the UAV agent is accomplished entirely through video-game based software.

Environmental Telepresence

The core system consists of a UAV with a unique real-time near omnidirectional camera system and a Head-Mounted-Display. Quadrasense uses low-latency networking transport, high-quality, low-bandwidth video compression and an optimized display pipeline to feed the video and graphics to the Head-Mounted-Interface. This gives UAV users the illusion they are on-board their UAV agent, immersed in the remote location. The user interface is designed to be intuitive and unobtrusive, allowing users to explore the remote geography however their interests and the data may guide them.

Augmented Reality

Users see graphics, informed by sensor data, synthetically blended with a visual frame of the sensed environment. The video game engine based software continually receives physical location information from a semi-autonomous UAV agent which is used to render graphics registered to incoming video stream, in real-time. Using a video game engine for rendering sensor data visualization enables rich augmented reality experiences without the complexity of custom graphical frameworks or trackers.

Virtual Reality

Quadrasense users may freely move between telepresence or augmented reality view of an environment to an on-ground, virtual reality view of the geography. The virtual reality implementation is based off the existing Responsive Environments Unity Tidmarsh visualization. This mode faithfully reproduces an actual landscape using synthetic graphics allowing users free-form exploration without the considerations of a physical UAV agent. Because the Quadrasense software is continually tracking the physical UAV, users in the virtual view can still see a rendered version of the agent and access related video streams.

8.2 Future Work

Quodrasense has integrated a very diverse set of hardware and software components in an attempt to realize the outlined vision. There are a number of areas for future exploration with-in the established frame work.

Visuals

By employing a video game engine for graphics rendering the possibilities for interactive and compelling visualizations is tremendous. The implemented sensor “tag” illustrates this – an “inert” 3D solid model, combined with shaders and associated software resulted in a useful sensor informant.

Future visuals could include circular modalities rendering pictorially into the red sensor marker, dynamic shapes based on sensing category and even the use of other Syphon applications to post-process and “remix” Unity’s renderings. These visualizations can be rich and yet latency free using the capabilities built into modern GPUs.

Interaction

The current interaction model has users click on areas of interest to explore, and the UAV follows. Because the sensor data agency concept is mostly realized in Quodrasense/Tidmarsh, future iterations of this program could guide users on sensor-informed paths, providing for “curated experiences”. The technical groundwork has been laid out to enable this, only a small amount of effort would be needed to support this modality.

Latency and Agent Tracking

The augmented reality implementation in Quodrasense relies on the UAV’s telemetry link for control data. Update rates from this link neighbor in the 5-15 Hz territory. By utilizing the WiFi radio instead this update frequency could be increased, possibly beyond 100 Hz. A 10ms update rate would be significant in reducing jerkiness.

Similar to the work by Fuchs[16] in low-latency AR, using a Kalman filter and other statistical and techniques, it is also possible to interpolate and estimate the actual position of the UAV, reducing the perceived AR latency in Unity.

3D Image Capture

There are numerous ideas that are planned for 3D image capture. These include using a pair of the GC6500 platforms to enable omnidirectional stereoscopic capability, enabling 360 deg image capture rather than 180 deg, adding depth sensing through stereo & structured light, and even using two completely separate UAVs to generate a stereo pair, providing variable inter-ocular distance.

Interface

The iDome project mentioned in related work examines one possible new immersive display. By jointly examining the capture and display problem, it is possible other unique arrangements could provide more compelling experiences than an HMD.

Video game engine / multi-party collaboration

The use of Unity to control a singular UAV is unique. By embracing modern video game capabilities, one could enable the control of numerous UAV agents, and also enable many different users to simultaneously experience the data from these UAV agents. Such accomplishments can extend collaborative and immersive sensing. The Quadrasense imaging hardware can already transmit eight different, independent dewarped views – capitalizing on this capability to enable multiple-party collaboration is highly feasible with additional time.

Safety

Many UAV systems implement “geofencing”, a safety measure that prevents the vehicle from going beyond a set distance from the takeoff point. Usually the geofence model is a sphere, limiting the range of the UAV motion to the boundaries of this particular shape. Because Quadrasense enables true remote operation of the UAV while abstracting the physical environment through synthetic graphics, there is an extra need for proper boundary enforcement.

By using graphics textures and simple boolean operations, it should be possible to “paint” a geofence mask. This would ensure the UAV can never enter forbidden zones or accidentally collide with a building or other physical edifice while still enabling navigation of safe areas. This geofence can take on any arbitrary shape and could even be discontinuous unlike the existing geofence implementation in the Pixhawk.

UAV and Sensor Interactions

A number of useful applications exist for the direct integration of UAVs and sensing. UAVs can carry a precise and accurate instrumentation package suitable for in-field calibration of sensor nodes. Nodes tuned for one band of operation (i.e. high temperature sensing) can be dynamically re-purposed or adjusted as seasons and environmental conditions change. The UAV agent can also be used to deploy sensor nodes with “fly by” collection of data. Such remotely deployed nodes may not need network connectivity, instead the UAV can connect and retrieve data on demand when passing by the node. For measuring data in tenuous conditions (i.e. flowing water and streams), the UAV can carry probes or waterproof cameras, attached via a tether. Such probes can be submerged, on-demand, in remote locations by lowering the altitude of the UAV in the geography. Such a sensing approach can augment the quantity of measurements in traditionally challenging and sparsely sampled environments.

References

- [1] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. & Anderson, J. Wireless sensor networks for habitat monitoring. in 88–97 (ACM Press, 2002). doi:10.1145/570748.570751
- [2] Hart, J.K. & Martinez, K., 2006. Environmental Sensor Networks: A revolution in the earth system science? *Earth-Science Reviews*, 78(3-4), pp.177–191.
- [3] Akyildiz, Kasimoglu, 2004. Wireless sensor and actor networks: research challenges, *Ad Hoc networks* 2, pp. 351–367.
- [4] "Self Powered Ad-hoc Network." Lockheed Martin, 9 July 2014. Web. 15 Aug. 2015. <<http://www.lockheedmartin.com/us/products/span.html>>.
- [5] d'Oleire-Oltmanns, S., Marzloff, I., Peter, K. & Ries, J. Unmanned Aerial Vehicle (UAV) for Monitoring Soil Erosion in Morocco. *Remote Sensing* 4, 3390–3416 (2012).
- [6] Ueyama, Jó. "Using Wireless Sensor Networks and UAVs for Urban River Monitoring." 26 Nov. 2012. Web. 15 Aug. 2015. <<http://people.ufpr.br/~tobias.dhs/aeba/seminario/12-Jo.pdf>>.
- [7] Fenstermaker, L., McGwire, K., & Wilcox, E. (2014, April 28). Nevada UAS Research and Test Site. Retrieved August 15, 2015, from http://web.evs.anl.gov/evs_uas_workshop_2014/presentations/Fenstermaker.pdf
- [8] Leighton, Joshua. System Design of an Unmanned Aerial Vehicle (UAV) for Marine Environmental Sensing. M.S. Thesis. Massachusetts Institute of Technology, 2013.
- [9] Ravela, S. Mapping Coherent Atmospheric Structures with Small Unmanned Aircraft Systems. in (American Institute of Aeronautics and Astronautics, 2013). doi:10.2514/6.2013-4667
- [10] Paradiso, J. A. & Landay, J. A. Guest Editors' Introduction: Cross-Reality Environments. *IEEE Pervasive Computing* 8, 14–15 (2009).
- [11] Hypermedia APIs for Sensor Data: A pragmatic approach to the Web of Things. in (2014). doi:10.4108/icst.mobiquitous.2014.258072
- [12] Davenport, Glorianna. Tidmarsh Living Observatory, 6 Mar. 2012. Web. 15 Aug. 2015. <http://tidmarshfarms.com/index.php?option=com_content&view=article&id=126&Itemid=86>.
- [13] UAV Systems for Sensor Dispersal, Telemetry, and Visualization in Hazardous Environments. in (2005). doi: 10.2514/6.2005-1237
- [14] Sinha, A., Tsourdos, A. & White, B. A. Multi UAV Coordination for Tracking the Dispersion of a Contaminant Cloud in an Urban Region. *EJCON* 15(3-4 15, 441–448 (2009).
- [15] Maimone, A., Bidwell, J., Peng, K. & Fuchs, H. Enhanced Personal Autostereoscopic Telepresence System Using Commodity Depth Cameras. *CG* 36, 791–807 (2012).

- [16] Towards Performing Ultrasound-Guided Needle Biopsies from within a Head-Mounted Display. 591–600 (1996). doi:10.1007/BFb0047002
- [17] Zheng, F. et al. Minimizing latency for augmented reality displays: Frames considered harmful. ISMAR 195–200 (2014). doi:10.1109/ISMAR.2014.6948427
- [18] Baker, S. & Nayar, S. K. A Theory of Single-Viewpoint Catadioptric Image Formation. IJCV 35, 175–196 (1999).
- [19] O. Rawashdeh, B. Sababha. "An Image-Processing-Based Gimbal System Using Fisheye Video". Computer Technology and Application 2 (2011) 85-93
- [20] Hals, Erik, Mats Svensson, and Mads Wilthil. "Oculus FPV." Norwegian University of Science and Technology, 25 Apr. 2014. Web. 15 Aug. 2015. <<https://github.com/Matsemann/oculus-fpv>>
- [21] Environmental Sensing Using Land-Based Spectrally-Selective Cameras and a Quadcopter. 259–272 (2012). doi:10.1007/978-3-319-00065-7_19
- [22] Bourke, Paul. "IDome Gallery." IDome. University of Western Australia, 12 June 2008. Web. 15 Aug. 2015. <<http://paulbourke.net/dome/UnityiDome/>>
- [23] Butterworth, Tom, and Anton Marini. "Introducing Syphon." 5 June 2012. Web. 15 Aug. 2015. <<http://www.syphon.v002.info/>>
- [24] Gu, Yunhong. "UDT Manual." UDT Manual. 8 Feb. 2011. Web. 15 Aug. 2015. <<http://udt.sourceforge.net/udt4/index.htm>>
- [25] Y. Ren, H. Tang, J. Li, H. Qian, "Performance Comparison of UDP-Based Protocols over Fast Long Distance Network". Information Technology Journal 8 (4) 600-604, 2009. ISSN 1812-5638
- [26] Hansen, Normen. Syphon-Camera., 21 May 2012. Web. 15 Aug. 2015. <<https://code.google.com/p/syphon-camera/source/checkout>>
- [27] "Field of View." Valve Developer Community. 2 Aug. 2012. Web. 15 Aug. 2015. <https://developer.valvesoftware.com/wiki/Field_of_View>