

SCANS FRAMEWORK:
SIMULATION OF CUAS NETWORKS AND SENSORS

A Dissertation
Submitted to the Faculty
of
Purdue University
by
Austin Riegsecker

In Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

December 2020
Purdue University
West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL STATEMENT OF
COMMITTEE APPROVAL

Dr. Eric T. Matson

Department of Computer and Information Technology

Dr. J. Eric Dietz

Department of Computer and Information Technology

Dr. John Springer

Department of Computer and Information Technology

Dr. John Gallagher

Department of Electrical Engineering and Computer Science

University of Cincinnati

Approved by:

Dr. Kathryne A. Newton

This work is dedicated to my best friend who has unconditionally supported me through my grad school journey. I look forward to the many other journeys we have ahead.

ACKNOWLEDGMENTS

I would like to thank Emily King for her support and willingness to help with editing. I would also like to thank my parents for endlessly calling me to see if I still had my mental faculties, because at times there were reasons to question. I would like to thank my committee members who have helped me grow professionally and academically during my time at Purdue. Without their encouragement this work would not have been completed in the time frame it was. Finally, I'd like to thank my past teachers whose words have driven me to achieve more.

PREFACE

This research presents a novel approach for simulating counter unmanned aerial system (CUAS) systems. The SCANS Framework, which stands for the Simulation of CUAS Networks and Sensors. The hope is for this framework to adapt and grow to fit the needs of the research community allowing for rapid prototyping and design of CUAS models. By employing the framework, planning and developmental cost can be lowered leading to more time for developing better systems. Hopefully, this project will grow as a research platform allowing teams the ability to perform system validation tests while still in the research planning stage. The overall goal is to have a modular, system-agnostic framework capable of adapting to the user's needs and personal choice for software implementation.

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES	xi
SYMBOLS	xiii
ABBREVIATIONS	xiv
ABSTRACT	xvi
CHAPTER 1. INTRODUCTION	1
1.1 Problem Statement	2
1.2 Significance	3
1.3 Research Question	4
1.4 Assumptions	4
1.5 Limitations	5
1.6 Delimitations	5
1.7 Definitions	6
1.8 Summary	6
CHAPTER 2. REVIEW OF RELEVANT LITERATURE	7
2.1 Sensors	7
2.1.1 Acoustic	7
2.1.2 Electro-Optical and Infrared	9
2.1.3 Radar and LiDAR	11
2.1.4 Radio Frequency	14
2.1.5 Inverse-Square Law	15
2.2 Consumer UAS Specifications	15
2.3 Communication Networks	18
2.3.1 Physical Wireless Network Characteristics	19
2.3.2 Wireless Network Performance Based Upon Environment . .	23
2.4 Current CUAS Simulation Models	26
2.5 Summary	27
CHAPTER 3. FRAMEWORK DESCRIPTION AND METHODOLOGY . .	28
3.1 AEIOU	29
3.2 Input Parameters	31
3.3 Model Outputs	42
3.4 Model Framework Layers	47

	Page
3.5 Agent and Model Functions and Variables	48
3.5.1 UAS Agent	48
3.5.2 Common Sensor Agent	49
3.5.3 Acoustic Sensor Agent	51
3.5.4 Radar Sensor Agent	53
3.5.5 Camera Sensor Agent	54
3.5.6 PassiveRF Sensor Agent	56
3.5.7 Common Communication Device Agent	57
3.5.8 CommunicationNode Agent	57
3.5.9 CommandAndControl Agent	59
3.5.10 Model Startup and Shutdown	60
3.6 Methodology for Evaluating the Framework	62
3.7 Summary	64
CHAPTER 4. ANYLOGIC IMPLEMENTATION	65
4.1 Building the Model	65
4.1.1 Creating the <i>Main</i> Agent	67
4.1.2 Creating the <i>UAS</i> Agent	75
4.1.3 Similarities Between Sensor Agents	77
4.1.4 Creating Specific Sensor Agent Types	79
4.1.5 Similarities Between Communication Agents	85
4.1.6 Creating Specific Communication Agent Types	86
4.2 Summary	89
CHAPTER 5. USING THE SCANS FRAMEWORK	90
5.1 Acoustic Experiments	91
5.2 Radar Experiments	97
5.3 Camera Experiments	101
5.4 PassiveRF Experiments	103
CHAPTER 6. SUMMARY	106
6.1 Future Work	107
6.2 Closing Thoughts	108
LIST OF REFERENCES	110
APPENDIX A. CUSTOM SENSORDATA DATA STRUCTURE	119
APPENDIX B. DATABASE TABLE STRUCTURE	123
B.1 Database Table: <i>model_runs</i>	123
B.2 Database Table: <i>sensor</i>	123
B.3 Database Table: <i>sensor_data</i>	124
APPENDIX C. DATA PROCESSING SCRIPTS	126
C.1 PowerShell: Sort and Reformat AnyLogic Output	126
C.2 Python: Generate <i>pColorMesh</i> Graphs	129

	Page
C.3 PowerShell: Quick Sort by SensorID for AnyLogic Output	130
VITA	132

LIST OF TABLES

Table	Page
2.1 UAS Specification Comparison	17
3.1 Determined Input Parameters Using AEIOU Analysis	33
3.2 Specific Additional Sensor Inputs	34
3.3 UAS Parameter Inputs	37
3.4 General and Acoustic Sensor Parameter Inputs	38
3.5 Camera Sensor Parameter Inputs	39
3.6 Radar Sensor Parameter Inputs	40
3.7 PassiveRF Sensor Parameter Inputs	40
3.8 DHS suggested sensor parameters	41
3.9 Communication Node and CC Parameter Inputs	43
3.10 SensorData Class Parameters and Datatypes	44
3.11 Abbreviated Agent Identification Codes	45
3.12 UAS Variable Names, Datatypes, and Default Values	49
3.13 UAS Variable Names, Datatypes, and Default Values	50
5.1 Model Completion Time for Yang50R10A	93
5.2 Results of Experiment Yang50R10A: Trial 1	96
5.3 Results of Experiment Yang50R10A: Trial 2	96
5.4 Results of Experiment Yang50R10A: Trial 3	96
5.5 Input Parameters For Experiment Farlik5KM10G	99
5.6 Iterative Tests for the Farlik5K10G Model	100
5.7 <i>Radar</i> Agent Input Parameters for Park et al. (2020)	101
5.8 Infrared <i>Camera</i> Agent Tests For Farlik, Kratky, Casar, and Stary (2019)	102
5.9 Infrared <i>Camera</i> Agent Tests Using Calculated PPF	103
5.10 Input Parameters for <i>PassiveRF</i> and <i>UAS</i> Agent	104

Table	Page
B.1 Database Structure for the <i>model_runs</i> Table	123
B.2 Database Structure for the <i>sensor</i> Table	124
B.3 Database Structure for the <i>sensor_data</i> Table	125

LIST OF FIGURES

Figure	Page
3.1 Framework Communication Chain	31
3.2 Object Class and Inheritance Structure	35
3.3 General Model Framework	47
4.1 AnyLogic Element Palettes	66
4.2 SCANS Framework Landing Page	67
4.3 Experimental Setup for Yang (2019)	68
4.4 Yang's (2019, p.35) Acoustic Node Layout for Experiments	69
4.5 Annotated Experiment 1 of Yang (2019) Within the <i>Main</i> Agent . . .	70
4.6 Experiment 1 of Yang (2019) Without Annotations	71
4.7 Zoomed View of Agent Population Elements and Point Node Collections	72
4.8 Annotated <i>UAS</i> Agent	76
4.9 Annotated Generalized Sensor Agent	78
4.10 Annotated <i>Acoustic</i> Agent as Implemented	80
4.11 Annotated <i>Radar</i> Agent as Implemented	82
4.12 Annotated <i>Camera</i> Agent as Implemented	83
4.13 Annotated <i>PassiveRF</i> Agent as Implemented	84
4.14 Annotated Generalized Communication Device Agent	85
4.15 Annotated <i>CommunicationNode</i> Agent as Implemented	87
4.16 Annotated <i>CommandAndControl</i> Agent as Implemented	88
5.1 Example of Records Created Each Model Run	90
5.2 Record of Each Agent and Parameters Used in Every Model Run . . .	91
5.3 Sensor Data Recorded by <i>CommandAndControl</i> Agents	92
5.4 Yang's First Experiment: Yang50R10A	94
5.5 Yang50R10A Experiment 1 Detection Graph Results	95

Figure	Page
5.6 Flight Path and Radar Placements of Farlik et al. (2019, p.6)	98
5.7 Radar Sensor Evaluation Based on Farlik et al. (2019)	99
5.8 Radar Model Based on Parameters of Park et al. (2020)	100
5.9 <i>PassiveRF</i> Agent Configured as DeDrone’s RF-100	105

SYMBOLS

A	Lens Angle (of Camera)
C	Circumference
c	Speed of Light
D	Distance
F	Frequency (of Radio Wave)
I	Energy Intensity
I_o	Threshold of Human Hearing
r	Radius
t	Time
λ	Wavelength
$^{\circ}$	Degree

ABBREVIATIONS

AEIOU	Activities, Environments, Interactions, Objects, Users
CC	Command and Control
CCW	Counter Clockwise
CN	Communication Node
CUAS	Counter Unmanned Aerial System
CW	Clockwise
dB	Decibel
dBm	Decibel-Milliwatts
DHS	Department of Homeland Security
DPR	Dropped Packet Ratio
FAA	Federal Aviation Administration
FOV	Field of View
FSPL	Free Space Path Loss
g	Gram
GHz	Gigahertz
IoT	Internet of Things
km	Kilometer
m	Meter
m/s	Meters per Second
MHz	Megahertz
mW	Milliwatts
PPF	Pixels per Foot
px	Pixel
RCS	Radar Cross Section

RF	Radio Frequency
Rot	Rotation
SCANS	Simulation of CUAS Networks and Sensors
SPL	Sound Pressure Level
Tx	Transmit or Transfer
UAS	Unmanned Aerial System/Vehicle
W	Watt

ABSTRACT

Riegsecker, Austin Ph.D., Purdue University, December 2020. SCANS Framework: Simulation of CUAS Networks and Sensors . Major Professor: Eric T. Matson.

Counter Unmanned Aerial System (CUAS) security systems have unrealistic performance expectations hyped on marketing and idealistic testing environments. By developing an agent-based model to simulate these systems, an average performance metric can be obtained, thereby providing better representative values of true system performance.

Due to high cost, excessive risk, and exponentially large parameter possibilities, it is unrealistic to test a CUAS system for optimal performance in the real world. Agent-based simulation can provide the necessary variability at a low cost point and allow for numerous parametric possibilities to provide actionable output from the CUAS system.

This study describes and documents the Simulation of CUAS Networks and Sensors (SCANS) Framework in a novel attempt at developing a flexible modeling framework for CUAS systems based on device parameters. The core of the framework rests on sensor and communication device agents. These sensors, including Acoustic, Radar, Passive Radio Frequency (RF), and Camera, use input parameters, sensor specifications, and UAS specifications to calculate such values as the sound pressure level, received signal strength, and maximum viewable distance. The communication devices employ a nearest-neighbor routing protocol to pass messages from the system which are then logged by a command and control agent.

This framework allows for the flexibility of modeling nearly any CUAS system and is designed to be easily adjusted. The framework is capable of reporting true positives, true negatives, and false negatives in terms of UAS detection. For

testing purposes, the SCANS Framework was deployed in AnyLogic and models were developed based on existing, published, empirical studies of sensors and detection UAS.

CHAPTER 1. INTRODUCTION

Autonomy of systems has been one of the goals for computers since their inception. Today, websites have code which help write themselves, autonomous cars work towards gaining public confidence and are optimized to drive safer and longer, and quad-copters are capable of following their owners and delivering packages with minimal, if any, human intervention. While these advancements seem recent in the consumer market, governments and militaries have made use of autonomous systems for decades with Unmanned Aerial Systems (UAS) first developed by the U.S. military at the end of the first World War. In contrast, it was not until 2006 when the Federal Aviation Administration (FAA) granted the first UAS pilot licence to a civilian.

Small, consumer UAS, also known as drones in the consumer market, are just one example of a complex systems-of-systems involving flight stabilization, waypoint or GPS navigation, autonomous flight, robust communication systems, and numerous safety measures which promote ease of use and safety. Due to the prevalence and ease of obtaining a UAS today, government organizations, home owners, and other property owners are concerned for their safety and privacy in regards to the ease at which small UAS can be operated. From these concerns, as well as illegal activity wherein small UAS were used, companies have developed complex systems in which to detect and interdict these small, nimble aerial vehicles. Due to numerous laws preventing their use, system complexity, and the research and development needed to create such a counter system, testing system performance is expensive and, in some cases, impossible until the countermeasures are implemented.

1.1 Problem Statement

Current research in the Counter Unmanned Aerial Systems (CUAS) space focuses on a specific detection, tracking, or interdiction method, and commercial CUAS systems leave performance metrics unknown or overstated. The problem addressed in this study is the inability to perform cost-effective planning when developing a CUAS system from a generalized sensor detection and sensor communication perspective. Due to current laws, lack of availability, and nature of the application, CUAS systems are difficult for potential customers to procure and test for the system's claimed efficiency given a customer's unique environment. Using software simulation modeling, a mock-up of an environment can be made and rapid-prototyping is possible due to the low cost associated with virtual models. Using the parameterization of sensors, UAS, and sensor communication allows for optimizations of sensor placement to be found based upon expected system needs.

According to the FAA report posted March 2018 (Federal Aviation Administration, n.d.), the consumer-based small UAS fleet is expected to more than double by 2022 to 2.4 million, with an annual growth rate of 16.9%. The annual growth rate for remote pilots are forecast to be 32.4% over the next five years. In addition to the growing number of small UAS, commonly referred to as drones, there have been three incidences where they have been used to get close to heads of state; namely German Chancellor Angela Merkel in 2013 (Gallagher, 2013), the White House lawn landing in 2015 (Schmidt & Shear, 2015), and an attempted attack on Venezuelan President Nicolas Maduro in August of 2018 (Koettl & Marcolini, 2018). Furthermore, UAS have been used for carrying contraband into prisons (Phillip, 2014), smuggle drugs over country borders (*Drug delivery drone crashes in Mexico*, 2015), and have twice flown over French nuclear reactors (De Clercq, 2018; Labbe & Rose, 2014).

Research teams, commercial companies, and governmental agencies have been working to detect and counter the threats posed by small UAS in numerous

ways. Methods for detection include radar, radio frequency scanning, sound signature, infrared camera capture, and via standard, visual cameras, among other methods (Drozdowicz et al., 2016; Liu et al., 2017; Nguyen, Ravindranatha, Nguyen, Han, & Vu, 2016; Shin, Jung, Kim, Ham, & Park, 2017; Solomitskii, Gapeyenko, Semkin, Andreev, & Koucheryavy, 2018). According to the co-director of the Center for the Study of the Drone at Bard College, New York, more than two-hundred companies are working on the CUAS problem (Watson, 2018, section 3, para. 2). However, despite all of these products and research, Brett Velicovich, the current CEO of Drone Experts and a former U.S. soldier, says “companies are spending millions of dollars to defeat a threat that cost about \$500 apiece” (Watson, 2018, section 3, para. 9). A down-side to many of these products is the lack of swarm detection, with the the first commercially advertised solution being DroneTacker 3.5 by DeDrone (*Drone Swarm Detection Capabilities*, 2018).

1.2 Significance

Developing a flexible model framework with generic, parameterized sensors for UAS detection, based upon mathematical models for sensing distance and radio communication distance, will provide verifiable performance metrics for proposed CUAS systems. Such a model would also be capable of providing optimizations for sensor placements, of which is currently missing in available real-world systems due to current implementation cost and environment variability.

Small, consumer drones and UAS are not going away; and among their many benefits, there are also several security pitfalls. Legally, in the United States, these UAS must fly below 400 feet (122 meters) (*Recreational Fliers & Modeler Community-Based Organizations*, 2019). However, the DJI drones software allows flights up to 1,500 feet (457 meters) and the machines are capable of altitudes over 20,000 feet (6092 meters) (Atherton, 2016; *Mavic 2 - Specifications, FAQs, Videos, Tutorials, Manuals*, n.d.). This, coupled with the ability of some drones to carry

several pounds of payload (*How Much Weight Can A Drone Lift?*, n.d.), gives rise to a unique concern not seen before on the civilian battleground. That being the capability for anyone to weaponize small, consumer UAS and launch attacks on large gathering spaces or restricted spaces undetected, and therefore unstoppable (De Clercq, 2018; Finn & Wright, 2012; Labbe & Rose, 2014).

By developing a system to measure effectiveness and creating better, more capable systems of detection and counter-measure, crimes and terrorism performed with these devices such as in *Drug delivery drone crashes in Mexico* (2015) and Phillip (2014) can be mitigated. This can then extend to educational courses, certifications, and businesses to help raise awareness about these issues and combat the bad actors. The market for which this technology would be applicable includes secure government areas of operation, military zones, correctional facilities, airports, schools, and large companies.

1.3 Research Question

What is necessary to develop and validate an extensible and general use case CUAS simulation model, with a focus on sensor simulation and a detection-alert network communication system?

1.4 Assumptions

The assumptions for this study include:

- Performance data reported by other research teams are accurate.
- Regarding the AnyLogic implementation, unidirectional agents are assumed to be at the proper pitch (regarding angle) for UAS detection
- The detection of a UAS by a sensor agent is assumed to have occurred when all sensor threshold criteria are met

1.5 Limitations

The limitations for this study include:

- Due to AnyLogic programmatic limitations, features of the framework, such as object inheritance cannot be strictly adhered to.
- Functions specifically related to the z-axis space (having an upper and lower bound of sensor detection height) could not be implemented due to time constraints. However three-dimensional (3D) space is used for calculating all distances.
- Due to sensor complexity and unavailability of sensor parameters, LiDAR could not be implemented into the system.
- The SCANS framework is meant to be a continuously evolving framework and its implementation is currently limited by the single viewpoint of the researcher.

1.6 Delimitations

The delimitations for this study include:

- The SCANS framework is system agnostic, but will be implemented for testing purposes in AnyLogic 8 Professional.
- The implemented model will only test open space, clear sky environments so as to limit theoretical signal interference in the form of reflections, refractions, and diffractions.
- The implemented model will focus solely on sensor- and communication-level interaction and will not implement an advanced detection, tracking, or interdiction system.
- This work is not concerned with advanced network routing protocols.

1.7 Definitions

The following are definitions that provide a common understanding of ambiguous concepts within the UAS and CUAS space.

Unmanned Aerial System (UAS): A small, consumer-based rotary-wing or fixed-winged craft weighing less than 55 pounds per the Federal Aviation Administrations specifications in Part 107 Federal Aviation Administration (2018). Commonly referred to as a drone in the consumer market. Also known as UAV, quad-copter or hex-copter.

Counter-UAS (CUAS) System: Any system capable of either detecting, tracking, interdicting, or any combination thereof, one or more UAS.

Agent: An entity within AnyLogic containing program data and functions capable of interacting with other agents. Symbolically, it is synonymous with a “class” in object-oriented programming.

Sensor: An abstracted entity implemented as an agent within AnyLogic, which performs calculations against the rogue UAS or the environment and generates data related to a detection event.

Detection Event: An action triggered when all detection criteria has been met, which then sends an alert message to Command and Control.

1.8 Summary

This chapter provided the scope, significance, research question, assumptions, limitations, delimitations, definitions, and other background information for the research project. Chapter 2 provides background information on UAS, sensors, wireless communication networks, and simulation models. The chapter covers specifications, functions, the mathematics used to determine sensor performance, and recent work performed within the CUAS simulation space.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter provides a review of the literature relevant to the simulated model's function and form. A review is given for various types of sensors and their functions, a collection of UAS specifications, and communication network technologies. Review is also given for several CUAS system simulations.

2.1 Sensors

The following provides a description of sensor types used in CUAS activities and the theoretical mathematical equations for calculating their range and performance. Also provided are experimental examples wherein the specific sensor type is used. In total, the sensors discussed are acoustic, electro-optical (visual and infrared cameras), radar, LiDAR and passive radio frequency (RF) sensors.

2.1.1 Acoustic

Acoustic sensors, also known as acoustic wave sensors, sound sensors, or simply, microphones, are inexpensive, versatile, and abundant. This makes them common components in CUAS research (Mezei, Fiaska, & Molnar, 2015; Mezei & Molnar, 2016) and professional systems (Busset et al., 2015; Hauzenberger & Holmberg Ohlsson, 2015; Sathyamoorthy, 2015). The technology used to create acoustic wave sensors can be transformed to form pressure, temperature, torque, mass, and humidity sensors, among many others (Drafts, 2001). These sensors work by measuring the voltage changes created by pressure waves against a thin diaphragm or quartz surface, resulting in sensitive changes in electrical resistance (Drafts, 2001). Acoustic sensors have a limited range when used in an

omni-directional capability, however using funnels or parabolic dishes provide a focusing of sound waves allowing for greater detection capabilities at the cost of the maximum sensing angle. The U.S. Department of Homeland Security (2019) states that with the use of additional computer tools such as software analytics and UAS sound sample databases, acoustic sensors can be used for triangulation and determining the speed and direction of an incoming threat because of the Doppler effect. The FAA, however, in its UAS technical considerations informs that acoustic sensors do not provide the necessary range to warrant their use as a primary detection device, and should typically be considered only as validation or secondary sensors (Federal Aviation Administration, 2019, p.2).

Two formulas, the sound intensity formula and the Inverse-square law, are used to measure sound wave intensity and the distance between a sound source and a sensors, respectively. The sound intensity formula (Moore, 1995, p.11) is given as

$$dB = 10 \log\left(\frac{I}{I_o}\right), \quad (2.1)$$

where $I_o = 10^{-12} W m^{-2}$. The term I_o is the reference value commonly used in sound applications representing the threshold of human hearing (Nave, 1999). The Inverse-square law formula and explanation is given in Section 2.1.5 due to other sensors relying on the formula.

In terms of testing done in the acoustic CUAS space, Farlik et al. (2019) performed four years worth of tests on various small consumer and professional grade UAS including the DJI Phantom 2, DJI DJI MAVIC Pro, 3DR X8, DJI Inspire 1, and the TAROT F650. The specifications of these machines are listed in Section 2.2, Table 2.1. From Farlik et al., effective detection of a drone using a SONY F-V120 was between 70 and 300 meters using the TAROT F650 and DJI S800E UAS. Hauzenberger and Holmberg Ohlsson (2015), Mezei et al. (2015), Busset et al. (2015), and Mezei and Molnar (2016) each show successful detection of UAS using acoustical technologies, however these systems are had a limited range of

less than 150 meters. There is also a distinct lack of computational times for the detection process. It was noted in Mezei et al. (2015) that when using pre-recorded sound samples of 30 seconds, their system had a confidence score of 84%, while at 140 seconds the confidence score was only 57%.

2.1.2 Electro-Optical and Infrared

CUAS systems utilizing imaging solutions depend on collecting enough information to accurately classify a rogue UAS. For this purpose, an image subject (the UAS) must meet the minimum pixel per foot (PPF) requirements of an image detection system. As the PPF specification depends upon the detection algorithm and camera specifications, no one answer exists. Few published papers report on the PPF requirement of their algorithms and instead opt to describe the maximum functional distance of their systems, if any distance is reported at all.

Liu, Qu, Liu, Zhao, and Chen (2018) designed and implemented a 30 camera array and detection algorithm for detecting UAS, however no specifications were given as to the maximum distance the system was capable of for the purposes of detecting and classifying a UAS. Ganti and Kim (2016) developed an implementation for detection and tracking of small UAS using an eletro-optical system, with acoustic sensors used for confirmation. Ganti and Kim noted their system, as every other electro-optical solution, relied on the number of pixels the UAS occupied in the video frame. From this, their system could identify small UAS up to 61 meters away and commercial airplanes up to 1524 meters.

The following equations can use a given PPF to calculate the maximum distance an object can be detected, reliant upon an individual system's performance. Using algebra would allow the PPF to be calculated from the

maximum distance detected and camera resolution, as in Ganti and Kim’s study. The maximum FOV is calculated by

$$FOV = \frac{Resolution\ Width}{px./ft.\ Required}. \quad (2.2)$$

With the maximum FOV calculated, the necessary lens angle for the desired viewing distance can be calculated. Inversely, if the CUAS system must be able to detect a UAS at a certain distance, the lens angle can be calculated based upon the camera resolution. This is achieved by calculating the circumference (C) based upon lens angle (A) with

$$C = FOV * 360/A. \quad (2.3)$$

From the calculated circumference, the radius is derived, which provides the maximum detection distance of the camera. The radius of the circle is calculated with

$$r = \frac{C}{2\pi} \quad (2.4)$$

Infrared imaging is also used in the CUAS space and operates in much the same way as a standard camera does, but with sensors sensitive to the near-infrared and infrared spectrum, instead of visual light. Qi, Zhang, and Xu (2018) proposes a drone detection method using 460x520 pixel images, whereby the highest achieved precision rating was 97.8% with a 98.2% recall rate. This was achieved with a pixel density of 4 px/ft giving a maximum detection range of 288 feet, or 87 meters. Farlik et al. (2019) also tested for infrared, or heat-based imaging, and concluded that with a MATIS HH thermal imaging camera “effective detection of a UAV was possible at distances up to 1.8 km, recognition up to 1 km, and identification up to 0.4 km” (p.19). By using a FLIR A40M infrared camera and no additional lenses, the maximum detection distance was only 140 meters. It should be noted that these distances are based upon having a clear sky background. Having a complex background, such as a cityscape, would drastically reduce the detection distance.

The FAA has also stated in their technical considerations report that electro-optical solutions should be secondary sensors due to their low detection ranges (Federal Aviation Administration, 2019).

2.1.3 Radar and LiDAR

Radar is a popular technology reliant upon radio waves to determine an object's range, angle, or velocity. The size of the object able to be detected depends upon the radio frequency used for detection. LiDAR uses lasers instead of radio waves and is typically used to survey a given landmass area. Both technologies send energy pulses and measure the time to receive a reflection back. Based upon this time, distance are then calculated using the speed of light. Distance for both radar and LiDAR can be calculated by

$$D = \frac{c * t}{2}, \quad (2.5)$$

where c is $3 * 10^8 m/s$ and t is time measured from pulse transmit to reflection received. Ullrich, Pfennigbauer, and Rieger (2013) can be referenced for a detailed overview of the various operational types of LiDAR.

C. J. Li and Ling (2017) investigated radar signatures of a DJI Phantom 2, a DJI Inspire 1, and a 3DR Solo using both 3-6 Gigahertz (GHz) and 12-15 GHz radar. The measurements were taken at short distances within 35 centimeters. However, this shows the ability to detect these UAS using the above stated frequencies. Of note, with all other variables controlled, using the lower frequencies showed a lowered maximum signal between 10 and 11 decibels (dB) when compared to using the 12-15 GHz frequencies. Drozdowicz et al. (2016) showed successful detection of the DJI Phantom 4 and S1000, using a 35 GHz radar. However, the environment was optimistic and appears to be a proof-of-concept experiment.

Skolnik (1980, p.4) defines the radar equation as such,

$$P_r = \frac{P_t G^2 \lambda^2 \sigma}{(4\pi^3) R^4} \quad (2.6)$$

where P_r is the reflected power received by the radar measured in watts, P_t is the power transmitted by the radar in watts, G is the radar antenna's gain in decibels (dB), λ is the wavelength used by the radar in meters, A_e is the effective capture area of the antenna measured in square meters, σ is the radar cross-section (the area radio waves can reflect off of on the target) measured in square meters, and R is the distance between the radar and the target in meters.

The maximum range of a radar can be determined by

$$R_{max} = \left(\frac{P_t A_e^2 \sigma}{4\pi \lambda^2 S_{min}} \right)^{\frac{1}{4}} \quad (2.7)$$

or

$$R_{max} = \left[\frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 S_{min}} \right]^{\frac{1}{4}}, \quad (2.8)$$

where S_{min} is the lowest detectable signal in watts. While these equations are the basis for the power needs of radar systems, Skolnik mentions that they are usually over optimistic by as much as double due to not accounting for loss. To counteract this, the equation can have a total loss parameter, L , added to the denominator. This loss includes internal attenuation, fluctuation losses during reflection, and atmospheric losses. This is difficult to quantify without measuring the environment directly. L is given in dB and the modified equation becomes

$$R_{max} = \left[\frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 S_{min} L} \right]^{\frac{1}{4}}, \quad (2.9)$$

LiDAR (also lidar or LIDAR) has been shown to be useful in UAS detection. There are two main types of LiDAR: scanning or flash LiDAR. Scanning LiDAR is the more traditional approach, whereby a single or dual laser apparatus is pulsed

against an angled, rotating mirror. This type of LiDAR can have a wide field of view (FOV), which is controlled by the rotation of the mirror. A flash LiDAR illuminates a given FOV and detects the photon reflections using a focal plane array. McManamon et al. (2017) provides an in-depth comparison of several flash LiDAR systems and discusses the energy needed for various degrees of photon detection and difference frequencies. A general reference to the many complexities of LiDAR is given by National Oceanic and Atmospheric Administration NOAA Coastal Services Center (2012). Rocadenbosch (2007) provides an expansive look at the calculations used to determine LiDAR performance. A simplistic form of the power detected by a LiDAR system is provided in McManamon and McManamon (2019) with

$$P(R) = KG(R)\beta(R)T(R) \quad (2.10)$$

where P is the power received from a distance R . K captures the overall performance of the system. G is the range-dependent measurement geometry. β is the backscatter coefficient, which is typically unknown until measured as it describes the ability of the atmosphere to reflect light back to the emitter. Finally, T is the transmission term, accounting for the loss of energy while traversing the distance R and back. K can be defined as

$$K = P_0 \frac{c\tau}{2} A\eta. \quad (2.11)$$

P_0 is the average power of a single laser pulse, and τ is the temporal pulse length in seconds. A is the area of the primary optical receiver, and η is the overall system efficiency. McManamon and McManamon (2019) also provides additional equations for the other terms, however they will not be provided here at this time given their complexity, and the decision to not include LiDAR in the SCANS Framework at this time.

Hammer, Borgmann, Hebel, and Arens (2019) developed a multi-sensor UAS detection and classification system using a 360° LiDAR sensor and camera sensor.

While deemed a successful method, the system only had an effective range of 35 meters. Laurenzis, Rebert, Schertzer, and Christnacher (2019) used a LiDAR system to track and make predictions as to a UAS' flight behavior, showing positive results for a range of 450-550 meters. From a military approach Kim, Khan, Choi, and Kim (2019) were able to show a double pan-tilt system capable of detecting UAS from 2000 meters with a FOV of 100°.

2.1.4 Radio Frequency

UAS commonly use the 2.4GHz and 5.8GHz radio frequency (RF) bands to communicate with their controller and to pass video information back to an operator. These frequencies are used because the communication relies upon Wi-Fi systems and they are unlicensed. Passive RF detectors can listen for these frequencies and alert CUAS systems of a possible breach. Passive RF detection is similar to acoustical detection in the sense that the emitter is the UAS itself, instead of the UAS acting as a reflector, such as with LiDAR or radar. If the radio signal strength from the UAS is known, then the potential Received Signal Strength (RSS) can be calculated using the Free Space Path Loss (FSPL) equation (Coleman & Westcott, 2018, pp.89-90).

FSPL is a theoretical equation used to calculate the loss in power of a radio signal for a given frequency between two points in free space. The FSPL equation is defined as

$$FSPL = 32.44 + 20\log_{10}(F) + 20\log_{10}(D) \quad (2.12)$$

where F is frequency in Megahertz (MHz) and D is distance between the target and the detector in kilometers.

RF detection is a common method employed in the commercial world due to the ease of use and setup. Farlik et al. (2019) used DeDrones' DroneTracker system (*Dedrone, Inc.*, n.d.) to successfully alert on a DJI MAVIC Pro, Phantom II, and a 3DR Y6 at 1400 meters. A major point of concern with the use of RF technology,

however, are the legal implications involved, driving the FAA and the U.S. Department of Homeland Security to suggest seeking legal advice before implementing any such system (Federal Aviation Administration, 2019; U.S. Department of Homeland Security, 2019).

2.1.5 Inverse-Square Law

The effective distance of each sensor type described above can be described using the Inverse-square law. The Inverse-square law states that a specified physical quantity is inversely proportional to the square of the distance from the source of that physical quantity. Phenomena affected by this law include electricity, magnetism, light, sound, and radiation. Mathematically speaking, the Inverse-square law can be written as

$$intensity \propto \frac{1}{distance^2} \quad (2.13)$$

for a single source, or

$$\frac{intensity_1}{intensity_2} = \frac{distance_2^2}{distance_1^2} \quad (2.14)$$

if comparing two sources or a source and receiver together.

2.2 Consumer UAS Specifications

UAS come in many shapes and sizes. While distinctions can lie between simple quad- or hex-copters and UAS, for the purposes of this study only the term “UAS” will be used. “UAS” is defined as a small, consumer-based rotary-wing or fixed-winged craft weighing less than 55 pounds (25 kilograms) per the FAA specifications in Part 107 (Federal Aviation Administration, 2018).

The consumer market has no shortage of drone companies. Popular brands such as DJI, Yuneec, Parrot, and Uvify make up a large portion of sales across the

U.S. These UAS typically weigh between 300 - 1500 grams and are capable of distances between 2 - 15 kilometers. Despite their small size, UAS such as the Phantom 3, are capable of lifting payloads upwards of 1 kilogram (Murray, 2016). Even small toy quad-copters have been shown to lift a substantial weight for their size (Fungineers, 2016). Moving past typical consumer UAS, the unique classification of “heavy-lifting” UAS specify devices capable of lifting payloads up to 12 kilograms.

Table 2.1 lists a variety of commercial UAS and their relevant specifications such as size, weight, flight range, and speed. All information is found on either the manufacture’s specification sheet, or promotional information distributed by the manufacture. Data that was unable to be located is denoted with an en dash. The selected UAS were chosen based upon buying guides and popularity from online retailers and technology blogs. The intention was to gather information on UAS consumers are likely to buy. In addition to the consumer-level UAS, several popular heavy-lifting UAS were included, denoted in italics. Finally, a “racing” UAS, the Uvify Draco, specifically designed to achieve high speeds, was included as the powerful motors allow for a large payload based upon the motor’s performance specification. The table is organized by manufacture, then model, with the heavy-lifting UAS listed at the bottom of each manufacture.

To categorize the UAS into more comparable groups, the *DJI Spark*, *Parrot Bebop*, and *Yuneec Breeze* are very small camera UAS designed for novices costing around \$300 to \$600. The *DJI Mavic* line, *Parrot Anafi*, and the *Autel Robotics Evo* are for “prosumer” customers, who are categorized between the average consumer and the enthusiast or professional, but make up the largest sector of the market of consumer drone sales. These UAS typically cost between \$1000 - \$2000.

As mentioned earlier, the *Uvify Draco* is designed for racing. This UAS is small and made for speeds of 160 kilometers per hour (100 miles per hour). To achieve these speeds, powerful motors are needed, which has the unintended benefit of being able to lift relatively heavy objects. Each of the above mentioned UAS are

Table 2.1.

UAS Specification Comparison

Brand / Model	Size (mm) (LxWxH)	Weight (g)	Max Range (km)	Control Distance (km)	Max Speed (m/s)
Align					
<i>M690L</i>	900x900x362	3400	17	2	33.5
Autel Robotics					
Evo	254x127x127	863	18	7	20
DJI					
Inspire 1	437x302x453	3060	11.8	5	22
Inspire 2	427x317x425	3440	21	7	26
Mavic Air	168x184x64	430	10	4	19
Mavic Pro	83x83x198 (folded)	734	13	6.88	18
Mavic 2 Pro	322x242x84	907	18	8	20
Phantom 4	221x381x325	1380	16.8	5	20
Spark	143x143x55	300	2	0.1	14
<i>Agras MG-1</i>	1471x1471x482	12500	2.4	1	8
<i>Matrice 100</i>	650x650x—	2355	14.5	5	22
<i>Matrice 600</i>	1668x1518x759	9100	8.1	5	18
<i>S900</i>	900x900x130	3300	7.2	—	16
<i>S1000</i>	1045x1045x130	4200	7.2	—	16
FreeFly					
<i>Alta 8</i>	1325x1325x253	6200	7	7	15.6
Parrot					
Anafi	175x239x63	320	11.25	4	15
Bebop	328x328x88	500	0.3	0.3	16
Uvify					
Draco	214x166x63	692	2.68	2.68	44.7
Yuneec					
Breeze	196x196x65	385	1.8	0.007	5
H520	520x455x295	1633	12.75	1.6	17
Typhoon H Plus	480x425x295	1995	11.25	1.6	13.4
<i>Tornado H920</i>	920x920x461	890	10	1.6	14

quad-rotor devices. The *DJI Inspire* line, the *Yuneec Typhoon*, and the *H520* are UAS capable of speeds useful for filming moving vehicles. Finally, the heavy-lifting UAS include the *DJI Matrice* series, a professional UAS capable of carrying a variety of payloads; the *S* series, a customizable bare-bones development system; the *Yuneec Tornado H920* and *Align M690L* are comparable systems to the *DJI Matrice* series; and finally, the *FreeFly Alta 8* and *DJI Agras MG-1* are both examples of high-end UAS designed for the professional market. Both are capable of payloads above 10 kilograms and cost between \$15000 and \$17000.

While there have been several demonstrations to show the lack of counter-UAS security (De Clercq, 2018; Gallagher, 2013; Schmidt & Shear, 2015), small UAS have also been used in actual attacks against government officials (Koettl & Marcolini, 2018) and in Syria (Hennigan, 2017). The attacks in Syria involved small quad-copters, or crude fixed-wing UAS, flying over ground forces and dropping munitions on them. The use of UAS on the battlefield has led the U.S. military to invest in counter technologies such as nets, radio frequency jammers, and lasers, an aggregated list of which can be found in Michel (2018). In addition to these attacks, the Federal Emergency Management Agency (FEMA), the Department of Homeland Security (DHS), and independent researchers have published studies and reports as to the effects of explosives, devices which these UAS are known to carry in varying environments as well as potential damage to the human body (Federal Emergency Management Agency, 1995; U.S. Department of Homeland Security, n.d.; Zipf, Kenneth, & Cashdollar, 2010). Based upon the average speed data presented in Table 2.1 (18 m/s), a rogue UAS could cover an expanse of 500 meters in only 27.7 seconds, leaving little time to react to a threat.

2.3 Communication Networks

While sensors are necessary for detection, communication networks are necessary to pass information from those sensors to a communication hub where

further processing can take place. There are several variables to consider when designing a wireless network and choosing a communication technology. These include the intended use case, frequency restrictions, throughput and bandwidth requirements, user capacity, power usage, and transmission distance. The environment in which a CUAS system is deployed makes a difference for the recommended wireless network technology. If transmissions must travel through vegetation, a sub-gigahertz frequency with high robustness, such as LoRaWAN, may be ideal; whereas, if within a city, cellular technologies or Wi-Fi may be better suited due to the speed of packet retransmission and advanced signal processing techniques.

2.3.1 Physical Wireless Network Characteristics

The following outlines technology characteristics of potential communication networks based upon an Internet of Things (IoT) or sensor network mindset.

LoRaWAN

Long Range Wide-Area Network, more commonly referred to as LoRaWAN, is managed by the LoRa Alliance. Created in 2015, LoRaWAN relies on the proprietary underlying radio technology LoRa (LoRa Alliance, 2015). The radio technology LoRa uses Chirp Spread Spectrum (CSS) modulation allowing it to transmit over large distances through heavy-noise environments (LoRa Alliance, 2017). LoRa radios operate at either 433 MHz in Asia, 868 MHz in Europe, or 915 MHz in North America.

LoRaWAN is designed for IoT applications, and as a result, varying configuration options exists to enable either faster throughput, longer battery life, greater communication distance, or higher network capacity in terms of usable clients (LoRa Alliance, 2015). The variance in performance comes from adjusting the spreading factor of the CSS modulation, the bandwidth allotted to

communication, and the coding rate used for sending the data. Along with these variances, LoRaWAN allows for different classes of nodes which relate to when the node sends and receives data.

Class A nodes are best for long-term deployments and only allow for data to be received shortly after data is sent. This allows for packet acknowledgments.

Class B devices open a receive window at set intervals, allowing for communication from the gateway to occur even if information hasn't been sent. This is optimal for situations needing low-priority status updates. *Class C* devices only close the receive window when transmitting data; as a result, these devices use the most power and are most akin in functionality to other technologies (LoRa Alliance, 2017).

LoRa radios have a remarkable link budget stemming from a low received signal sensitivity of -136 decibels (dB) (*Semtech SX1276*, n.d.) and several studies have measured their performance. Augustin, Yi, Clausen, and Townsley (2016) provides a detailed overview of LoRaWAN performance in an urban setting. The study showed communications were reliable at distances up to 3 kilometers using the maximum spreading factor, but also noted that issues arose with communication performance if network traffic became too dense. They concluded that LoRaWAN is best for low-power, low-throughput, and long-range networks. Cattani, Boano, and Römer (2017) found that as temperatures increase to 50°C, communications using LoRa radios cease. Conversely, Riegsecker (2018) found that as temperatures approached 0°C, communication performance suffered, but as temperatures dropped below freezing communication started to improve once more. Yim et al. (2018) found inconclusive results related to LoRa performance in a tree farm, possibly due a blocking of the Fresnel Zone. Petajajarvi, Mikhaylov, Roivainen, Hanninen, and Pettissalo (2015) measured LoRa performance while driving in a car and over water in a boat. From the car, an average packet loss ratio of 13.5% was seen at distances less than 5 kilometers, and a packet loss ratio of 74% was seen at distances between 10 and 15 kilometers. From the boat, an average packet loss ratio of 32% was seen at distances between 5 and 30 kilometers. Of note, while all of above-mentioned

studies measured the true LoRa radio performance, each made reference that performance would only increase given the ability of LoraWAN to manage retransmitted packets and dynamically adjust transmission parameters of clients.

LTE-M

Long Term Evolution category M1 (LTE-M) is the communication method of low complexity user equipment (UE) for machine-type communication (Rohde & Schwarz, n.d.). First defined in Release 13 from 3GPP, LTE-M supports multiple access multiplexing, allowing multiple clients to simultaneously communicate with the base station. LTE-M allows for one megabit per second throughput and has a bandwidth of 1.08 MHz (Ray, 2017). LTE-M was designed to have a ten year battery life using a five watt-hour battery, have a low cost, and reach seven times further than current cellular networks (Vos, n.d.). Currently, LTE-M has been deployed throughout the U.S., Canada, Brazil, Western Europe, Australia, and several countries in Asia.

In terms of characteristics, LTE-M uses current 3G and 4G cellular networks and their technologies. Differences lie within the narrow bandwidth, the low sensitivity of the receivers, and the strong power signal provided by LTE-M modules (Vos, n.d.). This allows for signals to be received underground with 20-30 dB of attenuation and a 95% received packet ratio. Outdoors, LTE-M obtains a a 99% received packet ratio with less than 10 dB of attenuation (Lauridsen, Kovacs, Mogensen, Sorensen, & Holst, 2016). Vos (n.d.) shows LTE-M coverage to be between 100 meters and 10 kilometers. Dawaliby, Bradai, and Pousset (2017) provides an in-depth analysis of LTE-M performance and have concluded that while LTE-M provides better coverage, latency, and jitter metrics than CAT-0 connections, at the time of publication, it had yet to match the prescribed performance metrics described by 3GPP. Dawaliby et al. suggest protocol modifications to reduce bandwidth further, thereby allowing more IoT devices onto

a network. Typical use cases for LTE-M include resource monitoring for energy or water, agricultural sensing, smart city applications, patient monitoring in hospitals, and wearable devices, among others (Vos, n.d.).

IEEE 802.11n-2009

The IEEE 802.11 family of protocols has become ingrained in the daily lives of people everywhere. Due to extensive testing and design work, the protocols themselves are extremely robust and provide high data throughputs and reliability. A major impact to this family of technologies was the IEEE 802.11n-2009 (802.11n) standard (Coleman & Westcott, 2018).

The IEEE 802.11 protocols, known commonly as Wi-Fi, operate in the free, unlicensed Industrial, Scientific, and Medical (ISM) and unlicensed national information infrastructure (U-NII) bands. From these frequency bands, the 2.4 GHz and 5 GHz are used for Wi-Fi and are shared with Bluetooth, microwave ovens, and UAS control systems. Being free and unlicensed bands means frequencies are often overcrowded and signal interference is great. 802.11n improved on previous Wi-Fi technologies in several ways. First, throughput was increased in both the 2.4 GHz and 5 GHz bands; second, 802.11n capitalized on the phenomenon known as multi-path instead of being hindered as other technologies were. This allowed 802.11n to increase signal range (Coleman & Westcott, 2018).

In accordance with all U.S. regulations set forth by the Federal Communications Commission (FCC), 47 CFR 15, the 2.4 GHz frequency is capable of transmitting up to 160 kilometers. This, however, is unrealistic due to the need of accounting for the curvature of the Earth. Realistic distances are upwards of 32 kilometers (20 miles). However, even at these relatively closer distances, 2.4 GHz signals need to remain mostly line-of-sight due to rapid link deterioration when obstructed by an obstacle, such as a tree.

5G Cellular

5G, or the fifth generation of cellular communication, has received a lot of excitement since its initial release in 2018. It has even generated a few marketing legal battles related to branding (Clover, 2019). While 5G is built upon current 4G fundamentals, the vast expansion of frequencies available for use, along with multiple modes and module virtual radio functions, enables a large selection of use cases. These include pervasive video, high-speed mobility, sensor networks, tactile internet, lifeline communications in emergencies, e-health services and broadcast services (Next Generation Mobile Networks Alliance 5G Initiative, 2015). 5G should also support “several tens of [megabits per second] ... for tens of thousands of users in crowded areas, such as stadiums or open-air festivals” (Next Generation Mobile Networks Alliance 5G Initiative, 2015, p. 28).

Lauridsen, Gimenez, Rodriguez, Sorensen, and Mogensen (2017) took a precursory look at expected limitations of 5G technology based upon current LTE technology. From their study they identified that in order to achieve the requisite time for hand-offs, latency, and execution time major advancements would need to be made regarding user and control plane efficiency. Said differently, the radio access network is the largest contention area for delays. Solomitckii et al. (2018) proposed a plan to use 5G itself for small UAS detection given the proliferation of base stations and available bandwidth. A system such as this could help reduce overall communication delay if the sensor network is made up of 5G base stations.

2.3.2 Wireless Network Performance Based Upon Environment

As stated earlier, the medium in which wireless signals propagate affects network performance. The following environments were chosen based upon their diversity and use cases within the CUAS space. Each environment provides a unique challenge to wireless network communication.

Sports Stadiums

Sports stadiums are troublesome for wireless communication networks. Holding between several thousand to one-hundred thousand people, stadiums create harsh RF environments given their highly reflective nature and large amounts of attenuation due to patrons (Klepal, Mathur, McGibney, & Pesch, n.d.; Mathur, Klepal, McGibney, & Pesch, n.d.). Young et al. (2010) measured RF propagation of the Philadelphia stadium before demolition. Tests were performed for frequencies between 49 MHz and 1830 MHz. The study found an increase in the median received power and a decrease in standard deviation as frequency increased. Young et al. (2014), work performed by the same team, found that given a similar environment and equivalent power, received power is negatively correlated with the frequency used. However, each tested frequency, between 430 MHz and 2400 MHz, acted similarly; the 4900 MHz signal showed a faster signal decay rate. This provides evidence that frequencies between 49 MHz and 2400 MHz are better suited for stadium use. Frequencies higher than 2400 MHz appear not to propagate well enough to be useful. Finally, Remley, Koepke, Holloway, Camell, and Grosvenor (2009) describes radio propagation in harsh environments. While the study looked at an oil refinery and a manufacturing plant, the RF-reflective building materials highlight communication issues within stadiums, which are also reflect RF signals strongly.

Urban Canyon

Urban canyons are streets with high rises or skyscrapers bordering them, creating a corridor for radio waves to travel down. While not many exists, some examples include the Magnificent Mile in Chicago, the Financial District in Toronto, the Canyon of Heroes in Manhattan, and the Central and Kowloon districts in Hong Kong. These areas typically hold parades and celebrations where recreational UAS

pilots will fly to capture video or pictures of events. This creates a public safety issue and is a candidate for UAS detection-interdiction systems to be deployed.

In terms of characteristics of urban canyons, Matolak et al. (2014) performed measurements for 700 MHz and 4900 MHz, both public safety frequencies, and noted wave-guiding along the buildings occurring for the 4900 MHz frequency. This is in line with expectations as higher frequencies tend to reflect, rather than penetrate, an object. Holloway, Koepke, Camell, Young, and Remley (2014) noted the effects of buildings on RF propagation with measurements taken before, during, and after several building collapses. Finally, Haneda et al. (2016); Lu, Bertoni, Remley, Young, and Ladbury (2014) have created models for describing path loss specifically in urban and urban canyon environments. All models described were then validated with empirical studies.

Forests

All U.S. national forests, and many private tree farms, restrict UAS flight with few exceptions (U.S. National Park Service, n.d.). Forests, or wooded areas in general, cover large expanses of land and some are protected by the National Park Service for their beauty and role in the ecosystem. For this reason, the National Park Service wants to minimize the amount of radio towers used within parks, opting for technologies that travel longer distances. As trees and foliage are a massive hindrance on RF propagation, this creates a unique use case for radio transmissions. As the best radio reception is had with no obstructions, radio towers could be built above tree lines. However, with trees capable of growing between 20 - 300+ feet in natural settings, this is impractical. Having towers above the treeline would also make it difficult to detect UAS flying below the treeline. For this reason, the following studies will focus on radio propagation through forested areas.

Jamrogowicz (n.d.) shows an average attenuation of 11 dB across 14 tree species, with a maximum attenuation of 20 dB at the 1600 MHz frequency.

Ghoraishi, Takada, and Imai (2016) performed a detailed analysis of radio propagation through forested areas noting the complex multi-path scenario within dense forests. Their work describes radio waves breaking into three distinct clusters, thereby causing receive delays and attenuations. The study used 2.2 GHz. Park et al. (2018) and Yim et al. (2018) provide measurements and use of LoRa radios for monitoring purposes, operated at 915 MHz, in tree farm operations giving possible credence to the use of LPWANs. Ahmad, Salleh, Segaran, and Hashim (2018) tested propagation in a rubber tree farm and a “light tropical jungle”. Using a spreading factor of 7 and 12 at 433 MHz, the researchers showed reliable transmissions with the antennas less than three meters off the ground. Specifically, the researchers recorded signal strengths of -101 dB at approximately 720 meters using a spreading factor of 12. With the antennas at three meters above ground level, this would have approximately a 36% Fresnel Zone blockage. While within recommended bounds, if the antennas could be placed even a few meters higher, it is possible the received signal would have improved.

Open Field

Open fields provide one of the most idealistic environments for wireless communication. Provided a transceiver can be mounted a significant height above the ground, any frequency could be used to communicate. Assuming no-blockage of the Fresnel Zone, wireless communication performance should be evaluated on the maximum distance possible, given any frequency and the transmit power necessary to do so.

2.4 Current CUAS Simulation Models

Modeling a complete CUAS system is complex due to the systems-on-systems design, and therefore a choice is made to model smaller pieces instead of the whole. Cline and Dietz (2020) used AnyLogic to model and evaluate

the critical threat speed of a UAS, whereby the ability of a counter system to detect and interdict becomes prone to failure. Aside from speed, all other aspects of the UAS and counter system interaction remained static. White, Shin, and Tsourdos (2011) provided simulated results for UAS obstacle avoidance. This system was not based upon counter tactics or detection, but rather the UAS functionality itself. Wagoner, Schrader, and Matson (2017) focused on UAS interdiction, assuming detection and basic tracking were already accomplished. Wagoner used Gazebo and the Robot Operating System (ROS) messaging protocol to develop a monocular hunter-drone counter interdiction system utilizing a man-in-the-loop system. The system operated with a human operator manually aligning the hunter drone before switching to autonomous mode for a kinetic interdiction. S. Li (2019) made use of NetLogo software to model a tracking algorithm making use of trilateration. A limitation of the study, however was the assumption that a sensor would detect a UAS at a specified distance all of the time.

2.5 Summary

This section provided a review of sensors, UAS, network communication technologies, and current CUAS simulation models common to CUAS research from around the world. Current research focuses on either a specific sensor type, or efforts are spent developing a method for a later phase of the detect, track, interdict CUAS model. This has left little research in the way of generalized sensor performance modeling, as well as the communications network necessary to connect those sensors to a control hub for processing and escalating a UAS threat. Chapter 3 proposes and defines a novel framework for such a system, including system inputs, outputs, functionality, and internal model processing.

CHAPTER 3. FRAMEWORK DESCRIPTION AND METHODOLOGY

There is no singular way to model a complex system such as a CUAS simulation. However, methods exist that can aid in discerning a system with value. The organizational framework AEIOU (Martin & Hanington, 2012, pp. 10-11) can be used as a means of documenting various aspects of the simulation's functionality. AEIOU stands for Activities, Environments, Interactions, Objects, and Users. It is typically associated with coding user interactions via observations, however it can work in a modified state for this research by ensuring interactions between agents are adequately developed. From the AEIOU documentation, scenarios (Martin & Hanington, 2012, pp. 152-153) will be developed to test the functionality and robustness of the model. By the end, the simulation model itself will allow end-users to design using a flexible modeling (Martin & Hanington, 2012, pp. 88-89) approach, thereby creating a rapid prototyping environment.

The focus of this work is to design, implement, and evaluate a new model framework, called the SCANS Framework, for CUAS system performance evaluation. In order to determine what should be included in such a framework, the AEIOU framework, along with the information presented in Chapter 2, has been used in an attempt to encompass the CUAS problem space. From this, an iterative process is used to further model and refine system interactions before the model is implemented in AnyLogic. AnyLogic is built upon the Java programming language, however the overall SCANS Framework is system agnostic.

The methodology for testing and validation of the model is provided at the end of this chapter. Validation relies upon the implemented framework's ability to replicate various experiments, whereby comparing the performance results of the virtual experiment to the original experiment. This chapter first describes the AEIOU process, followed by the SCANS Framework outline. The outline consists of

input parameters, output data, and model and agent functions and variables. The section concludes by describing the process used for evaluating the SCANS Framework.

3.1 AEIOU

Traditionally, AEIOU would be used to document activities as they occur naturally, however it can also be used on an imaginary scenario as well. Imagine a facility, such as a prison that requires its borders to be protected against contraband being flown in via a UAS air drop. Cline and Dietz (2020) has modeled such a facility and Sathyamoorthy (2015) highlighted several cases wherein UAS were used to deliver illicit drugs to a prison facility. The following is an analysis of the activities, environments, interactions, objects, and users of the scenario in an attempt to determine what inputs for the model are required. The overall goal of the proposed framework is to determine if a UAS can be detected given various combinations of sensors. The activities of such a system are few and straight forward.

One or more UAS partake in flying a mission. This activity involves the UAS pilot setting up the UAS, the UAS taking off and flying to a waypoint, and may or may not involve the UAS returning back to the pilot depending on the mission. Any given sensor has to measure the environment, with the potential for rotating in place, and must send its data onward to a collection point. Communication nodes act as aggregation points for the sensors to send their data and must forward the traffic to either another communication node or a CUAS command and control hub (CC) for further action to take place. The CC must record any data received in order for further counter action to take place. Any action above this point is out of the current scope for this research.

Due to time constraints, and the complexity involved with modeling and evaluating a system with numerous environmental obstacles, the SCANS Framework

will initially only consider a CUAS environment in an open field. This means any and all sensors and communication hubs have clear line of sight to each other. However, environmental weather factors such as high wind and poor visibility can still be considered in an error rate parameter explained in Section 3.2.

Interactions between agents, those being the UAS, sensors, communication nodes, and command and control hubs, are also straightforward in scope. The UAS flies overhead of the sensors on the ground. Noise generated by the spinning rotors, and radio communication signals for receiving commands are measured by different sensors. The UAS may also give off a heat signature from the computer and mechanical components allowing for the use of thermal cameras to visualize the UAS. The sensors, regardless of type, must measure an output of the drone relative to itself, typically dependent upon distance. If a sensor measures a significant signal, it should send an alert via the communication nodes to the CC. As the ultimate goal is for the data to be received at a CC, the communication nodes must forward the data either to other communication nodes or to the CC. This also dictates the communication nodes and CC be capable of receiving said data. Finally, the CC, having received the data, should log any data received so further tracking or interdiction action can be taken.

Objects for a given scenario include the UAS pilot, the UAS, numerous sensors of varying type (acoustic, passive RF, electro-optical, etc.), communication devices, CC(s), the CC operators, response teams, and interdiction systems, if applicable. For the purposes of this framework, the UAS pilot will be disregarded as the main focus is the UAS itself, also anything beyond the data collection stage at the CC will be disregarded as that lies beyond the sensor focus view of this research. From this preliminary analysis, Figure 3.1 is provided to show the general flow of information within this framework between agents.

The last piece of the AEIOU framework are the users involved. As the UAS pilot and various individuals enrolled in the interdiction phase of the project have been disregarded, it would be most beneficial to shift focus to the users of the



Figure 3.1. Framework Communication Chain

SCANS Framework to ensure their needs are met. As this framework is meant to assist in the evaluation and design of CUAS systems, a “CUAS system designer” is considered to be the primary user. These users may be researchers or professionals with a desire to design experiments, test various sensor configurations, define variables, or ascertain a general expectation of system behavior. A secondary user may be an individual using the pre-built framework for the purpose of cost analysis or system optimization.

3.2 Input Parameters

The next step in describing the SCANS Framework is to define the input parameters. Outputs and Functions are discussed later sections. In accordance with the AEIOU analysis and the information presented in Chapter 2, the general inputs for the system are given in Table 3.1, and the specific inputs for agents are discussed later in this chapter. Most inputs for the UAS are derived from the various needs of the sensors.

The *Maximum Sound Pressure Level* (SPL), for example, is the maximum volume of noise the UAS generates during flight. Typically, this noise is from the whirling of the rotors attached to the motors. The *Maximum SPL distance* specifies the distance at which the *Maximum SPL* value was calculated. As a SPL is only relevant at the point of measure, the distance at which the SPL was measured is needed in tandem with the intensity formula (Equation 2.1) to calculate the SPL at a new distance. As a default value, the *Maximum SPL* is 88.5 dB and the *Maximum*

SPL distance is 1 meter. These values are based upon measurements in Cabell, McSwain, and Grosveld (2016) for the DJI Phantom 2. The measurements of Cabell et al. were also used in Torija, Li, and Self (2020).

The radar sensor requires the cross sectional area of the UAS, provided in squared meters. Farlik et al. (2019) performed several measurements of consumer UAS with various sensor types. For the radar sensor type, the measurements found that a DJI Phantom 2 had the greatest cross section of 0.26, when using a 8600 MHz frequency, based upon a computer generated model using a software called SuperNEC SW. Based upon real-world measurements using a 10 GHz radar, the highest recorded radar cross section (RCS) was 0.35 m². C. J. Li and Ling (2017) also measured the DJI Phantom 2 using a 12-15 GHz radar and found the cross section area to be approximately 0.12 m² at the strongest measure.

The passive RF sensors require the various radio parameters to be provided including the transmission frequency, power, and antenna gain. As a UAS commonly makes use of both 2.4 GHz and 5.0 GHz channel frequencies, two sets of frequency, power, and antenna gain are added as input parameters.

In regards to the sensors, Table 3.2 highlights the specific, unique inputs required by a given sensor, while Table 3.1 provides the common inputs. Common inputs include items such as rotation speed, field of view, whether the sensor is unidirectional, and a sensor's polling rate. In terms of operation, the communication node agent and the CC agent are designed as radio communicators and differ only in how they process the data they are given. This causes input parameters for both to be similar.

Currently, both the communication node and the CC are considered wireless communication devices only, as wired communication would involve building more advanced routing protocols into the framework, which at this time is out of project scope. However, wired communication can be simulated. By setting the *Frequency* input to be the same on all nodes, the *Transmit Power* and *Antenna Gain* to 999, and the *Dropped Packet Ratio* (DPR) to 0, communication could be guaranteed for

Table 3.1.

Determined Input Parameters Using AEIOU Analysis

UAS	Sensor	Communication Node	Command and Control
Maximum Speed	Rotates	Noise Floor	Noise Floor
Maximum Sound Pressure Level (SPL)	Error Rate	Frequency	Frequency
Maximum SPL Distance	Rotation Speed	Transmit Power	Transmit Power
Radar Cross Section (RCS)	Field of View	Antenna Gain	Antenna Gain
Radio Frequency (1 & 2)	Polling Rate	Dropped Packet Ratio	Dropped Packet Ratio
Radio Transmit Power (1 & 2)	Rotation Offset	Sensitivity	Sensitivity
Radio Antenna Gain (1 & 2)			

almost any distance. At that point the current mechanism for route decision could be used to define a path of communication, however this would result in a system with no network hops. For this application, however, wired communication is not implemented due to the modeled distances between sensors and communication nodes, as well as non-necessity for the tested use cases.

Because of the commonalities between sensor types and the commonalities between the communication devices, it is beneficial to have a general *Sensor* object class and a *CommunicationDevice* object class from which the different sensor types and the communication devices inherit from. The envisioned object-oriented class inheritance structure is provided in Figure 3.2.

Table 3.2.

Specific Additional Sensor Inputs

Acoustic	Camera	Radar	Passive Radio Frequency
Noise Floor	Resolution Height	Transmit Power	Frequency
	Resolution Width	Minimum Receive Power	Antenna Gain
	Pixels Per Foot	Frequency	Sensitivity
	Lens Angle	Antenna Gain	
		System Loss	

Given the hierarchical structure of the framework, there are multiple common parameters present in each agent. The first parameter, *Error Rate*, acts as a buffer for the unknown, or difficult to compute scenarios, and does not have a necessarily rigid definition. For example, when applied to an acoustic sensor, a higher probability may be used to indicate a windier day; a radar may use it for a misidentification of a bird; or the passive RF sensor may use it as a sudden noise floor spike. The communication devices – *CommunicationNode* and

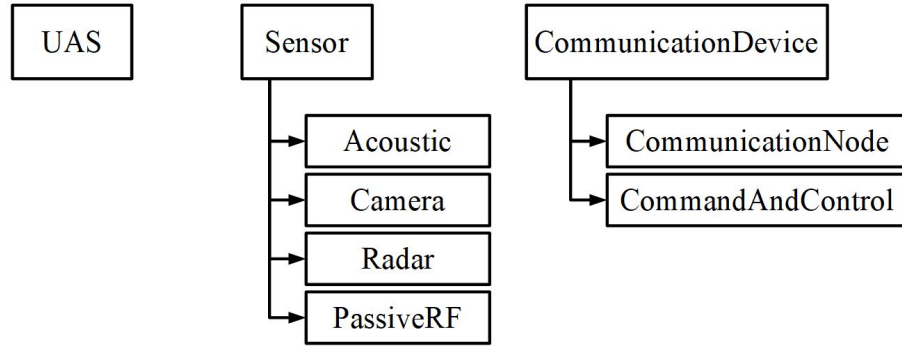


Figure 3.2. Object Class and Inheritance Structure

CommandAndControl – make use of a “Dropped Packet Ratio” (DPR) for semantic reasons, but the purpose is the same as an error rate.

Rotates, *Rotation Speed*, *Field of View*, and *Rotation Offset* govern the necessary attributes to create rotating agents. *Rotates* is a Boolean value used to define whether an agent will be considered omnidirectional or unidirectional and, in this particular implementation, defines changes in an agent’s graphical presentation (i.e. when *true* a directional indicator is added to the agent). *Rotation Speed* defines how quickly an agent rotates and is dependent on *Rotates* being *true*. The model currently implements this rotation functionality as a function of the *Polling Rate*. Due to this chosen implementation, *Rotation Speed* is given as a degree delta by which to rotate the agent. For example, a *Rotation Speed* of 60 would rotate the agent by 60° at every elapse of *Polling Rate*. An issue that arises from using these stated methods of rotating an agent is the inability to face an agent in a specified direction. Upon initialization, AnyLogic places all agents with a rotation angle of zero degrees (facing towards the positive X axis); *Rotation Offset*, used in accordance with *Rotates*, is used to offset this initial zero degree rotation. A 90° offset will face the agent toward the positive Y axis. During implementation it should be noted that AnyLogic’s implementation of the coordinate plane system requires these coordinates to be adjusted. In order for an agent to stay stationary at *Rotation Offset*’s value, *Rotation Speed* should be set to zero.

Finally, *Polling Rate* defines the time delay necessary between querying the sensor value. For this framework, this rate is used as the general speed delimiter for the sensor by providing a rate limiter for sensor rotation and a rate limiter for the amount of data generated. Most sensor types have been given a *Polling Rate* of 250 milliseconds ± 10 , with the exception of the *Camera* sensor type, which was set to 500 milliseconds ± 10 . The ± 10 is necessary due to a limitation of the communication devices not having a data packet queue, and acts to limit data collisions. Data collision mechanisms would be an improvement on the system, but are not currently implemented.

As mentioned before, Table 3.4 outlines the common sensor inputs. Tables 3.3 - 3.9 provide each parameter for the UAS, sensors, communication nodes, and CC; the data type; and the default value. The *Acoustic* sensor has a single additional parameter, *noiseFloor*, which measures the environmental noise level in decibels (dB). The *noiseFloor* parameter is given a default value of 30, which is considered to be similar to the noise level of a quiet room. The default value is a subjective decision and will change based upon use-case, but this parameter, and every other parameter's default value, is chosen as a reasonable starting point based upon the work presented in Chapter 2.

Much thought has gone into whether the *noiseFloor* parameter should be only a part of the *Acoustic* sensor agent, or if it should be applied to every other sensor type, as well. In general, every sensor, regardless of type, has a lower limit of what can be measured due to environmental noise. Measurements below this threshold may be impossible due to the electrical noise present in a radar, the traffic noise on a freeway (in the case of a microphone), or electromagnetic waves in the case of radio communication. The noise floor is usually independent of the sensor, and dependent on a specific location and time. However, while implementing the framework, the noise floor only seemed applicable to acoustic sensors and radio communication. Radar may have an applicable noise floor, however it was not apparent from the literature that such a metric should be considered.

Table 3.3.

UAS Parameter Inputs

Parameter	Datatype	Default Value	Description
maxSpeed	Double	25	Maximum speed of a DJI Phantom 2 in m/s
maxSoundPressureLevel	Double	88.5	Max SPL generated by a Phantom 2 in dB
maxSPLDistance	Double	1	Distance maxSPL was measured from in m
radarCrossSection	Double	0.26	Measured in m ²
radioFrequency1	Double	2400	UAS control frequency 1 measured in MHz
transmitPower1	Double	20	UAS radio 1 transmit power measured in dBm
antennaGain1	Double	2.5	Gain for a 2.4 GHz dipole antenna measured in dB
radioFrequency2	Double	5800	UAS control frequency 2 measured in MHz
transmitPower2	Double	20	UAS radio 2 transmit power measured in dBm
antennaGain2	Double	3.2	Gain for a 5.8 GHz dipole antenna measured in dB

Electro-optical sensors, which includes visible spectrum and infrared cameras, also do not make use of a noise floor due to the closest comparison for the parameter being camera focus. For these reasons, it was determined that *noiseFloor* should not be applied as a global parameter. Later studies may find this to be an errant

Table 3.4.

General and Acoustic Sensor Parameter Inputs

Parameter	Datatype	Default Value	Description
rotates	Boolean	false	If true, sensor is considered unidirectional. If false, omnidirectional
rotationSpeed	Double	30	Step value of rotation in degrees
rotationOffset	Double	0	Beginning rotational offset from 0 degrees
fieldOfView	Double	30	Angle of view for a sensor in degrees
pollingRate	Double	250	Delay between sensor calculations in ms
errorRate	Double	0.05	Definition changes per sensor type. Percentage range of 0.0 - 1.0

choice. A comparable parameter to a noise floor is a sensor's sensitivity, however the two are not the same. In general, the sensitivity of a sensor marks the lowest measurement possible if the noise floor was not present or defines the amount of received power necessary to record a chosen voltage.

The *Camera* class, representing electro-optical sensors, has a redundancy for the *fieldOfView* (FOV) parameter due to also specifying the *resolutionWidth* and the *pixelsPerFoot* parameters. FOV can be calculated by dividing the latter two parameters together, as in Equation 2.2. The use of either set of parameters are left to personal choice, however, care should be taken to specify all applicable parameters as error checking is not currently implemented.

Radar and LiDAR systems are difficult to model due to the typical lack of system specifications by manufactures. It is for this reason, along with the number of inputs necessary to have a meaningful distance calculation, other than maximum

Table 3.5.

Camera Sensor Parameter Inputs

Parameter	Datatype	Default Value	Description
resolutionHeight	Double	480	Resolution height of the camera picture in px
resolutionWidth	Double	720	Resolution width of the camera picture in px
pixelsPerFoot	Double	4	Number of pixels per foot required for object detection in px
lensAngle	Double	30	Lens angle of the camera in degrees

range, LiDAR systems are not implemented in the SCANS Framework at this time. Radar systems, while having slightly more available information, still appears lacking in terms of desired performance and usability for this framework. However, based upon the studies done by Farlik et al. (2019) and C. J. Li and Ling (2017), among others, information can be acquired in order to obtain range calculations. The *Radar* agent assumes data availability via a systems designer actively seeking to implement radar technology into a CUAS solution. The default values presented in Table 3.6 are chosen from within ranges cited in other studies but are, for all intents and purposes, arbitrary and hold no significance.

The many parameters proposed by this framework may lower usability simply due to the amount information needed. For comparison, Table 3.8 provides the sensor parameter recommendations from the U.S. Department of Homeland Security (DHS) (U.S. Department of Homeland Security, 2019, pp.20-22). These parameters have the benefit of being uniform across sensor types, are few and simplistic, and are more in-line with what other researchers, such as Cline and Dietz (2020) and S. Li (2019), have used. Yet, while obtaining the information for these parameters is

Table 3.6.

Radar Sensor Parameter Inputs

Parameter	Datatype	Default Value	Description
transmitPower	Double	1000	Power of the radar in W
minimumReceivedPower	Double	0.000000001	Smallest energy signal detectable in W (-90dBW)
frequency	Double	3.5	Radio frequency in GHz
antennaGain	Double	36	Gain for the antenna measured in dB
systemLoss	Double	1	Entire power loss of the system measured in dB

Table 3.7.

PassiveRF Sensor Parameter Inputs

Parameter	Datatype	Default Value	Description
sensitivity	Double	-80	Smallest energy signal detectable in dB
frequency	Double	2400	Radio frequency in MHz
antennaGain	Double	2.5	Gain for a 2.4 GHz dipole antenna measured in dB

usually readily available from manufactures, the measures provided may be inflated, which is precisely the problem the SCANS Framework aims to combat.

Now, more than ever, there are a variety of communication technologies built for sensor communication, Internet of Things (IoT) applications, and low data applications. A few of these were described in Section 2.3 and have similar inputs regardless of the technology. Differences between technologies such as Wi-Fi and LoRaWAN come in the the form of modulation schemes, frequency usage, and

Table 3.8.
DHS suggested sensor parameters

Acoustic	EO/IR	Radar	RF
Detection Range	Detection Range	Detection Range	Detection Range
Direction Finding	FOV	Field of Regard	Radio Frequencies
Geolocation	Scan Rate	Frequency Bandwidth	Direction Finding
Classification	Image Resolution	Scan Rate	Geolocation
		Transmit Power	Classification

maximum packet size. For most technologies, besides from LoRaWAN in some instances, the communication window between radios is less than one millisecond.

For LoRaWAN, if the spreading factor and coding rate is high, the transmission window can be up to four seconds due to the Chirp Spread Spectrum (CSS) modulation (LoRa Alliance, 2015). This extra dwell time between communications is not accounted for in this model, as a real-world application of communication nodes requiring four seconds of transmission time is deemed unacceptable. A UAS travelling at 18 m/s, would cover 72 meters (236 feet) in 4 seconds. If a communication network had only three network hops before the CC, a UAS could travel 288 meters (945 feet) before a single detection signal was registered.

Keeping this in mind, the most pertinent factors to account for include the (i) radio frequency, (ii) transmit power, and (iii) antenna gain as these directly relate to the distance a signal can travel using FSPL as given in Equation 2.12. The noise floor and radio sensitivity are also important factors as they provide a threshold for the lowest signal a radio can receive. Finally, the dropped packet ratio (DPR) defines the rate at which a communication packet fails to be received. As stated before, the communication nodes and CC nodes differ only in how they process the received data, and therefore have the same inputs, as provided in Table 3.9.

3.3 Model Outputs

While there are many inputs to consider within the SCANS Framework, the outputs are few and easy to follow. However, as the SCANS Framework is merely a framework, data output can be adapted to meet any need. To enable easier post-processing of results, all model output is only logged by the Command And Control (CC) agent, which can then be passed to a database, text file, or another function for further processing. A custom datatype, *SensorData*, was created to

Table 3.9.

Communication Node and CC Parameter Inputs

Parameter	Datatype	Default Value	Description
frequency	Double	2400	Radio frequency in MHz
transmitPower	Double	30	Transmit power of the radio measured in dBm
antennaGain	Double	2.5	Gain for a 2.4 GHz dipole antenna measured in dB
sensitivity	Double	-80	Smallest energy signal detectable in dB
droppedPacketRatio	Double	0.05	Percentage a packet will not be received, range of 0.0 - 1.0

allow custom messages to be passed between agents. The Java data structure is given in Appendix A, and the general parameters are given in Table 3.10.

The *ID* parameter is a custom abbreviated code for each agent. These codes are provided in Table 3.11, with an index value for each agent appended to the code, starting at zero. For example, the second communication node in the model would have an *ID* value of “CN-1”. This identification system was made, in part, to provide a mechanism for labeling different agents so they can be referenced in a meaningful way, but also because it was shown that AnyLogic’s internal mechanism for providing identification values appears random for each model run.

AnyLogic numbers every agent within the same rolling number system, with identification numbers sometimes starting randomly at 600, 650, 700, or even 730. The SCANS Framework solution is useful in the case that another modeling program not have an identification assignment mechanism. The proposed framework method for labeling each agent groups them into individual agent classes, then

Table 3.10.

SensorData Class Parameters and Datatypes

Parameter	Datatype	Default Value	Description
ID	String	""	ID code of the data-generating agent
detected	Boolean	false	Whether a detection occurred
inRange	Boolean	false	Whether the UAS was in range
distance	Double	0.0	Distance between detecting agent and UAS
measure	Double	0.0	Specific measure calculated by the agent
units	String	""	Units of the measure parameter
time	Double	0.0	Model time the detection occurred in seconds
message_received	Boolean	true	Was this message received by a CC node
last_com_node	String	""	The last node to successfully receive the message

assigns an identification number, beginning at zero. The proposed identification system is consistent between model runs and the identification assignment process allows for an expected outcome to be delivered on each model run. How these identification numbers are applied is discussed later in Section 3.5.10.

The *detected* and *inRange* parameters are used together to determine false negatives. Due to the *errorRate* for sensors, it is possible to have a rogue UAS within range of the sensor but for detection to fail. This would be represented by a *false*, *true* response for *detected* and *inRange*, respectively. Output data is only generated if *inRange* is *true*. This is to limit the amount of ignorable data that would otherwise be generated by a sensor at every *pollingRate* interval. Another

Table 3.11.

Abbreviated Agent Identification Codes

Agent	Code
UAS	DR (for “DRone”)
Unidirectional Acoustic	AU
Omnidirectional Acoustic	AO
Unidirectional Radar	RU
Omnidirectional Radar	RO
Visible Spectrum Camera	VC
Infrared Camera	IR
Unidirectional Passive RF	RFU
Omnidirectional Passive RF	RFO
Communication Node	CN
Command and Control Node	CC

type of false negative can occur due to a communication node’s *droppedPacketRatio*, which is synonymous to a sensor’s *errorRate*. Both the *message_received* and *last_com_node* parameters assist in detecting when a communication failure has happened. If a *SensorData* message is sent, but a communication node fails to receive it due to the DPR, the sender’s ID is recorded in the *last_com_node* field and *message_received* is set to *false*. The packet is then forwarded on through the communication channel to the CC with no further modifications. If a message is successfully received by all communication agents, the CC will record its own ID in the *last_com_node* field and *message_received* will remain *true*.

Finally, *distance*, *measure*, *units*, and *time* relate to the measurable data present when a sensor “detects” a UAS. The *time* parameter is the model run time, in seconds, when a detection occurred. The decision to use model time instead of real-world time was two-fold. The first reason being the model could be artificially sped up or slowed down, which would render real-world time irrelevant. The second

reason being the ability to more readily compare model runs together, as the runtime always starts from zero. The real-world time is recorded when the model begins and ends, the details of which are discussed further in Section 3.5.10.

The *distance* parameter is calculated using Java’s `getDistance()` function in AnyLogic, and is scaled to the units in the *Main* agent of the model. This value is used by the model to calculate and compare against the various values each sensor is measuring. For example, the distance between the UAS and an acoustic sensor is necessary to calculate a drop in sound pressure levels, as the mathematical equation relies on comparing distances. This is similar for the other sensors as well. The *measure* parameter records the calculated value for a given sensor. For example, *Acoustic* agent sensors measure SPL, *PassiveRF* agents record the Received Signal Strength (RSS), and *Camera* and *Radars* agents record the maximum range at which a particular UAS could be detected.

The *units* parameter records the specific unit a *measure* pertains to. *Acoustic* agents return the SPL in decibels, recorded as “dB”, *PassiveRF* sensors agents return the RSS, also in dB, and the *Camera* and *Radar* sensor agents return maximum ranges in the unit of measurement declared in the AnyLogic *Main* agent, where the model is configured. These values are measured units such as yards, meters, or feet.

All of the aforementioned output is logged only by the CC agent. As a result, troubleshooting errors within the framework are minimized in relation to output data, and the functionality mimics a real-world system. Following the communication chain shown in Figure 3.1, a sensor is the only device that can generate new data. This is done when all criteria are met for a detection to occur. The sensor sends the *SensorData* message to the nearest communication node. The communication node, via a simple nearest neighbor routing protocol, sends data to only one other communication agent – either another communication node or the CC, if within range. Once the *SensorData* message reaches the CC, information is logged whatever manner chosen. Output data for the model also includes a record

of when the model started, ended, and the number of agents and their parameters for each model run.

3.4 Model Framework Layers

By combining the hierarchical class structure (Figure 3.2) with the communication chain (Figure 3.1), the SCANS Framework can be defined as a three layer system. These layers encompass the object classes, the communication chain, and how users interact with the model, which is shown in Figure 3.3. The *User Interface* layer includes simulation-program specific interfacing as well as the setting of parameters for the sensors and communication devices. The middle *Agent* layer is where sensor and communication device specific variables and functions are defined. This layer should be considered separate from the user interference. Finally, the third layer is the *Data Collection* layer and is only reachable from the CC. It is from this layer that extensions to the model's functionality can be made, such as the creation of an interdiction system.

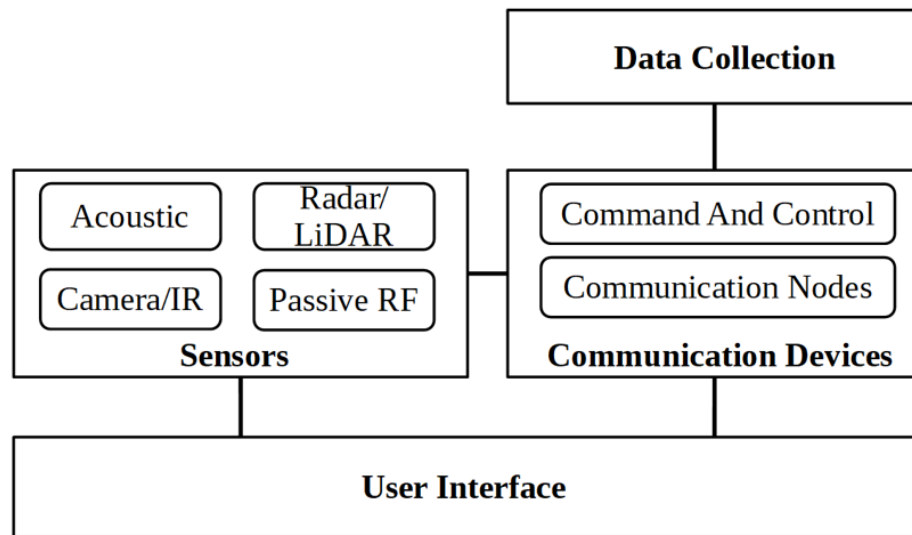


Figure 3.3. General Model Framework

3.5 Agent and Model Functions and Variables

This section provides an outline of agents, functions and variables. For distinction of terminology, what was discussed in Section 3.2, were agent *Parameters*, which are synonymous with function arguments used when instantiating a new object instance. *Variables* are for use by the model and agents themselves, without user interference. They are bits of data used solely by an agent and not directly by the user. For clarity, the agent functions will be discussed first, followed by the model startup functions, and finally any closing functions.

3.5.1 UAS Agent

The *UAS* sensor, despite being arguably the most important agent in the model, as without it this framework would not have a purpose, is the simplest agent. The variables, including the datatype and default values, are given in Table 3.12. The variables are used by the single function within the agent, `getSoundLevel()`. This function is used to assign a value to *soundPressureLevel*, which in turn is called by the *Acoustic* agent. The `getSoundLevel()` function is a simple single-line equation, given in Equation 3.1 and returns a numerical value of datatype `double`.

The *ID* variable holds the agent code (Table 3.11) and index number. As the UAS moves around the map, the framework allows for the UAS speed to vary. The *currentSpeed* variable stores the varying speed. Finally, the *soundPressureLevel* variable stores the current noise level being produced by the UAS, dependent upon speed, as shown in Equation 3.1. This variable is referenced by the *Acoustic* agent when calculating the relative SPL.

$$SPL = maxSPL * \left(\frac{currentSpeed}{maxSpeed} \right) \quad (3.1)$$

Table 3.12.

UAS Variable Names, Datatypes, and Default Values

Name	Datatype	Value
ID	String	""
currentSpeed	Double	0
soundPressureLevel	Double	0

3.5.2 Common Sensor Agent

The following variables are representative of the features in the *Sensors* agent, shown in Figure 3.2. The variables are used to create the *SensorData* message that is sent from a sensor to a communication device. These variables are given in Table 3.13. While all of the function names are shared between all sensor types, only two are truly universal: `determineClosestComNode()` and `calculateRelativeDronePosition()`. The general algorithms for each are given in Algorithm 3.1 and 3.2, respectively. Of note, `calculateRelativeDronePosition()` does not currently take into account the altitude of the UAS when determining sensor FOV. The model does, however, determine the overall distance between a sensor and the UAS in three-dimensional (3D) space. This means an assumption exists that the sensor is properly directed at the rogue UAS in terms of the altitude angle. Given a sensor with a small vertical FOV, this limitation may provide unrealistic results. These inaccurate results would occur if, in the real-world test, a UAS would fly overhead a sensor, undetected. This undetected overhead flight scenario is currently not possible in the SCANS Framework.

Concerning the variables listed in Table 3.13, *ID* is as described in Section 3.5.1. The variables *isDetected*, *inRange*, *droneDistance*, and *sensorMeasure* are all part of the *SensorData* message sent upon the detection of a UAS. The *sensorMeasure* variable is the individual measure made by a given sensor and represents the data that could be acted upon by an interdiction system. The

nearestCom variable holds the agent to which this sensor should send data to. At this time, it is assumed that a sensor will always connect with the nearest communication node, regardless of distance. However, the communication node could still fail to receive the message based upon its DPR. When the *rotates* parameter is used in conjunction with *rotationSpeed*, an agent will rotate in place. The agent's current angle of rotation is held in the variable *rotationAngle*. For each following sensor descriptions, the functions and procedures are similar regardless of whether a sensor is rotating. If movement is involved, the sensor first rotates based upon the *rotationSpeed*, then proceeds through the standard procedure described.

Table 3.13.

UAS Variable Names, Datatypes, and Default Values

Name	Datatype	Value
ID	String	""
isDetected	Boolean	false
inRange	Boolean	false
droneDistance	Double	0
sensorMeasure	Double	0
nearestCom	CommunicationNode	null
rotationAngle	Double	0

Algorithm 3.1 Determine the Closest Communication Node to Send SensorData to

```

1: function DETERMINECLOSESTCOMNODE
2:   currentDistance  $\leftarrow$  0 ▷ Distance of the current nearest neighbor
3:   for For Every ComNode Agent do
4:     cn  $\leftarrow$  ComNode(i)
5:     potentialDistance  $\leftarrow$  getDistance(sensor.Location, cn.Location)
6:     if potentialDistance < currentDistance | currentDistance = 0 then
7:       currentDistance  $\leftarrow$  potentialDistance
8:       nearestCom  $\leftarrow$  cn

```

3.5.3 Acoustic Sensor Agent

The *Acoustic* sensor agent is the simplest of all sensors and is most similar to its parent *Sensor* agent. The *Acoustic* agent has no unique variables. However, similar to all other specific sensor types, it has `sense()`, `calculateMeasure()`, and `sendAlert()` functions. The `sense()` and `calculateMeasure()` functions require a

Algorithm 3.2 Calculate the UAS Position Relative to a Sensor

```

1: function CALCULATERELATIVEDRONEPOSITION
2:   UAS output  $\leftarrow$  null
3:   for Every UAS agent do
4:     drone  $\leftarrow$  UASagent(i)
5:     x, y, z  $\leftarrow$  sensor.Location - drone.Location
6:     rot  $\leftarrow$  |sensor.GetRotation() * 180/ $\pi$ | mod 360
7:     relativeRot  $\leftarrow$  (arctan(|y|/|x|) * 180/ $\pi$ ) mod 360
8:     droneInView  $\leftarrow$  false
9:     ccwRot  $\leftarrow$  (((rot + sensor.FoV/2) mod 360) + 360) mod 360
10:    cwRot  $\leftarrow$  (((rot - sensor.FoV/2) mod 360) + 360) mod 360
11:    if x < 0 then ▷ If x is negative
12:      if y < 0 then ▷ If y is negative
13:        if ccwRot > relativeRot & cwRot < relativeRot then
14:          droneInView  $\leftarrow$  true
15:        else ▷ Negative x, Positive y
16:          if ccwRot > (360 - relativeRot) & cwRot < (360 - relativeRot)
17:            droneInView  $\leftarrow$  true
18:        else
19:          if y < 0 then ▷ Positive x, Negative y
20:            if ccwRot > (180 - relativeRot) & cwRot < (180 - relativeRot)
21:              droneInView  $\leftarrow$  true
22:            else ▷ Positive x, Positive y
23:              if ccwRot > (180 + relativeRot) & cwRot < (180 + relativeRot)
24:                droneInView  $\leftarrow$  true
25:          if droneInView = true then
26:            output  $\leftarrow$  drone
27:            BREAK
28:   Return output

```

UAS agent be passed as a parameter to enable processing. All of the functions are interwoven amongst each other, with one function calling another. Algorithm 3.3 is the start of this process and is similar for other sensor types. Algorithms 3.4 - 3.6 show the three individual functions. While `sense()` and `calculateMeasure()` change drastically depending upon the sensor employed, the only difference for `sendAlert()` between sensors is the *sensorMeasure* value and the *measureUnits* (from the *SensorData* datatype). As such, Algorithm 3.3 and Algorithm 3.6 will only be given here and each subsequent sensor section will specify adaptations as necessary.

To better clarify potential confusion within the algorithms, *scaleUnits*, listed in Algorithm 3.4 line 4, is a variable implemented in the *Main* agent of AnyLogic. The variable holds the unit of measure (i.e. meter, foot, etc.) used for model scaling. This implementation is used because of AnyLogic quirks, regarding access to the *scale* entity. The Java function `getDistance()` returns the number of pixels between two points, and *scaleUnit* is used to convert this data using the `scale.pixelsPerUnit(LengthUnit)` function. Another point of clarification is for the use of `randomFalse()`. This native Java function takes in a percentage value between 0.0 and 1.0 (given by the *errorRate* parameter) and will return *false* by that percent based upon a random number generator. Said another way, if *errorRate* = 0.05 and `randomFalse(errorRate)`, then *false* would be returned 5% of the time. This function is used throughout the model to introduce error in what would otherwise be a never-faulting simulation.

Algorithm 3.3 Determine if a UAS is Nearby

```

1: procedure IS A UAS NEARBY
2:   for For Every UAS Agent do
3:     SENSE(UAS(i))
4:     if isDetected & inRange then
5:       BREAK

```

Algorithm 3.4 Acoustic Scanning Function

```

1: function SENSE(UAS)
2:   sensorMeasure  $\leftarrow$  CALCULATEMEASURE(UAS)
3:   droneDistance  $\leftarrow$  getDistance(sensor.Location, UAS.Location)
4:   droneDistance  $\leftarrow$  droneDistance/scaleUnits    ▷ Scale the distance by the
   proper units
5:   if sensorMeasure  $\geq$  noiseFloor & randomFalse(errorRate) then    ▷
   java.randomFalse returns false by the percentage rate given it
6:     Set Detection to true
7:     isDetected  $\leftarrow$  true
8:     inRange  $\leftarrow$  true
9:   else
10:    if sensorMeasure  $\geq$  noiseFloor then
11:      inRange  $\leftarrow$  true
12:    else
13:      inRange  $\leftarrow$  false
14:      isDetected  $\leftarrow$  false
15:      Set Detection to False

```

Algorithm 3.5 Calculate the SoundPressureLevel Using UAS speed

```

1: function CALCULATEMEASURE(UAS)
2:   droneDistance  $\leftarrow$  getDistance(sensor.Location, UAS.Location)
3:   droneDistance  $\leftarrow$  droneDistance/scaleUnits    ▷ Scale the distance by the
   proper units
4:   intensityRatio  $\leftarrow$   $(\log_{10}(\frac{droneDistance^2}{UAS.maxSPLDistance^2}) * 100)/100$ 
5:   dB  $\leftarrow$   $-1 * (10 * intensityRatio - UAS.soundPressureLevel)$ 
6:   return dB

```

Algorithm 3.6 If Detection Occurs, Send an Alert

```

1: function SENDALERT
2:   Create a new instance of SensorData with units = dB
3:   data  $\leftarrow$  new SensorData(...)
4:   if isDetected = false then
5:     data.setMessage_Received  $\leftarrow$  false
6:     data.setLastComNode  $\leftarrow$  sensor.ID
7:   if nearestCom = null then DETERMINECLOSESTCOMNODE
8:   Send data to nearestCom

```

3.5.4 Radar Sensor Agent

The *Radar* sensor agent has only one additional variable over the general *Sensor* agent, *droneRange*. This variable is a **Map** interface type in the Java

language and is used as a `HashMap`. Given the lack of literature pertaining to the specifications of radar systems, this framework assumes an approach of calculating the maximum distance a radar system could potentially view a UAS using its RCS. For the `sendAlert()` function, the *sensorMeasure* used is the maximum distance the radar could potentially detect the UAS and the *measureUnits* are based upon whichever unit of length is used in the model's main view (the *Main* agent). Algorithm 3.7 and 3.8 are provided to highlight the differences in processing between the sensor types.

3.5.5 Camera Sensor Agent

Similar to the *Radar* agent, the *Camera* agent adds only the single *droneRange* variable. It functions in the same manner as described in Section 3.5.4. The `sense()` function is also the same as the *Radar* agent. Algorithm 3.9 defines the changes in distance processing from the *Radar* agent. Instead of using the speed

Algorithm 3.7 Radar Scanning Function

```

1: function SENSE(UAS)
2:   CALCULATEMEASURE(UAS)
3:   droneDistance  $\leftarrow$  getDistance(sensor.Location, UAS.Location)
4:   droneDistance  $\leftarrow$  droneDistance / scaleUnits  $\triangleright$  Scale the distance by the
   proper units
5:   if droneRange.UAS.ID  $\geq$  droneDistance & randomFalse(errorRate)  $\triangleright$ 
   droneRange.UAS.ID is a HashMap then
6:     Set Detection to true
7:     sensorMeasure  $\leftarrow$  droneRange.UAS.ID
8:     isDetected  $\leftarrow$  true
9:     inRange  $\leftarrow$  true
10:  else
11:    if droneRange.UAS.ID  $\geq$  droneDistance then
12:      inRange  $\leftarrow$  true
13:    else
14:      inRange  $\leftarrow$  false
15:    isDetected  $\leftarrow$  false
16:    Set Detection to False

```

Algorithm 3.8 Calculate Maximum Detection Range Using UAS RCS

```

1: function CALCULATEMEASURE(UAS)
2:   if droneRange = null then
3:     Instantiate the droneRange HashMap
4:   if droneRange does not have the UAS key then
5:      $c \leftarrow 299792458$  ▷ Speed of Light
6:     Let s reference this sensor
7:      $numerator \leftarrow s.TxPower * s.antennaGain^2 * \frac{c}{s.frequency * 10^9}^2 * UAS.radarCrossSection$ 
8:      $denominator \leftarrow (4 * \pi)^3 * s.minimumReceivePower * s.systemLoss$ 
9:      $range\_max \leftarrow \frac{numerator}{denominator}^{0.25}$ 
10:    droneRange  $\leftarrow UAS.ID, range\_max$  ▷ Add data to droneRange HashMap

```

of light and radio frequencies, the *Camera* sensor uses the picture resolution, pixels per foot required for detection, and the camera lens angle to determine the maximum detection range. The specific dimensions of the object being detected are of no consequence as this is captured in the *pixelsPerFoot* (PPF) metric.

Algorithm 3.9 Calculate Maximum Detection Range of Camera

```

1: function CALCULATEMEASURE(UAS)
2:   if droneRange = null then
3:     Instantiate the droneRange HashMap
4:   if droneRange does not have the UAS key then
5:     Let s reference this sensor
6:      $FOV \leftarrow 0$ 
7:     if s.fieldOfView is not set then
8:        $FOV = s.resolutionWidth / s.PPF$ 
9:     else
10:       $FOV = s.fieldOfView$ 
11:       $range\_max = \frac{FOV * 360 / s.lensAngle}{2\pi}$  ▷ Returns distance in feet
12:      Convert range_max to scaleUnits ▷ Where scaleUnits is the unit of length the model displays
13:      droneRange  $\leftarrow UAS.ID, range\_max$  ▷ Add data to droneRange HashMap

```

3.5.6 PassiveRF Sensor Agent

For the *PassiveRF* sensor agent, no additional variables are required over the ones provided in Section 3.5.2. Functionally speaking, `sense()` adds an additional condition to lines 5 and 10 of Algorithm 3.4, that being *sensorMeasure* \geq *s.sensitivity*. The `calculateMeasure()` function, which is unique to this agent, is given in Algorithm 3.10

Algorithm 3.10 Calculate Received Signal Strength Using UAS Signaling

```

1: function CALCULATEMEASURE(UAS)
2:   Let s reference this sensor
3:   DroneTxFrequency  $\leftarrow$  0  $\triangleright$  The UAS agent has multiple radio settings, these
   holder variables allow us to focus on just one
4:   droneTxPower  $\leftarrow$  0
5:   droneTxGain  $\leftarrow$  0
6:   if  $|UAS.frequency1 - s.frequency| \leq 20$  then  $\triangleright$  Make sure the
   frequencies are within one channel of each other
7:     droneTxFrequency  $\leftarrow$  UAS.frequency1
8:     droneTxPower  $\leftarrow$  UAS.TxPower1
9:     droneTxGain  $\leftarrow$  UAS.antennaGain1
10:  else if  $|UAS.frequency2 - s.frequency| \leq 20$  then
11:    droneTxFrequency  $\leftarrow$  UAS.frequency2
12:    droneTxPower  $\leftarrow$  UAS.TxPower2
13:    droneTxGain  $\leftarrow$  UAS.antennaGain2
14:  else
15:    return -999  $\triangleright$  If the frequencies don't match, -999 guarantees the radios
   will never sense each other
16:  distance  $\leftarrow$  getDistance(s.Location, UAS.Location)
17:  Scale distance by Kilometers
18:  Set droneDistance to distance, scaled for scaleUnits
19:  FSPL  $\leftarrow$   $20 * \log_{10}(distance) + 20 * \log_{10}(droneTxFrequency) + 32.44 -$ 
   s.antennaGain - droneTxGain
20:  return droneTxPower - FSPL

```

3.5.7 Common Communication Device Agent

Between The *CommunicationNode* and the *CommandAndControl* agents, only two additional variables are added for their personal use. The first is *ID* which operates in a similar fashion as the other agents. The second variable, *sensorData*, is of type *SensorData* and stores any incoming message for further processing. The general process for handling data is straight forward for each agent and does not warrant an algorithm to be developed, but is explained in the follow paragraph.

Upon receiving a message, *msg*, the *CommunicationDevice* has a *droppedPacketRatio* chance of marking the message as failed to receive. If the message is not received, *msg.messageReceived* is set to *false* and *msg.lastComNode* is set to *this.ID* where *this* refers to the sensor itself. If the packet is received, nothing within the *msg* changes. The *msg* is then stored in the *sensorData* variable. Finally, the data is either sent onward, if the agent is a *CommunicationNode*, or recorded in the case of being a *CommandAndControl* agent.

3.5.8 CommunicationNode Agent

The *CommunicationNode* acts as the glue to allow sensor agents to communicate with the CC. To achieve this, the *CommunicationNode* employs a rudimentary nearest-neighbor routing mechanism, which are discussed in the Algorithms 3.11 - 3.13. Once a nearest-neighbor is determined, that agent is stored in the *nearestNeighbor* variable, which has a default value of *self*. Ideally, this would be the only variable needed. However, because a *CommunicationNode* and *CommandAndControl* agent are two different datatypes, two variables are needed. Hence, there is also a *nearestCCNode* variable. As long as *nearestCCNode* is assigned, the *CommunicationNode* agent will send data directly to the CC as that is the ultimate message destination. Otherwise, the data is sent to the nearest *CommunicationNode*. To accomplish these goals, the agent makes use of a *determineConnections()* (Algorithm 3.11 & 3.12), *determineNeighbor()*

(Algorithm 3.13), and `sendData()` (Algorithm 4.5) function. None of the functions require an input parameter.

A new AnyLogic element, different from other agents up to this point, is what AnyLogic calls “Connections”. The *CommunicationNode* has two, *comNodeLinks* and *ccLinks*, and are simply ArrayLists of *CommunicationNode* and *CommandAndControl* agents, respectively. These allow for visual links to be drawn between agents as well as allows for easier directed communication between agents. They are not crucial for the SCANS Framework, but a list is necessary for `determineConnections()` to determine which agents are closest.

Algorithm 3.11 Determine All Potential NearestNeighbors: Part 1 of 2

```

1: function DETERMINECONNECTIONS
2:   if     nearestCCNode is null | nearestNeighbor is not set to self
   then
3:     for Every CommandAndControl agent do
4:        $cc \leftarrow \text{CommandAndControl}(i)$ 
5:       if  $|cc.frequency - self.frequency| \geq 20$  then
6:         CONTINUE
7:        $distance \leftarrow \text{getDistance}(cc.Location, self.Location)$       ▷ Returns a
   value in pixels
8:       Scale  $distance$  to Kilometers
9:        $FSPL \leftarrow 20 * \log_{10}(distance) + 20 * \log_{10}(self.frequency) + 32.44 -$ 
    $self.antennaGain - cc.antennaGain$ 
10:       $RSS \leftarrow self.transmitPower - FSPL$ 
11:      if  $RSS \geq noiseFloor$  &  $RSS \geq cc.sensitivity$  then
12:        if ccLinks does not contain cc then
13:           $RSS \leftarrow cc.transmitPower - FSPL$       ▷ Test that the
   connection will work for both agents
14:          if  $RSS \geq self.noiseFloor$  &  $RSS \geq self.sensitivity$ 
   then
15:             $ccLinks.Add(cc)$ 
16:          else
17:             $ccLinks.Remove(cc)$ 
18:          else
19:             $ccLinks.Remove(cc)$ 

```

Algorithm 3.12 Determine All Potential NearestNeighbors: Part 2 of 2

```

20: function DETERMINECONNECTIONS
21:   if nearestCCNode is null | nearestNeighbor is set to self then
22:     for Every CommunicationNode agent do
23:        $cn \leftarrow \text{CommunicationNode}(i)$ 
24:       if  $|cn.frequency - self.frequency| \geq 20$  |  $cn = self$  then
25:         CONTINUE
26:        $distance \leftarrow \text{getDistance}(cn.Location, self.Location)$  ▷ Returns a
        value in pixels
27:       Scale  $distance$  to Kilometers
28:        $FSPL \leftarrow 20 * \log_{10}(distance) + 20 * \log_{10}(self.frequency) + 32.44 -$ 
         $self.antennaGain - cn.antennaGain$ 
29:        $RSS \leftarrow self.transmitPower - FSPL$ 
30:       if  $RSS \geq noiseFloor$  &  $RSS \geq cn.sensitivity$  then
31:         if comNodeLinks does not contain cn then
32:            $RSS \leftarrow cn.transmitPower - FSPL$  ▷ Test that the
            connection will work for both agents
33:           if  $RSS \geq self.noiseFloor$  &  $RSS \geq self.sensitivity$ 
then
34:              $comNodeLinks.Add(cn)$ 
35:           else
36:              $comNodeLinks.Remove(cn)$ 
37:         else
38:            $comNodeLinks.Remove(cn)$ 

```

3.5.9 CommandAndControl Agent

The *CommandAndControl* (CC) agent is much simpler than the aforementioned *CommunicationNode* agent. Only a single function needs defining for this agent, which is `sendData()`. While the function name is similar to the *CommunicationNode* agent, in the CC agent `sendData()` is used to log data for later analysis. Simply put, a *SensorData* message arrives from a *CommunicationNode* agent and the output is sent to the chosen output venue. The function can be easily adapted to output to a CSV (comma separated values) file, a standard text file, or a database. No additional variables from those discussed in Section 3.5.7 are needed for this agent. Lastly, an AnyLogic *collection* named

Algorithm 3.13 Determine the NearestNeighbor

```

1: function DETERMINENEIGHBOR
2:   connectedCCNodes  $\leftarrow$  ccLinks.getAll()
3:   if connectedCCNodes.size() > 0 then
4:     nearestCCNode  $\leftarrow$  connectedCCNodes[0]  ▷ If any CC nodes are within
       range, send data directly there
5:     nearestNeighbor  $\leftarrow$  self
6:   else
7:     nearestCCNode  $\leftarrow$  null
8:     connectedNodes  $\leftarrow$  comNodeLinks.getAll()
9:     currentDistance  $\leftarrow$  0
10:    if nearestNeighbor  $\neq$  self.ID | cn.nearestNeighbor.ID = self.ID then
11:      CONTINUE
12:    potentialDistance  $\leftarrow$  getDistance(self.Location, cn.Location)
13:    if nearestNeighbor.ID = self.ID then
14:      nearestNeighbor  $\leftarrow$  cn
15:      currentDistance  $\leftarrow$  potentialDistance
16:      CONTINUE
17:    if potentialDistance < currentDistance then
18:      nearestNeighbor  $\leftarrow$  cn
19:      currentDistance  $\leftarrow$  potentialDistance
20:    for Every connectedNodes do
21:      if nearestNeighbor.ID = connectedNodes[i].ID |
22:        connectedNodes[i].nearestNeighbor.ID = self.ID then
23:        CONTINUE
24:      else
25:        comNodeLinks.Remove(connectedNodes[i])

```

comNodeLinks is used to track connections between this agent and a *CommunicationNode* agent. This is as described in Section 3.5.8.

3.5.10 Model Startup and Shutdown

By using AnyLogic, much of the preparatory work for the model begins before the model is even compiled. If implementing the framework in another environment, take note that the specifics for accomplishing the following tasks will differ. Focusing primarily on the *Main* agent, which is the default Java.Main class, the flight path for the UAS, flight mechanics, agent populations, and several public

variables are configured. It is specifically within this agent where *Sensors* are placed and the model's global scale is set.

There are three variables implemented, however one is merely for AnyLogic presentation purposes. As the model's scale can easily shift depending on the applicable distance used in a given scenario, the *globalScale* variable allows for all graphics to remain the same size for visibility. Previous works, such as Lee (2019) and Cline and Dietz (2020), utilized the scaling of presentation graphics and Java's `collision()` function to determine if an object is within Line of Sight (LOS). As each agent of the SCANS frameworks bases its detection of another agent on location and data values, presentation graphic scaling is a non-issue. This allows the graphical representation of agents to be scaled for visibility without effecting the overall performance of the model. The second variable, which has been discussed before in various sensor algorithms, is the *scaleUnits* variable.

As a simulation, scale must be applied in some fashion. Due to limitations, whether real or unknown, the scale units of an AnyLogic model is not a stored property. This variable stores the `LengthUnits` value which the *Main* agent's scale is configured for. For example, if the *Main* agent had a scale of 400 meters, *scaleUnits* would equal "METER". In this way, any subsequent agent can perform conversion gymnastics to ensure all output reports using similar units. Finally, the third variable is *droneCount*, which acts as the index counter for the *UAS* agent's *ID* field.

Due in part to the chosen implementation, yet still adaptable to any situation, exists two functions. The *Main* agent's `startup()` function and a `placeSensors()` function, are ran at model startup. The `placeSensors()` function is executed before the `startup()` function. Neither of the functions have input parameters associated with them.

The purpose of the `placeSensors()` function is twofold. The first is due to an AnyLogic implementation limitation dealing with how agents are placed within a model. AnyLogic is unable to specify a direct location for more than one type of

agent and must make use of a *PointNode*. These *PointNodes* are circles placed around the model space and are added to a specific collection associated with a given population of sensor. The `placeSensors()` function programmatically moves each agent into place, as well as logs the agent's parameters into a database table. The function also provides agents with their *ID*.

The `onStartup()` function is used to inject a single *UAS* agent into the model to begin. The function also provides the *UAS* agent with its *ID*. A button is used to add subsequent *UAS* agents to the model. A *droneCount* variable is incremented each time and used to keep track of the spawned *UAS* agents.

Finally, the `onDestroy()` function is used to log the number of agents and agent types to a database table along with the time the model started and ended. This record allows for the distinction of model runs to be made at a later time, as well as to ensure the models were run using a similar number of agents.

3.6 Methodology for Evaluating the Framework

Verification is gained by ensuring the model's programming code is implemented correctly. This chapter has shown, through the various algorithms, how the Framework should be implemented, however, the specific implementation described in Chapter 4 needs to be verified. This is accomplished by testing the SCANS frameworks functionality and feature set to ensure the AnyLogic implementation operates as expected. For example, when the *rotationOffset* parameter is enabled, does the agent rotate accordingly.

Validation confirms the accuracy of the model against a real-world system. The objective operational validation approaches described in Sargent (2011) are used to test the SCANS Framework validity. As a general sensor model, output of the system should be dependent upon the sensor parameters defined during model deployment. Historical positive economics validation is obtained by comparing the output of multiple published experiments involving various sensors such as Yang

(2019) (acoustics), Farlik et al. (2019) (radar, passiveRF, electro-optical), and Park et al. (2020)(radar). This is accomplished by comparing the calculated results of a built model, in AnyLogic, to the empirical measurements obtained through the studies. A positive result is determined when the published experiment’s measurement is matched by implemented AnyLogic model’s calculation.

For testing and evaluation purposes, models of the above-mentioned published works were created. A model of the first experiment of Yang (2019) was built and ran three times. This shows the AnyLogic model’s capability to vary data between runs, but highlights the similarities between runs as well. The output of the SCANS model can provide true positive, false negative, and true negative responses, but not false positive. False positives are impossible, as this would require the model to “make up” detection events in a random, believable fashion.

Data processing for the three iterations used the Python scripting language to generate `pColorMesh` charts showing detection events. This requires the `numpy`, `pandas`, and `matplotlib.pyplot` packages. The `pColorMesh` provides a yellow bar to indicate a detection and a purple bar to indicate no detection. The model outputs data to a database table called *sensor_data*, where the *originating_sensor_id*, *detected*, and *model_time* columns are used to create the charts. Windows PowerShell is used to prepare the data for ingestion by the `pColorMesh` function. All data processing scripts and a description of their function is provided in Appendix C.

In addition to the `pColorMesh` detection charts, the number of true positives, false negatives, average detected SPL value, and the detected SPL’s standard deviation is reported for each sensor, as well as the overall system. Only true positive values were used for the calculation of the average and standard deviation.

Unlike Yang’s study, Farlik et al. and Park et al. evaluated sensor performance, as opposed to system performance. Farlik et al. performed a survey of sensor measurements for CUAS applications including acoustic, electro-optical, radar, and passive RF sensors. Park et al. developed their own radar system and

tested the system performance using a car, person, and small UAS. For each applicable study of sensors (electro-optical, radar, and passive RF for Farlik et al., and radar for Park et al.) the sensor parameters were given, or were found using specification sheets from the equipment manufacture. In some cases, such as to find a camera sensor's lens angle, the maximum distance measured within the study and the appropriate sensor parameters were calculated using the equations from Section 2.1. From the found, or derived, sensor parameters, models were constructed within AnyLogic, where the resultant maximum sensed distance was compared to the empirical study's finding. Additional analysis concerning the model's performance versus expectations is then given for each sensor type.

3.7 Summary

This chapter has provided the outline for the SCANS Framework including class hierarchies, parameters, variables, and methods. Each agent was introduced, including the *UAS*, *Sensor* types, and *CommunicationDevice* types. Algorithms were provided for implementation within an individual's simulation software of choice and the interaction between agents was explained. All of this was derived from the AEIOU framework used to describe and document a potential CUAS scenario and the literature highlighted in Chapter 2. Finally, the methodology for performing validation and verification testing was described using the objective operational validation approach. Chapter 4 shows the implementation of the SCANS Framework within AnyLogic, highlighting the unique quirks of the system and the adaptations necessary to work within AnyLogic's specifications.

CHAPTER 4. ANYLOGIC IMPLEMENTATION

Chapter 3 defined the basic operational structure and feature set for the SCANS Framework. This chapter provides the details of how the SCANS Framework was implemented within AnyLogic. AnyLogic was chosen based upon the program availability, as well as the researcher’s previous experience with the software. Parts of the SCANS Framework are easily implemented within AnyLogic, such as agents and behaviors, however some components, such as object inheritance must be adapted.

The structure for the chapter begins with the *Main* agent’s elements. The chapter then details the *UAS* agent, the individual sensor agents, and finally the communication device agents. The chapter uses the work of Yang (2019) for the layout of the *Main* agent, as that work will be the first experiment compared against in Chapter 5.

4.1 Building the Model

Building any model in AnyLogic is as simple as clicking and dragging the desired element onto the screen, which is done through the various palette windows. All elements within this model were made from either the “Process Modeling” , “Presentation”, or “Agent” palettes, shown in Figures 4.1(a) - 4.1(d). In general, each element has standard properties such as size, color, and name, as well as areas to add additional Java code for a dynamic action.

In addition to creating the various agents, AnyLogic separates the process of executing the model from process of designing the model. In this way, experiments can have varied input parameters while using the same base model. Unfortunately, as each of the experiments and tests described later in the chapter are structurally

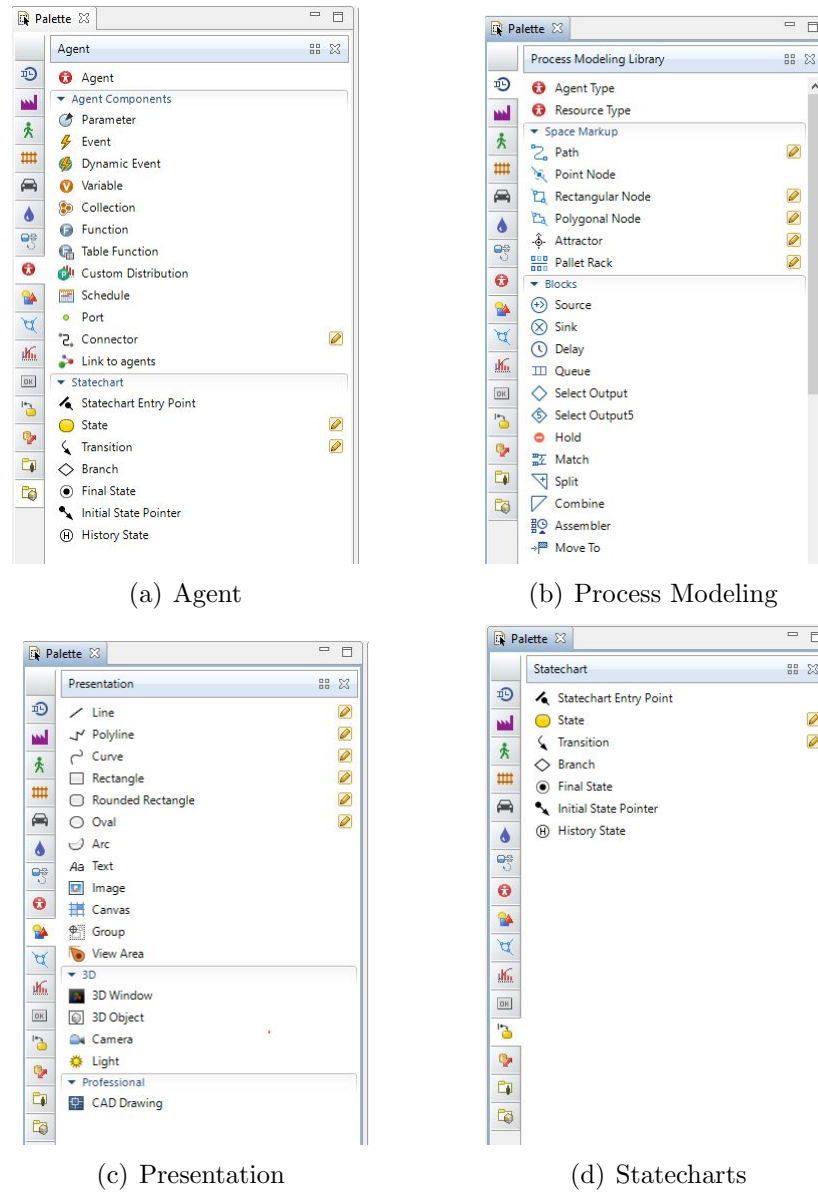


Figure 4.1. AnyLogic Element Palettes

different in their design, this feature is of no use in this application. Nonetheless, a single simulation screen must be configured, and that screen is shown is Figure 4.2. This is the landing page for the model at runtime. The single functional aspect on this page is the ability to enter a name for a particular model, which can be used as

a label for identifying multiple tests at a later date. This name is written to the *model_runs* internal AnyLogic database table.

4.1.1 Creating the *Main* Agent

The experiment performed in Yang (2019) is the ideal test case for the SCANS framework due to the experimental setup, the use of a communication network with a command and control center, and the graphical output, which can be replicated from the AnyLogic model output. As the understanding of Yang's system layout is paramount for understanding the goal of this build model, Figure 4.3 provides the basic system layout, the network structure, and flight paths as published in Yang's work 2019, p.52.

Yang described four separate experiments, each with six acoustical sensors, shown in Figure 4.4. Using Experiment 1 as a reference, and the AnyLogic palettes,

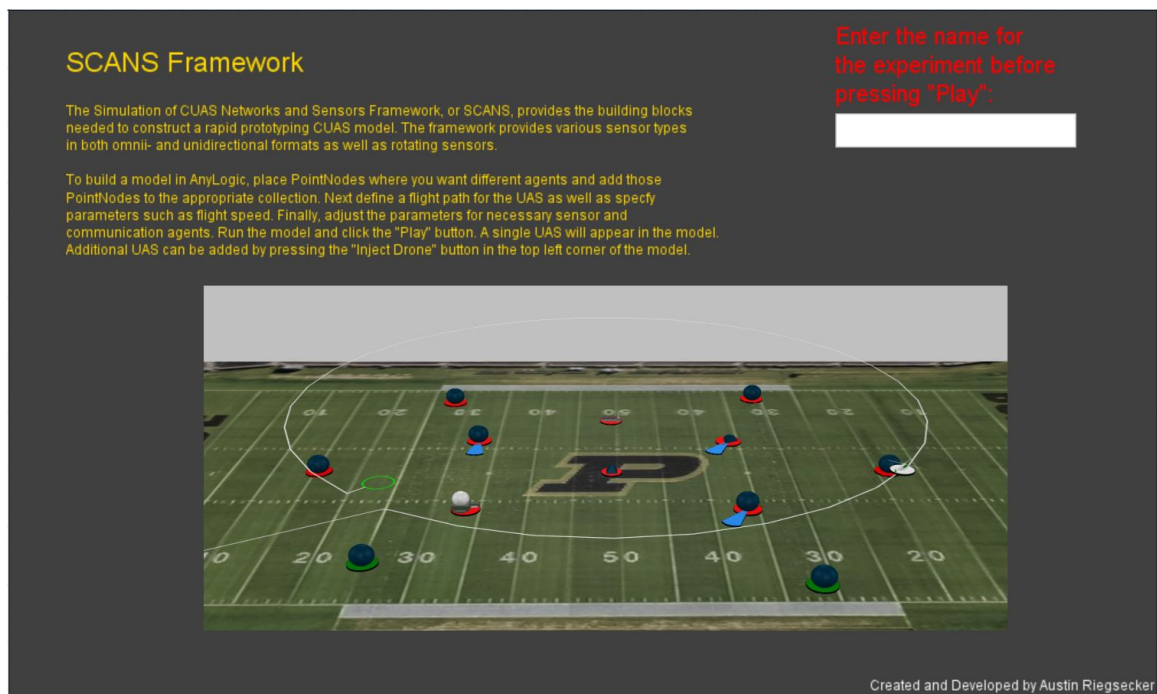


Figure 4.2. SCANS Framework Landing Page

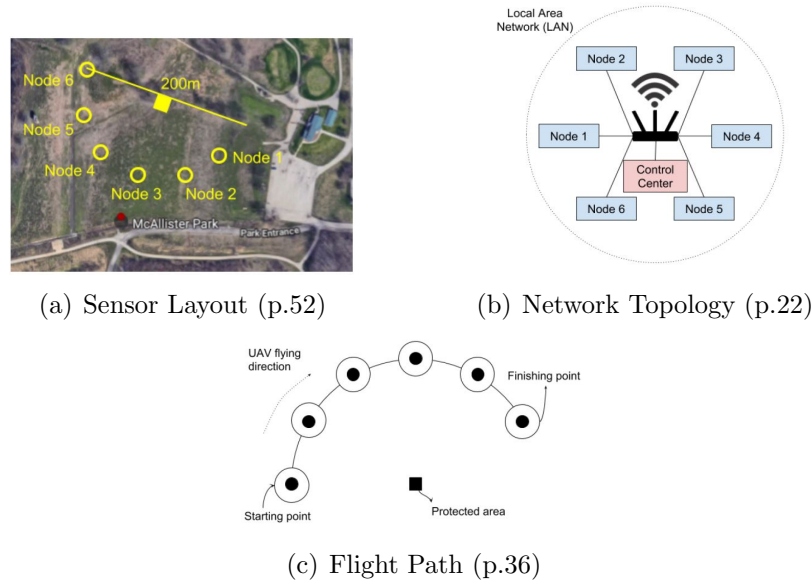


Figure 4.3. Experimental Setup for Yang (2019)

the visual aspect of the *Main* agent was made, which is shown in Figure 4.5. Figure 4.6 is provided without the additional annotations and represents the true view of the *Main* agent within AnyLogic. This and all models have a 3D graphical component to them, however only the 2D representation is shown here.

One of AnyLogic's limitations is lacking the ability to place agents without geographic information system (GIS) coordinates. To overcome this, *point node* elements are placed and color coded within the model and added to an AnyLogic *collection*. These *collections* are highlighted on the left of Figure 4.5 and are later referenced by the `placeSensors()` function to place agents at model runtime. A zoomed in view of the agent populations and collections is provided in Figure 4.7. Collections are simply `ArrayList` and are referenced as such within the Java programming language. The build begins by placing a central, magenta colored *point node* element which represents the *CommunicationNode* agent and a gold colored *point node* for the *CommandAndControl* agent, labeled as such in the figure.

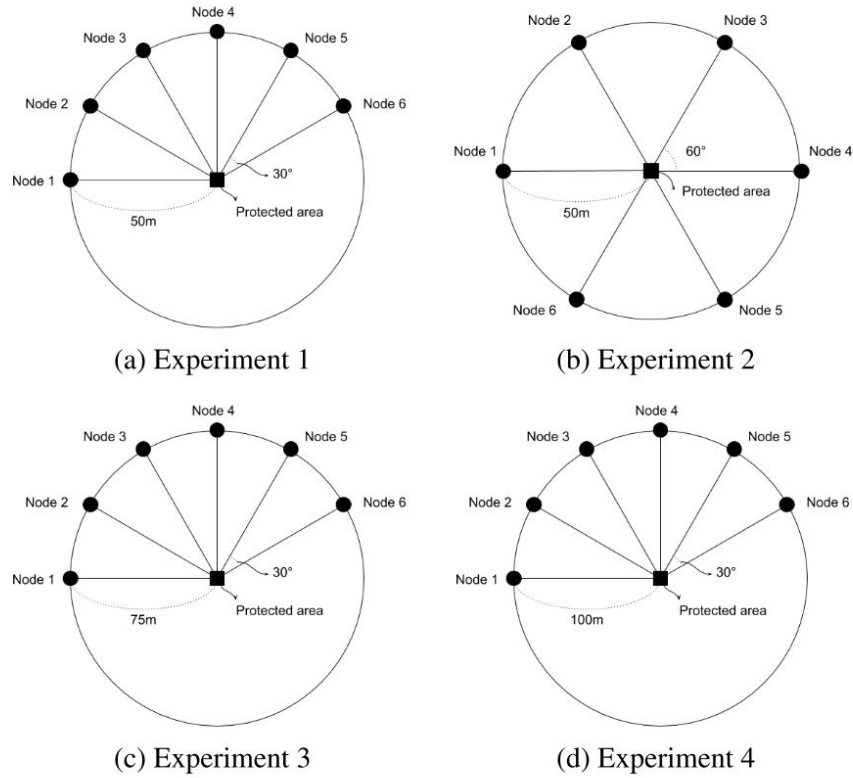


Figure 4.4. Yang's (2019, p.35) Acoustic Node Layout for Experiments

They are added to the *CommunicationNodes* and *CommandAndControlNodes* collections, respectively.

From Yang, the UAS was flown over each acoustic sensor with a potential horizontal flight space of twenty meters and an altitude of ten meters. The light-blue, dashed lines in Figure 4.5 represent this flight space. These lines are only present for the benefit of the model viewer and do not serve any functional purpose. The green *point nodes* (green circles), labeled “flight start” and “flight end” define where the *UAS* agent is spawned into the model and the destination goal of the *UAS*, respectively. Upon reaching its goal, the *UAS* turns around and follows the flight path back to the “flight start” node where the *UAS* is removed from the model. The white, dashed line in the figure is the *path* element and defines a *UAS*

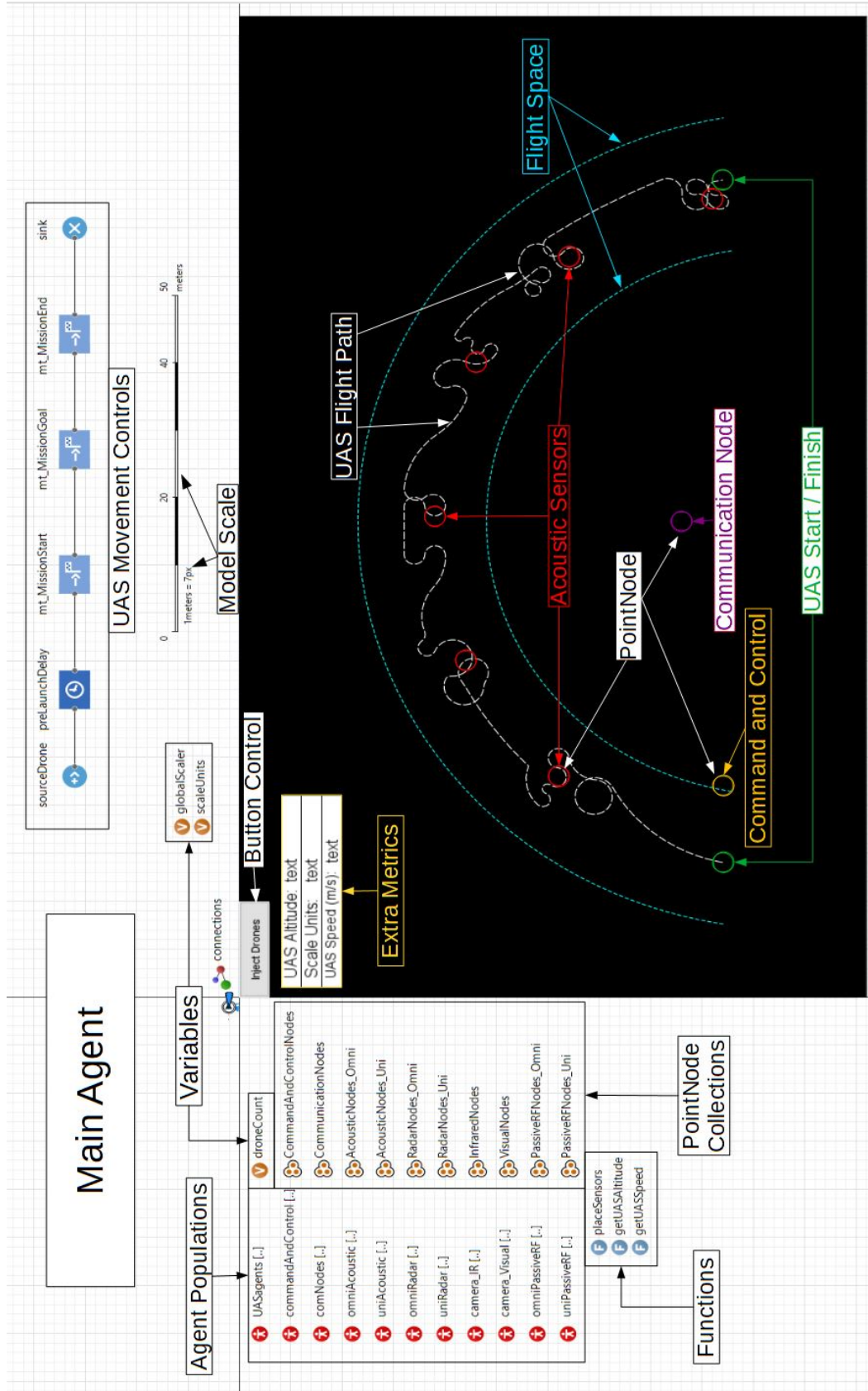


Figure 4.5. Annotated Experiment 1 of Yang (2019) Within the *Main Agent*

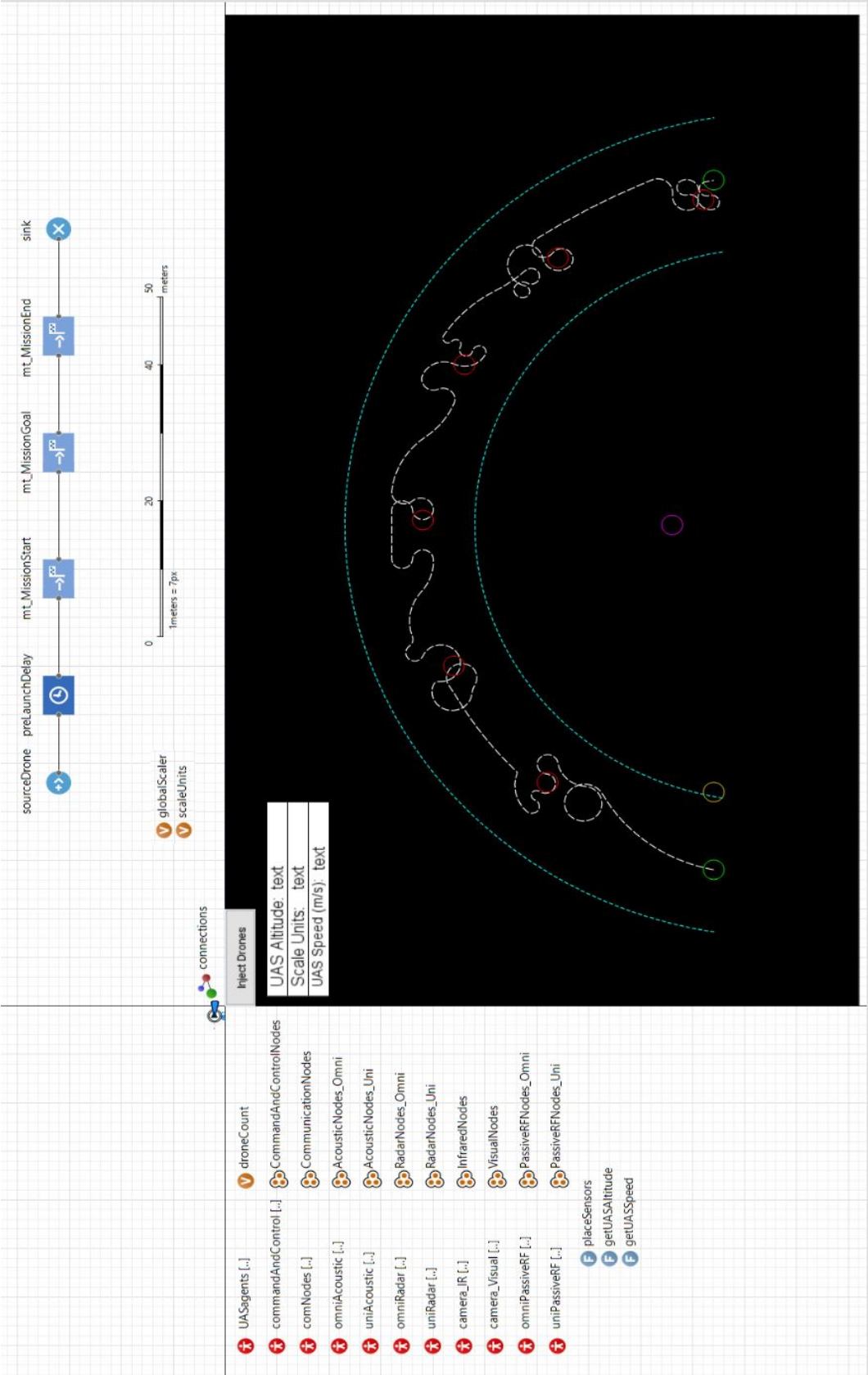


Figure 4.6. Experiment 1 of Yang (2019) Without Annotations






















Agent Populations	Point Node Collections
 UASagents [..]	
 commandAndControl [..]	 CommandAndControlNodes
 comNodes [..]	 CommunicationNodes
 omniAcoustic [..]	 AcousticNodes_Omni
 uniAcoustic [..]	 AcousticNodes_Uni
 omniRadar [..]	 RadarNodes_Omni
 uniRadar [..]	 RadarNodes_Uni
 camera_IR [..]	 InfraredNodes
 camera_Visual [..]	 VisualNodes
 omniPassiveRF [..]	 PassiveRFNodes_Omni
 uniPassiveRF [..]	 PassiveRFNodes_Uni

Figure 4.7. Zoomed View of Agent Population Elements and Point Node Collections

agent's flight path. As the “flight start” and “flight end” *point nodes* are connected via this *path*, the *UAS* agent will automatically follow along. While it is not observable from Figure 4.5, the flight path begins at elevation 0 and climbs to 10 meters (70 pixels), as specified in Yang's work (p.25). This height is set within the *path* element's *z-axis* coordinate. Additional bends and curves are added to the flight path, as is depicted in Yang's procedures.

Six red colored *point node* elements are added to the model and also the *AcousticNodes_Omni* collection. The acoustic parameters are set within the *omniAcoustic* agent population. Agent populations are similar to making a new interface or construct within a programming language and help link the various

agent classes together through the *Main* agent. The specifics of the acoustic sensor agent are discussed following this section.

As described in Section 3.5.10, several functions are implemented to help the flow of data. In addition to the `placeSensor()` function described earlier, two additional functions were added in order to populate the “Extra Metrics” as labeled in Figure 4.5. The `getUASAltitude()` and `getUASSpeed()` functions dynamically report on the flight altitude and travel speed of the first *UAS* agent in the model. This was added for both, troubleshooting, and useful feedback purposes while the model is operating. Above the “Extra Metrics” lies a button control for adding additional *UAS* agents to the model. The button calls an AnyLogic specific `inject()` function which in turn adds a new *UAS* agent to the *UASagents* population. The model is also configured to load a single agent at the startup and uses the same functionality.

After a new *UAS* agent is added to the model, the “UAS Movement Controls” are called. The `inject()` function mentioned earlier is a function of the *source* control block. This control is called at the use of the `inject()` function and is configured to spawn one *UAS* agent at a time. The control also specifies the location to spawn the new agent, which in this case is the “flight start” *point node*. Immediately following a *UAS* agent spawning via the *source* control, the *UAS* follows along the process line to a *delay* control block. This delay is added to simulate a pilot preparing the UAS for launch. As the *UAS* agent moves into the holding time of the *delay* block, the control sends a message to the *UAS* agent via AnyLogic’s native messaging system. This message (“Ready for launch”) is the catalyst for the *UAS* agent to progress further into its own statechart. This statechart, as well as the other elements of the *UAS* agent are given in Figure 4.8 and described in more detail later in this chapter. As the *UAS* agent leaves the *delay* control block, the *delay* block sends another message to the *UAS* agent (“Launching”). This propels the *UAS* agent into its next state. The *moveTo* control blocks instructs the *UAS* to travel to pre-specified location, whether that be

a marked area, or in this case, the “flight end” *point node*. It is possible to perform the same action with fewer *moveTo* controls, but this method provides flexibility within the model. For example, the *UAS* could spawn in a different area, then move to the start flight node, representing a pre-flight manoeuvre.

The last elements to discuss on the *Main* agent are *agent populations*, *collections* (where the *point node* elements are stored), and the model’s scale. As mentioned before, creating a new agent type in AnyLogic is similar to creating a new class in a programming language. Within the agent type, parameters, variables, functions, graphics, and statecharts are specified, just as they would if developing an traditional object class in computer programming. *Agent populations* are special collections within AnyLogic allowing instances of an agent to be instantiated. It is the *agent population* where agent parameters, as well as the initial number of agents at model runtime, can be specified. It is possible to add agents and modify their parameters programmatically, however an *agent population* must be present. It is also the presence of this element that allows a specific agent type to refer back to the *Main* agent. This linking feature allows the reference of variables or even other agent types, as is done in the *CommunicationNode* agent, from any other agent. *Collections* have been introduced earlier, but to summarize, they are similar to `ArrayList` and are interacted with as such.

Each agent within AnyLogic has a scale. Obviously, a simulation model is ran within a computer, and as such requires a scale by which to relate distances to real-world systems. When an AnyLogic model is ran, the user mostly sees and interacts with the *Main* agent. Also, given that the *Main* agent acts as a central hub connecting all other agents, its scale is the easiest to refer to globally. The scale of each agent can be different between agents and AnyLogic will handle the differences when the two or more agents are placed on the *Main* agent. The scale can be in any imperial or metric length unit and provides a conversion factor of pixels to length unit for conversion purposes. Utilizing this scale allows a model’s internal distance to change, without necessarily requiring the model layout change.

For easier access to the units used in the *Main* agents scale (referred to as “global scale” from here on), a variable called *scaleUnits* was created. The *scaleUnits* variable is used within other agent types to have consistent length conversions. As the SCANS Framework bases distance calculations upon agent positions and parameters, the scale of the graphical representation of agents does not matter. As such, every agent references the *globalScaler* variable to scale their graphical representations to an easily viewed scale. This scaling is set dynamically at model runtime using *15/main.scale.pixelsPerLengthUnit*, which scales every agent as if the *Main* agents scale was set to 15 pixels per 1 length unit. By using scaling in this way, each agent’s size is relative to one another, so the system works best if agents are scaled in a similar manner.

4.1.2 Creating the *UAS* Agent

Possibly the most critical piece in this Framework is the *UAS* agent. Without it, the sensor agents would have nothing to scan for, which causes the communication agents to have nothing to communicate. As with the *Main* agent, the *UAS* agent is made by placing AnyLogic elements and controls from the various palettes. The annotated agent is provided in Figure 4.8. AnyLogic’s 3D helicopter model was chosen as the graphical representation for the *UAS* agent. As this particular model is unable to scale dynamically, an additional circle graphic is placed behind the helicopter, which can be scaled. At model runtime, this circle changes color between blue and white via the *changeDetectionColor* event to aid in locating the *UAS* agent as it is moving. The model scale is set to one meter, which is the same relative size as the sensor and communication agents.

The parameters, variables, and functions are as described in Section 3.5.1, save for one additional variable, *detectionColor*. While the *UAS* agent does not detect anything, the *detectionColor* variable is added to every agent in the model and is used to control the color changing circle below the agent graphic. As every

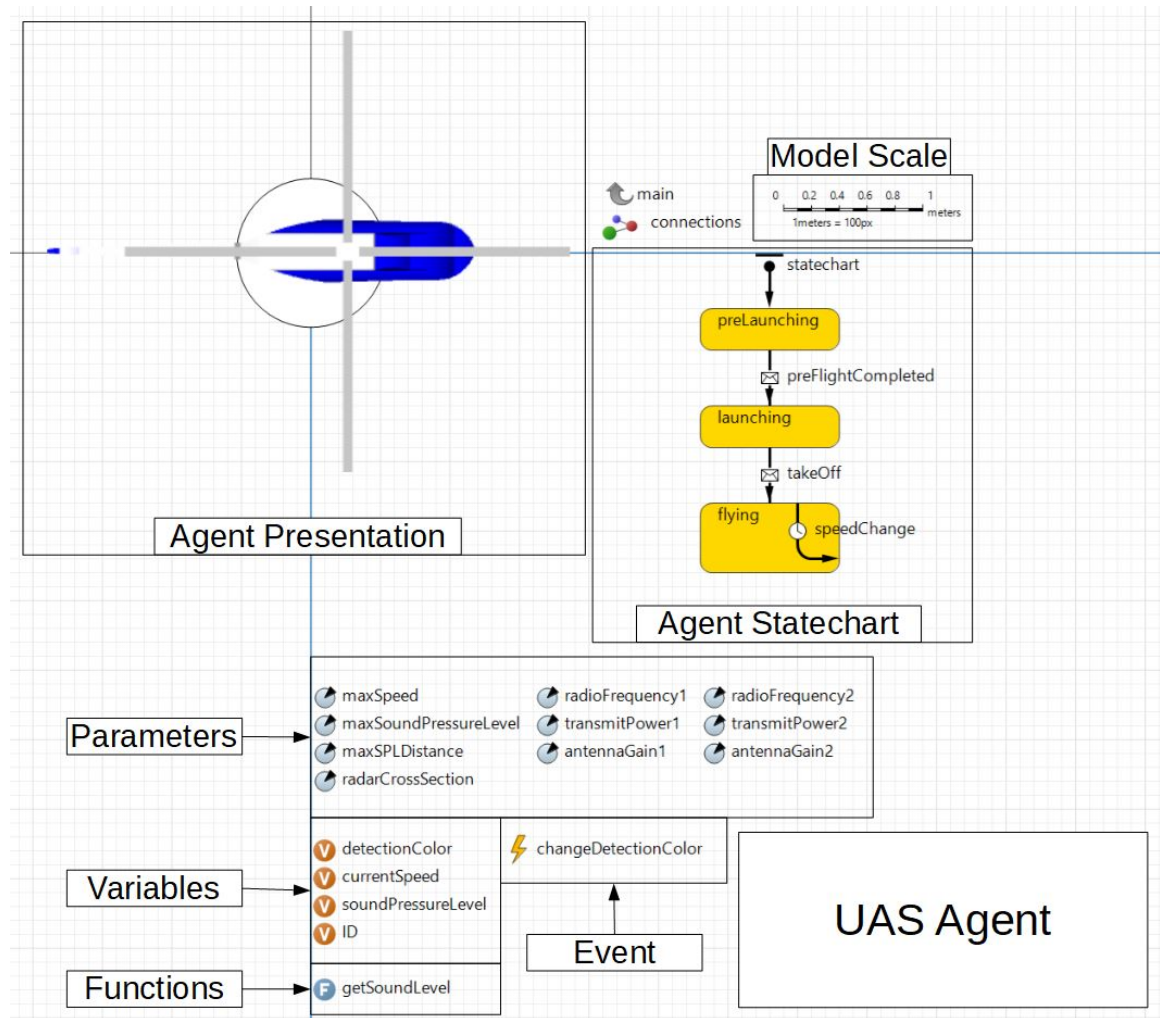


Figure 4.8. Annotated *UAS* Agent

other agent has similar functionality, the variable name is also similar for constancy purposes. The *detectionColor* variable is changed by the *changeDetectionColor* event, which changes the variable data from “blue” to “white” every one half seconds. This creates a slow strobing effect on the *UAS* agent.

When an agent is spawned, each of the agent’s statecharts move into the first state. For the *UAS* agent, this is the *preLaunching* state. As mentioned in the *Main* agent’s *UAS* Movement Controls, this corresponds to the *UAS* entering the *delay* control block. As the agent enters the *delay* block, the “Ready for Launch” message

is received, which triggers the *preFlightCompleted* transition. As the *UAS* leaves the *delay* block, the *takeOff* transition is triggered via the message “Launching”. In this instance, the multiple states are used to change the helicopter model’s color when viewed in 3D space. The *flying* state varies the current speed of the *UAS* agent via the *speedChange* transition. The algorithm for this transition is given in Algorithm 4.1. The *speedChange* transition is triggered every one half seconds. This is done to model potential UAS speed inconsistencies when flying. The speed currently varies by only the top ten percent of UAS’ top speed.

Algorithm 4.1 *speedChange* Transition

```

1: function CHANGESPEEDS
2:   currentSpeed  $\leftarrow$  UNIFORMDISTRIBUTION(maxSpeed * 0.9, maxSpeed)
3:   Set UAS agent’s current speed
4:   GETSOUNDLEVEL

```

4.1.3 Similarities Between Sensor Agents

The following is provided to describe the similarities between sensor agents in a single location. This allows the sections related to individual sensor agents to focus on their unique additions on top of this agent. Aside from the model’s scale, all elements presented in this agent are carbon copied into the more specific sensor agent types.

While every agent is distinguished graphically in the model, they each share the circle graphic, a FOV indicator (blue cone shape), and a text label. The circle changes color between red (not detected) and green (detected) dependent upon whether or not the sensor detects the *UAS* agent. If the *rotates* parameter is *true*, then the FOV indicator is used to signify the direction a sensor is currently pointed. If *rotates* is *false*, this FOV indicator is hidden from view as the sensor is deemed to be omnidirectional. The text label is configured to show the agent’s *ID* as assigned by the `placeSensors()` function in the *Main* agent.

Figure 4.9 provides a view of the shared parameters and features of all sensor agents. All elements shown in this figure are carbon copied to the specific implementations of the sensors. All parameters, variables, and functions are as described in Section 3.5, with the addition of a *detectionColor* variable. This variable is tied to the color property of the circle graphic and is changed between red and green via the *sensing* and *sensing1* states. Each agent also has three functions, *sense()*, *calculateMeasure()*, and *sendAlert()*. Both, *sense()* and *calculateMeasure()*, take an argument of datatype *UAS* agent. Each of these functions differ dependent upon the agent employing them. For that reason, the functions will be described within the following sections for each sensor agent.

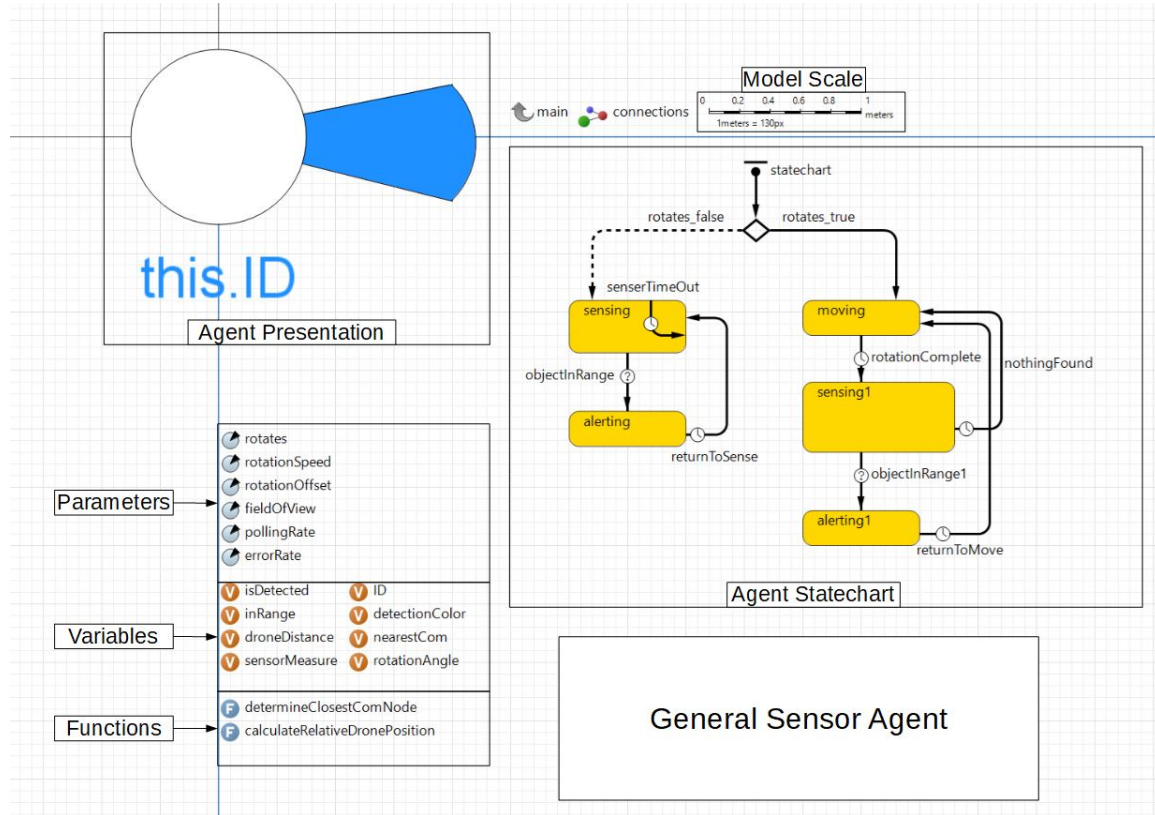


Figure 4.9. Annotated Generalized Sensor Agent

When the sensor agent is spawned into the model, the statechart is triggered automatically. Whether the left side or the right side of the statechart is used

depends upon the *rotates* parameter. If *rotates* is *false*, the sensor is considered to be omnidirectional, the FOV indicator is hidden, and the left branch of the state chart is used. If *rotates* is *true*, the sensor is considered unidirectional, the FOV indicator is made visible, and the right branch is used. The algorithm for the left branch is given in Algorithm 4.2, while the right branch is given in Algorithm 4.3.

Algorithm 4.2 Left Branch of the General Sensor Statechart

```

1: procedure OMNIDIRECTIONAL SENSOR OPERATION
2:   this.FOVIndicator.isVisible(false)
3:   while rotates = false do                                ▷ Agent is considered omnidirectional
4:     isDetected ← false                                       ▷ Enter sensing state
5:     inRange ← false
6:     while inRange = false do                                ▷ Trigger objectInRange transition
7:       for Every UASagent do
8:         drone ← UASagent.get(i)
9:         SENSE(drone)
10:        if isDetected = true & inRange = true then
11:          BREAK
12:        Delay(pollingRate)                                    ▷ Trigger sensorTimeout transition
13:      if no UASagents exist then
14:        detectionColor ← red
15:      SENDALERT                                                ▷ Enter alerting state
16:      Delay(pollingRate)                                       ▷ Trigger returnToSense transition

```

4.1.4 Creating Specific Sensor Agent Types

Most features for all agents are covered in the previous section. However, in addition to those features, each agent type has a specific graphical representation and the data added to the *SensorData* message when an alert occurs is different.

For the *Acoustic* agent, a 3D sphere is used for graphical representation, shown in Figure 4.10. The additional functions shown, **sense()**, **calculateMeasure()**, and **sendAlert()** are as described in Section 3.5.3. For the *Acoustic* agent, the *sensorMeasure* is the noise level generated by the *UAS* agent, and the *measuredUnits* are “dB”.

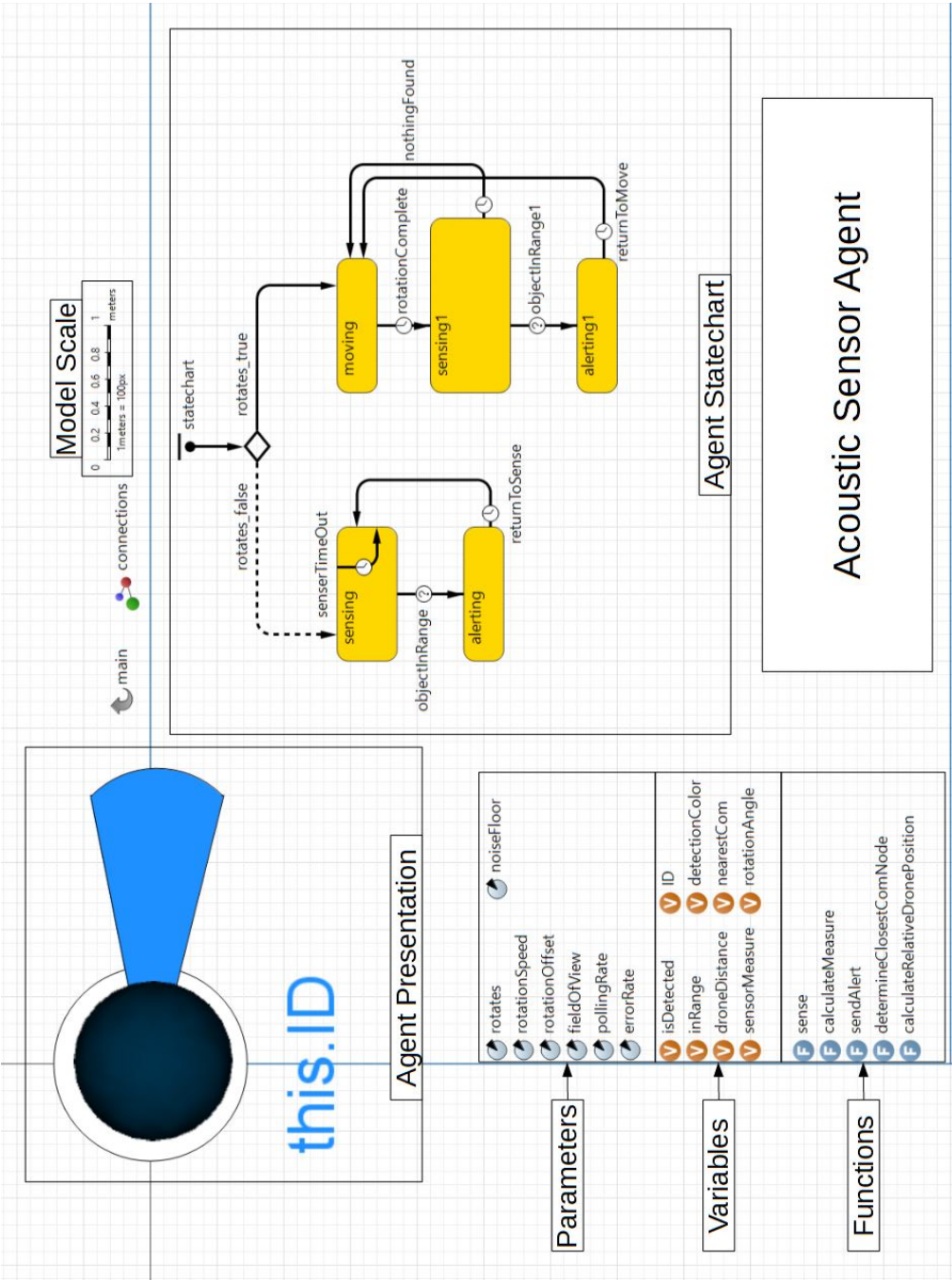


Figure 4.10. Annotated *Acoustic* Agent as Implemented

Algorithm 4.3 Right Branch of the General Sensor Statechart

```

1: procedure UNIDIRECTIONAL SENSOR OPERATION
2:   this.FOVIndicator.isVisible(true)
3:   while rotates = true do                                ▷ Agent is considered unidirectional
4:     while inRange = false do                                ▷ Trigger objectInRange transition
5:       if rotationSpeed! = 0 then                                ▷ Enter moving state
6:         rotationAngle  $\leftarrow$  rotationAngle + (rotationSpeed *  $\frac{\pi}{180}$ )    ▷
AnyLogic uses radians, parameters entered are in degrees
7:         self.setRotation(rotationAngle)
8:         Delay(10ms)                                ▷ Trigger rotationComplete Transition
9:         isDetected  $\leftarrow$  false                                ▷ Enter sensing state
10:        inRange  $\leftarrow$  false
11:        drone  $\leftarrow$  CALCULATERELATIVEDRONEPOSITION
12:        if drone! = null then
13:          SENSE(drone)
14:        else
15:          detectionColor  $\leftarrow$  red
16:          Delay(pollingRate)                                ▷ Trigger nothingFound transition
17:          SENDALERT                                ▷ Enter alerting state
18:          Delay(pollingRate)                                ▷ Trigger returnToSense transition

```

The *Radar* and *Camera* agents are provided in Figures 4.11 and 4.12, respectively. The *Radar* agent is represented by AnyLogic’s 3D radar shape, while *Camer* is represented by a 3D cone shape rotated 180°. The **sense()** and **calculateMeasure()** functions are detailed in Section 3.5.4 and 3.5.5. For both, the **sendAlert()** function records the maximum distance a particular *UAS* agent can be detected and converts that value to match the length units of the *Main* agent. The *main.scaleUnits* length unit is configured as the *measuredUnits*.

For the last of the sensor agents, *PassiveRF* is represented by an un-rotated 3D cone. Similarly to the other sensors, the **sense()** and **calculateMeasure()** functions are listed in Section 3.5.6. Finally, the **sendAlert()** function records the received signal strength from signals generated by the *UAS* agent and records a “dB” unit for the *measuredUnits*. The *PassiveRF* implementation is provided in Figure 4.13.

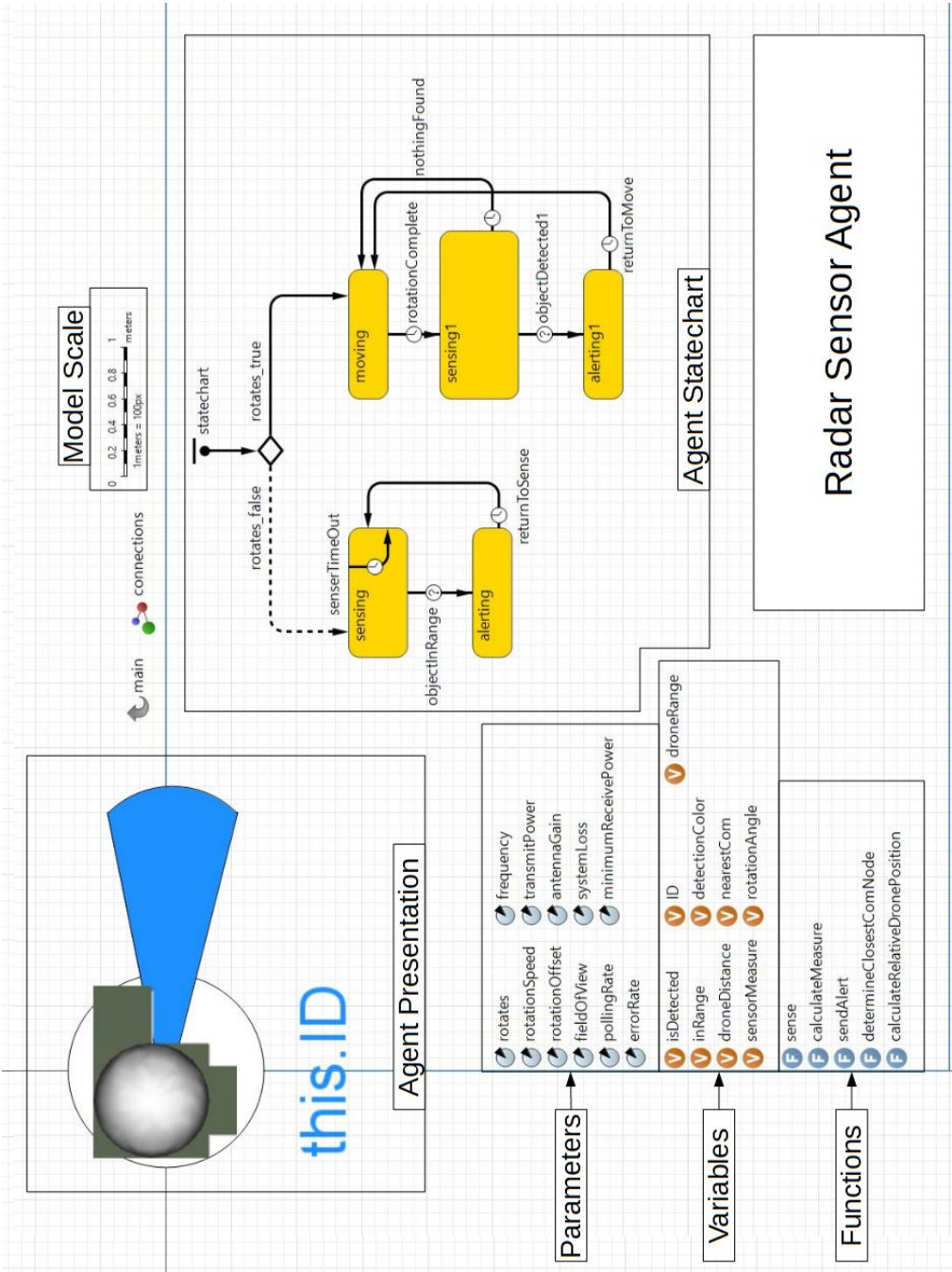


Figure 4.11. Annotated Radar Agent as Implemented

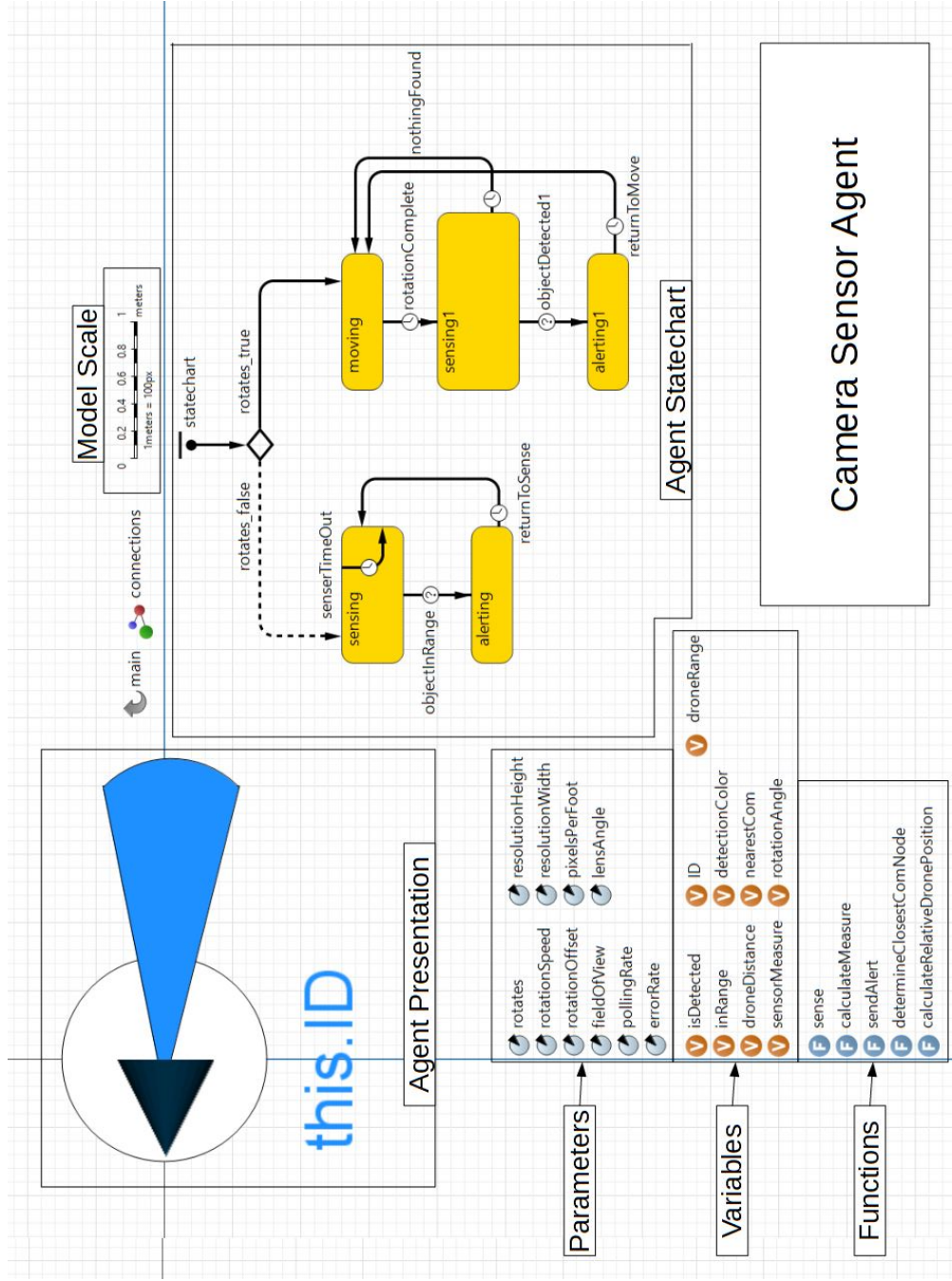


Figure 4.12. Annotated Camera Agent as Implemented

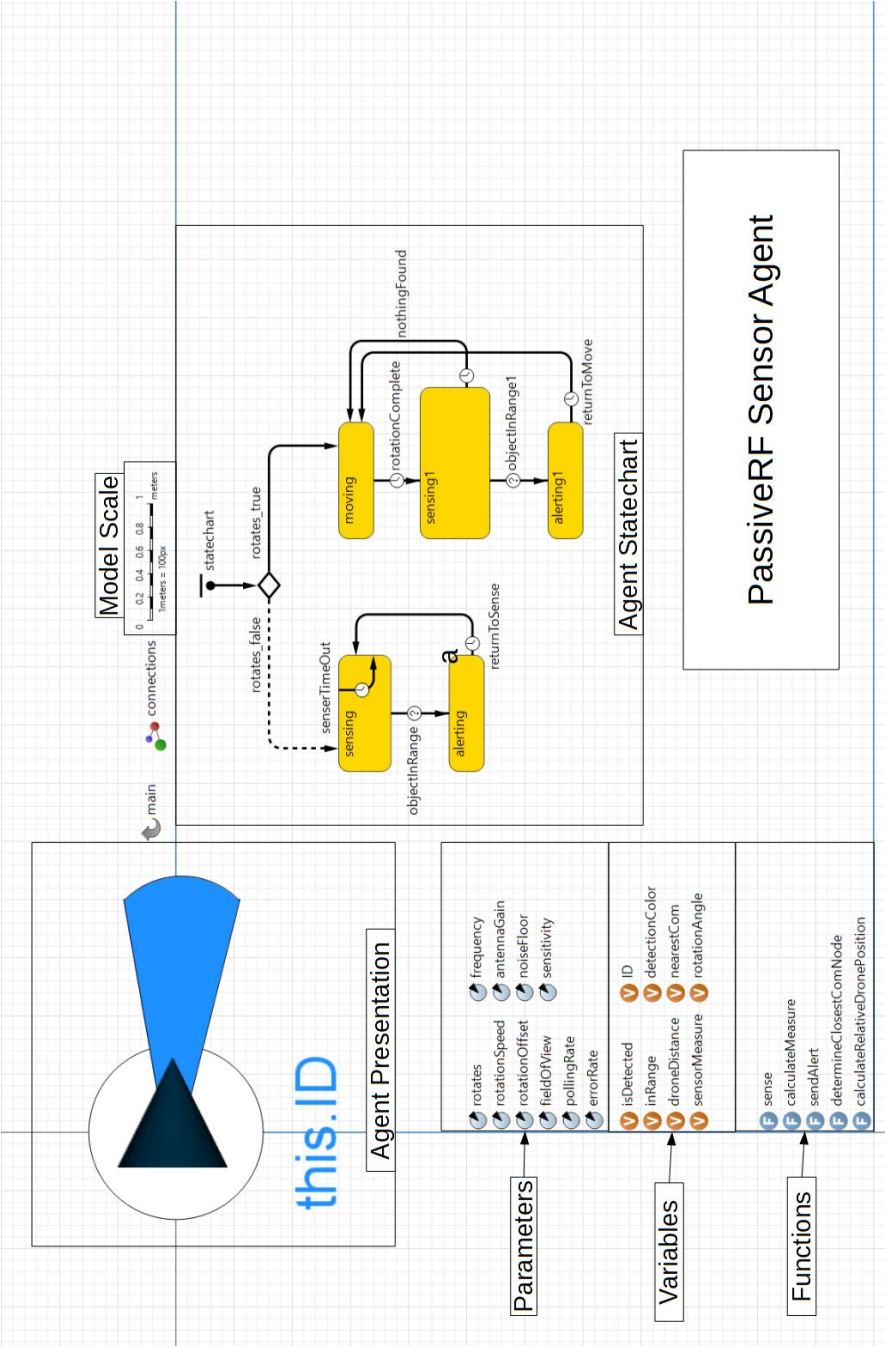


Figure 4.13. Annotated *PassiveRF* Agent as Implemented

4.1.5 Similarities Between Communication Agents

For the two communication device agents, the object class hierarchy (Figure 3.2), suggests many inputs and functions can be shared. Figure 4.14 highlights the shared elements of the *CommunicationNode* and *CommandAndControl* agents. The various parameters and variables used are detailed in Section 3.5.7. The algorithm for the statechart is provided in Algorithm 4.4.

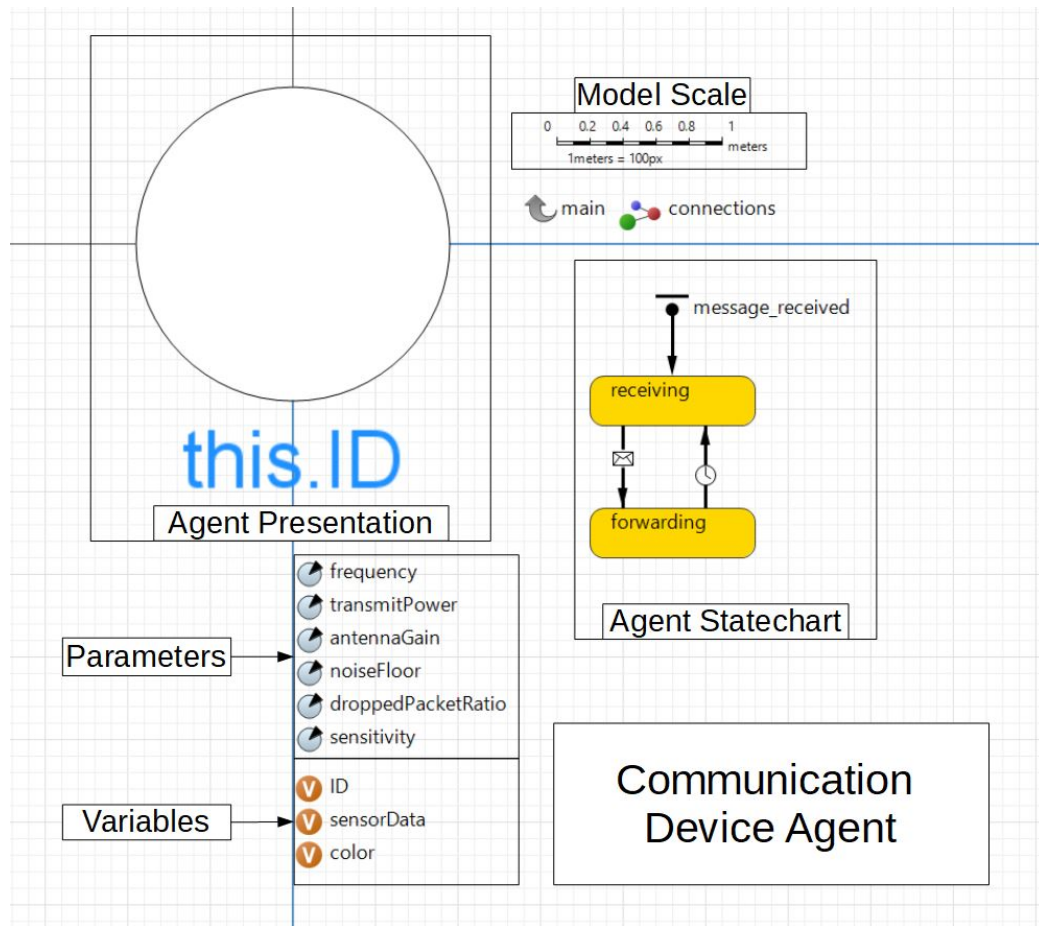


Figure 4.14. Annotated Generalized Communication Device Agent

Algorithm 4.4 General Procedures for Communication Devices

```

1: procedure RECEIVE MESSAGE
2:   while true do
3:     Rest in receiving state until message is received
4:     if messageReceived = true then           ▷ Trigger forwarding transition
5:       if message.getReceived() = true & randomFalse(DPR) then
6:         message.is of type SensorData
7:         message.setLastComNode ← this.ID
8:       else
9:         if message.getReceived() = true then
10:          message.setMessageReceived ← false
11:          message.setLastComNode ← self.ID
12:        sensorData ← message
13:        SENDDATA
14:        Delay(1ms)           ▷ Return back to the receiving state to wait for a new
                               message

```

4.1.6 Creating Specific Communication Agent Types

The *CommunicationNode* agent is represented by a 3D, upright pyramid, which when viewed from above, looks like a square. This representation is provided in Figure 4.15. While both the *CommunicationNode* and the *CommandAndControl* agents have a `sendData()` function, each performs a different task. The specific implementation for the *CommunicationNode* is given in Algorithm 4.5. The purpose of the function is to pass data onward to either a *CommandAndControl* agent, or another *CommunicationNode* agent. The *color* variable is used to turn the circle graphic of the agent between red and green when a message is received. It acts similarly to a link light on a computer's internet (LAN) port.

Algorithm 4.5 Sending Data Onward from a *CommunicationNode* Agent

```

1: function SENDDATA
2:   if nearestCCNode = null then
3:     SEND(sensorData, nearestNeighbor)   ▷ send() is an AnyLogic-specific
4:     messaging function
5:   else
6:     SEND(sensorData, nearestCCNode)

```

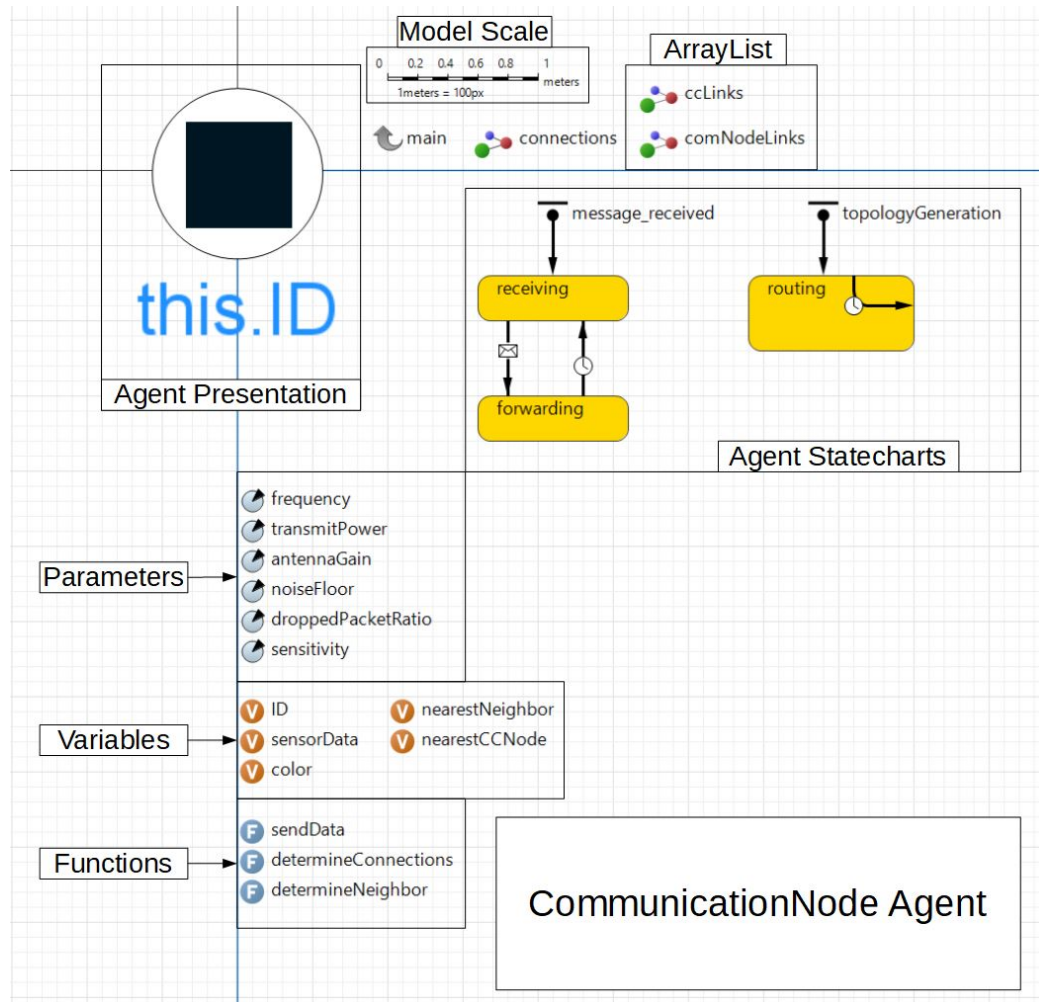


Figure 4.15. Annotated *CommunicationNode* Agent as Implemented

An item unique to the *CommunicationNode* agent is the *topologyGeneration* statechart. This statechart determines the communication route topology for sending messages between agents based on the `determineConnections()` and `determineNeighbor()` functions. Both of these functions are defined in Section 3.5.8. The algorithm for the *topologyGeneration* statechart is given in Algorithm 4.6.

Finally, the last agent in the model, the *CommandAndControl* agent, has the important job of recording any received messages. The agent is represented by AnyLogic's 3D representation of a factory, and is shown in Figure 4.16. The functional difference between this agent and the *CommunicationNode* agent is the

`sendData()` function. The function's purpose is to write *SensorData* messages to both the AnyLogic console and AnyLogic's built in database. The database table structures used in this model are provided in Appendix B.

Algorithm 4.6 Generate the Routing Topology for *CommunicationNode* Agents

- 1: **procedure** TOPOLOGYGENERATION
 - 2: DETERMINECONNECTIONS
 - 3: DETERMINENEIGHBORS
 - 4: Delay(1s)
-

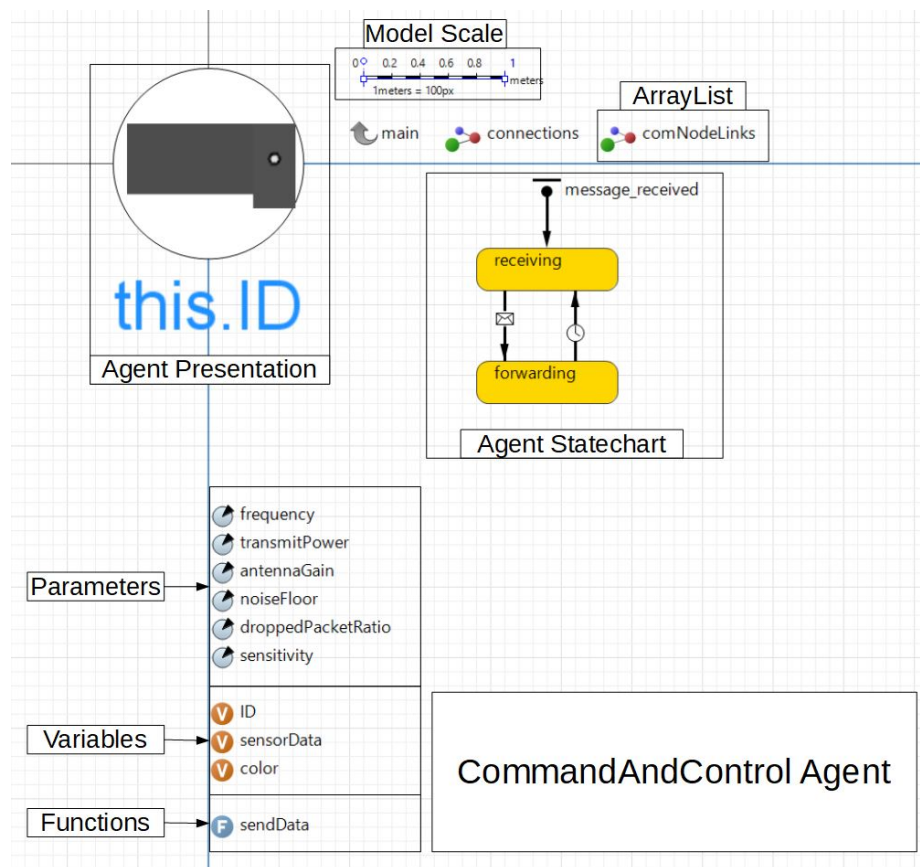


Figure 4.16. Annotated *CommandAndControl* Agent as Implemented

4.2 Summary

This chapter described the results of implementing the SCANS Framework within AnyLogic. While the desired object inheritance properties were not able to be used strictly, both the general sensor agent and the general communication device agent were provided to highlight the similarities between the specific agent types. This chapter also provided descriptions for the agents' statecharts which dictate their processing and movement. Additionally, the work of Yang (2019) was used as a reference when building the *Main* agent as it met many of the features the SCAN Framework makes use of including the use of acoustic sensors, wireless networking, and a command and control station.

In the following chapter, several models will be created based on experiments performed by sensor and CUAS researchers to test the SCANS Framework accuracy against empirical evidence. This process will act as the validation for the SCANS Framework.

CHAPTER 5. USING THE SCANS FRAMEWORK

Within this chapter, the AnyLogic implementation of the SCANS Framework, as detailed in Chapter 4, will be used to recreate published, empirical experiments based on sensor detection and CUAS systems. These tests will be used to show the SCANS Framework’s accuracy and ability to model the detection of a UAS. As mentioned in Section 3.6, each created model will be ran three times and a detection chart will be generated using `pColorMesh` in Python’s `matplotlib` package.

Examples of the model output are given in Figures 5.1 - 5.3. The *model_runs* table captures the start and end times of each model run (distinguished by pressing the play and stop buttons within AnyLogic), the number of each agent type within the model, and an optional name to distinguish model runs. The name is entered at on the simulation’s landing page. The second table, *sensors*, records every agent types parameters, including communication devices but not *UAS* agents, at model runtime. The third and final table, *sensor_data*, is the table to which *CommandAndControl* agents record the data received from sensors.

	start_time	end_time	acoustic_omni	acoustic_uni	radar_omni	radar_uni	visual_camera	infrared_camera	com_nodes	command_nodes	run_name
1	2020-11-09T17:00:...	2020-11-09T17:...	6	0	0	0	0	0	1	1	Yang50R10A: Trial 1
2	2020-11-09T17:00:...	2020-11-09T17:...	6	0	0	0	0	0	1	1	Yang50R10A: Trial 2
3	2020-11-09T17:01:...	2020-11-09T17:...	6	0	0	0	0	0	1	1	Yang50R10A: Trial 3
4	2020-11-09T17:04:...	2020-11-09T17:...	3	0	0	2	6	0	2	1	Hello World
5	2020-11-09T17:07:...	2020-11-09T17:...	3	0	0	2	6	0	2	1	Attempt 3729, Take 2.
6	2020-11-09T17:10:...	2020-11-09T17:...	3	0	0	2	6	0	2	1	Please let this work...

Figure 5.1. Example of Records Created Each Model Run

	id	param_name	param_value	real_time
1	CC-0	frequency	2400	2020-11-08T16:4...
2	CC-0	transmitPower	30.0	2020-11-08T16:4...
3	CC-0	noiseFloor	-80.0	2020-11-08T16:4...
4	CC-0	droppedPacketRatio	0.02	2020-11-08T16:4...
5	CC-0	antennaGain	2.5	2020-11-08T16:4...
6	CC-0	sensitivity	-76.0	2020-11-08T16:4...
7	CN-0	frequency	2400	2020-11-08T16:4...
8	CN-0	transmitPower	30.0	2020-11-08T16:4...
9	CN-0	noiseFloor	-80.0	2020-11-08T16:4...
10	CN-0	droppedPacketRatio	0.02	2020-11-08T16:4...
11	CN-0	antennaGain	2.5	2020-11-08T16:4...
12	CN-0	sensitivity	-76.0	2020-11-08T16:4...
13	AO-0	noiseFloor	44.0	2020-11-08T16:4...
14	AO-0	rotates	false	2020-11-08T16:4...
15	AO-0	rotationSpeed	30.0	2020-11-08T16:4...
16	AO-0	rotationOffset	0.0	2020-11-08T16:4...
17	AO-0	fieldOfView	0.0	2020-11-08T16:4...
18	AO-0	pollingRate	249.25546190815982	2020-11-08T16:4...
19	AO-0	errorRate	0.05	2020-11-08T16:4...
20	AO-1	noiseFloor	44.0	2020-11-08T16:4...
21	AO-1	rotates	false	2020-11-08T16:4...
22	AO-1	rotationSpeed	30.0	2020-11-08T16:4...
23	AO-1	rotationOffset	0.0	2020-11-08T16:4...
24	AO-1	fieldOfView	0.0	2020-11-08T16:4...
25	AO-1	pollingRate	245.6036132545907	2020-11-08T16:4...
26	AO-1	errorRate	0.05	2020-11-08T16:4...

Figure 5.2. Record of Each Agent and Parameters Used in Every Model Run

5.1 Acoustic Experiments

Much of the setup for this experiment was discussed in Chapter 4. The model examined Yang’s first experiment, where the acoustic nodes were placed at a 50 meter radius from the central network node. The UAS had a flight space of 20 meters and an altitude of 10 meters. Based on the radius measurement and the altitude, the model will be referred to as “Yang50R10A”. The model was set to fly the *UAS* agent from the start node, to the end node, and back to the start. The model layout is provided in Figure 5.4. Logged data is shown in the console on the right-handed pane of the figure. The flight completion times for Yang50R10A are provided in Table 5.1. According to the DJI published specification sheet, the

	originating_sensor_id	detected	in_range	distance	measure	sensor_units	model_time	real_time	message_received	last_com_node
1	AO-0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	17.989	44.213 dB		10	2020-11-09T13:02:33.175993800	<input type="checkbox"/>	AO-0
2	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.174	44.819 dB		11.243	2020-11-09T13:02:34.603774	<input checked="" type="checkbox"/>	CC-0
3	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.416	48.463 dB		11.737	2020-11-09T13:02:35.174035	<input checked="" type="checkbox"/>	CC-0
4	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.418	47.081 dB		12.23	2020-11-09T13:02:35.761918900	<input checked="" type="checkbox"/>	CC-0
5	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.037	45.357 dB		12.723	2020-11-09T13:02:36.335660500	<input checked="" type="checkbox"/>	CC-0
6	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.658	46.429 dB		13.216	2020-11-09T13:02:36.906760800	<input checked="" type="checkbox"/>	CC-0
7	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.642	48.374 dB		13.71	2020-11-09T13:02:37.492617900	<input checked="" type="checkbox"/>	CC-0
8	AO-0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	15.349	44.466 dB		14.703	2020-11-09T13:02:38.644670500	<input checked="" type="checkbox"/>	CC-0
9	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	15.396	44.466 dB		15	2020-11-09T13:02:38.986666900	<input checked="" type="checkbox"/>	CC-0
10	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.23	44.629 dB		15.75	2020-11-09T13:02:39.877567200	<input checked="" type="checkbox"/>	CC-0
11	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.028	47.032 dB		16.249	2020-11-09T13:02:40.472754600	<input checked="" type="checkbox"/>	CC-0
12	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.667	45.692 dB		17.249	2020-11-09T13:02:41.641024900	<input checked="" type="checkbox"/>	CC-0
13	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.677	44.141 dB		17.748	2020-11-09T13:02:42.234375100	<input checked="" type="checkbox"/>	CC-0
14	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.104	46.481 dB		18.248	2020-11-09T13:02:42.829209800	<input checked="" type="checkbox"/>	CC-0
15	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.458	45.742 dB		18.747	2020-11-09T13:02:43.424908	<input checked="" type="checkbox"/>	CC-0
16	AO-1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	12.711	47.744 dB		19.247	2020-11-09T13:02:44.012176600	<input checked="" type="checkbox"/>	CC-0
17	AO-2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	17.553	44.944 dB		19.25	2020-11-09T13:02:44.015266400	<input checked="" type="checkbox"/>	CC-0
18	AO-2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.901	46.099 dB		21.246	2020-11-09T13:02:46.255031	<input checked="" type="checkbox"/>	CC-0
19	AO-2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	11.933	47.642 dB		21.741	2020-11-09T13:02:46.812967600	<input checked="" type="checkbox"/>	CC-0
20	AO-2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.207	44.846 dB		22.237	2020-11-09T13:02:47.399048100	<input checked="" type="checkbox"/>	CC-0
21	AO-2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10.468	46.881 dB		22.732	2020-11-09T13:02:47.967045	<input checked="" type="checkbox"/>	CC-0

Figure 5.3. Sensor Data Recorded by *CommandAndControl* Agents

Phantom 2 is capable of speed up to 15 m/s, however this is not recommended (DJI, n.d.). As such, the speed of the *UAS* agent was set to 12 m/s, or 80% of the specified top speed. Yang’s work did not specify the speed. The default SPL for the model (88.5 dB) was used as Yang did not specify otherwise. The 88.5 dB value was measured by Cabell et al. (2016) for the DJI Phantom 2 and was used by Torija et al. (2020) in their study. The *noiseFloor* for the environment was configured for 44 dB, the parameter’s default value, as this appeared in line with Yang’s reports (p.44).

Table 5.1.

Model Completion Time for Yang50R10A

Trial	Model Run Time (s)
1	47.8
2	42.8
3	43.8
Average	44.6

The detection graphs of the three trial runs are provided in Figure 5.5. Only records where the *UAS* was detected and the *SensorData* message was received successfully by the *CommandAndControl* agent are shown in the graphs. These are considered the true positives. Individual detections are marked in yellow, while purple markings represents the sensor not detecting the *UAS*. This non-detection of the *UAS* agent represents both the false negatives and the true negatives of the system. The x-axis is the number of data points recorded, ordered by timestamp. These results are consistent with Yang’s findings, given the known variables, even though Yang’s data was processed using machine learning tools.

In order to understand the affects of a lesser SPL value, the *soundPressureLevel* of the *UAS* agent was set to 75 dB, a significantly lower value typical of a DJI Phantom 4 DJI (n.d.). This resulted in a clear path taken by the *UAS* over the *Acoustic* agents. This result is included with the original three tests, shown in Figure 5.5(d), for comparative purposes.

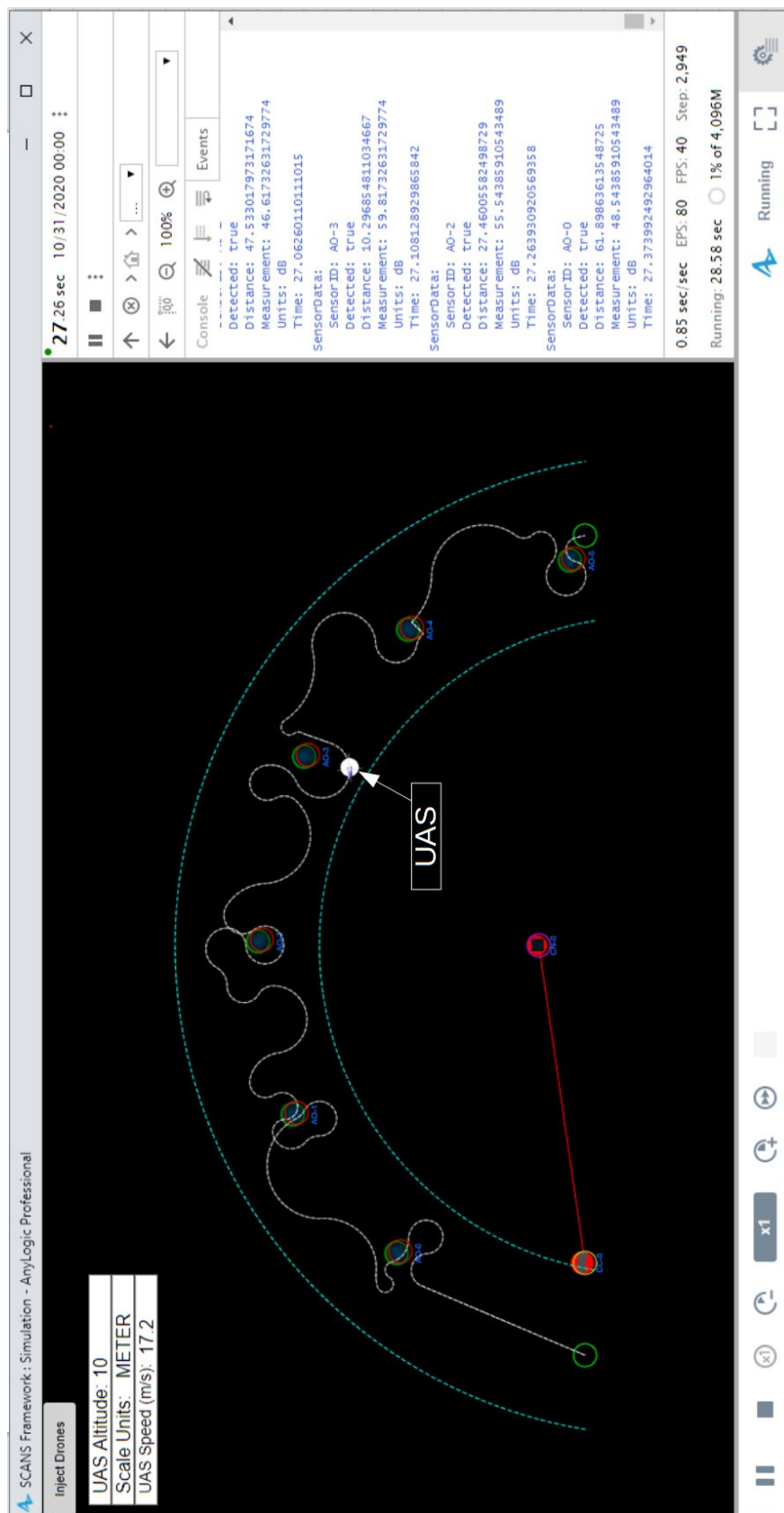


Figure 5.4. Yang's First Experiment: Yang50R10A

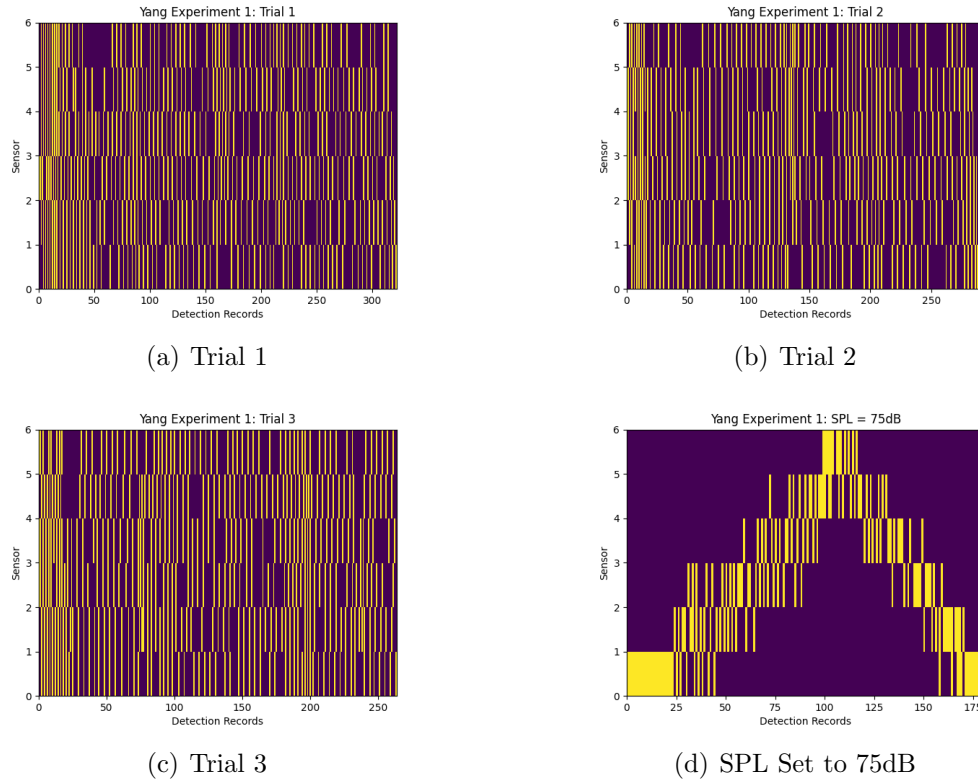


Figure 5.5. Yang50R10A Experiment 1 Detection Graph Results

According to the Equation 2.14, a SPL value of 88.5 dB will meet the noise floor threshold of 44 dB at approximately 150 meters, or 490 feet. This 150 meter measure, is consistent with the data presented in Section 2.1.1 by Busset et al. (2015), Hauzenberger and Holmberg Ohlsson (2015), and Mezei and Molnar (2016). For comparison, a SPL source value of 75 dB becomes 44 dB at approximately 32 meters, or 105 feet. This is also consistent with the detection graphs of Figure 5.4, where the *Acoustic* nodes were placed approximately 25 meters apart and a clear path of travel is shown.

Tables 5.2 - 5.4 provide several statistics from the three trials. Each table provides the total number of records, the true positives and false negatives generated by the system, as well as the average SPL measurement and the standard deviation of the SPL measurement. The tables calculate each metric for the

individual sensors, while the “Total” column encompasses all sensors. From the *sensor_data* database table, a true positive value is determined when both the *detected* and *message_received* fields are *true*. A false negative occurs when the *detected* field is *false*, but the *in_range* field is *true*. True negatives can be calculated by dividing the model run by the *Acoustic* agent’s *pollingRate* and subtracting the total records field from the results tables. As mentioned before, false positives are not possible in this framework.

Table 5.2.

Results of Experiment Yang50R10A: Trial 1

Acoustic Sensors	AO-0	AO-1	AO-2	AO-3	AO-4	AO-5	Overall
Total Records	84	87	86	88	82	75	502
True Positives	75	79	81	79	75	68	457
False Negatives	7	5	5	6	5	2	30
Average Measure	54.1	54	54.29	52.9	52.06	50.78	53.08
σ	6.58	5.59	4.93	4.89	5.01	4.93	5.47

Table 5.3.

Results of Experiment Yang50R10A: Trial 2

Acoustic Sensors	AO-0	AO-1	AO-2	AO-3	AO-4	AO-5	Overall
Total Records	73	76	76	76	75	69	445
True Positives	64	66	74	70	70	62	406
False Negatives	5	4	0	4	4	4	21
Average Measure	54.29	54.31	54.63	54	52.57	51.69	53.61
σ	6.68	5.09	5.18	5.45	5.85	5.39	5.69

Table 5.4.

Results of Experiment Yang50R10A: Trial 3

Acoustic Sensors	AO-0	AO-1	AO-2	AO-3	AO-4	AO-5	Overall
Total Records	77	79	78	77	74	69	454
True Positives	71	75	69	69	70	60	414
False Negatives	2	1	4	2	4	5	18
Average Measure	54.67	55.07	54.98	54.2	53.06	51.55	53.99
σ	6.18	5.28	5.48	5.53	5.61	5.95	5.77

5.2 Radar Experiments

The work by Farlik et al. (2019) extensively looks at radar, passive RF scanning, acoustic, and electro-optical sensors in the CUAS space, providing valuable measurements in regards to sensor performance and consumer UAS. Farlik et al. were able to show through simulation, and then with field experiments, the radar cross section (RCS) of a DJI Phantom 2, Phantom 4, and a 3DR Y6. Using the average RCS for a DJI Phantom 2 (0.33 m^2), a radar experiment can be constructed within AnyLogic to determine possible ranges of detection. No range testing was performed by Farlik in regards to radar systems. However, Farlik et al. provides in their study, a map of the UAS flight path, radar location, and a UAS spotter location. It is this figure with which the SCANS model was made. For comparison purposes, the map graphic from Farlik et al. is provided as Figure 5.6 and the SCANS Framework implementation is provided in Figure 5.7. The scales of each are matched as closely as possible with the scale bar on Farlik’s figure, which reads 5000 meters. From the terrain map’s scale, the UAS flew between 600 – 900 meters at the closest observation position. Given the 5 km distance and the use of the 10 GHz frequency, the AnyLogic model will be referred to as Farlik5K10G.

In total, three unidirectional radar nodes are placed in the model; two on the left and one on the right side of the flight path. The flight path is the white dotted line in the north/south direction. The chained agents across the bottom of the model are *CommunicationNode* agents. A blue connecting line indicates the traffic is passed from one *CommunicationNode* to another, while a red connecting line shows the messages are passed directly to the *CommandAndControl* agent. The *CommandAndControl* agent is the third agent from the right.

Farlik et al. used several Czech Republican military radar systems for their field experiments, however no performance metrics concerning the radar systems were published with their work. One such system was the ReVisor system, developed by Retia Incorporated. A request has been made for basic specification

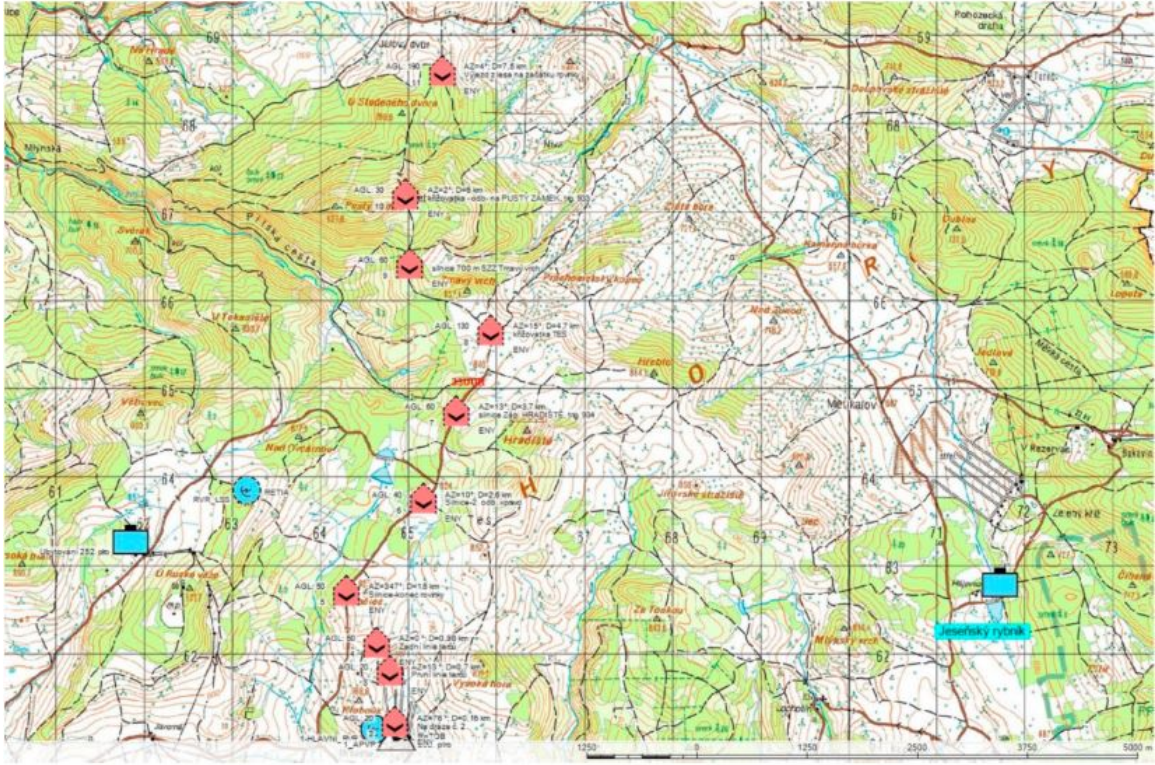


Figure 5.6. Flight Path and Radar Placements of Farlik et al. (2019, p.6)

details pertaining to the ReVisor radar, however a reply from ReVisor has not been received at the time of writing.

An iterative process was completed several times with the radar at the 10 GHz frequency, and testing began with the settings listed in Table 5.5. With a UAS RCS of 0.33 m^2 , the *Radar* agents calculated the maximum distance the *UAS* would be detectable to be 75.48 meters. Testing continued with the values given in Table 5.6. The strongest transmit power and lowest received power tested obtained a maximum calculated range for detection of 754.83 meters.

In an effort to understand the effects of parameters on calculated distance, the model was run with a sub-gigahertz radar frequency of 915 MHz, 70 dBm transmit power, and a minimum received power of -90 dBm. For an object with an RCS of 0.33 m^2 , this resulted in a maximum potential detection distance of 2495.41

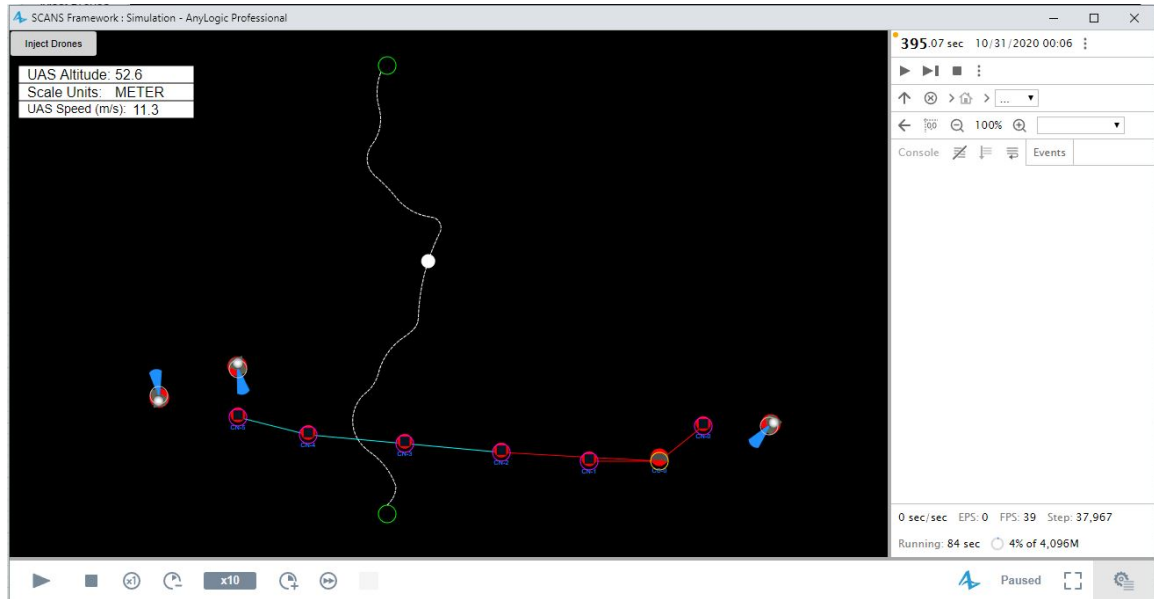


Figure 5.7. Radar Sensor Evaluation Based on Farlik et al. (2019)

meters. The problem with this frequency, however is the 915 MHz channel has a wavelength of 32.7 cm, which would be too long to provide meaningful detection of a small UAS. For comparison, a 10 GHz signal has a wavelength of 2.9 cm and a DJI Phantom 2 is 35x35x18 cm (LxWxH), including the propellers (DJI, n.d.).

Table 5.5.

Input Parameters For Experiment Farlik5KM10G

Parameter	Value
transmitPower	1000 W (60 dBm)
minimumReceivePower	0.000000001 W (-60 dBm)
frequency	10 GHz
antennaGain	32 dB
systemLoss	0.5 dB

Given these unexpected shortened distances, an additional test was performed to ensure the *Radar* agent was calculating the distance properly. A second radar model, based upon Park et al. (2020), was made using the detailed specifications for their self-made radar system. The AnyLogic model is given in

Table 5.6.

Iterative Tests for the Farlik5K10G Model

transmitPower (dBm)	minimumReceive Power (dBm)	Potential Detection Range (m)
60	-60	75.48
70	-60	134.23
60	-90	424.47
70	-90	754.83

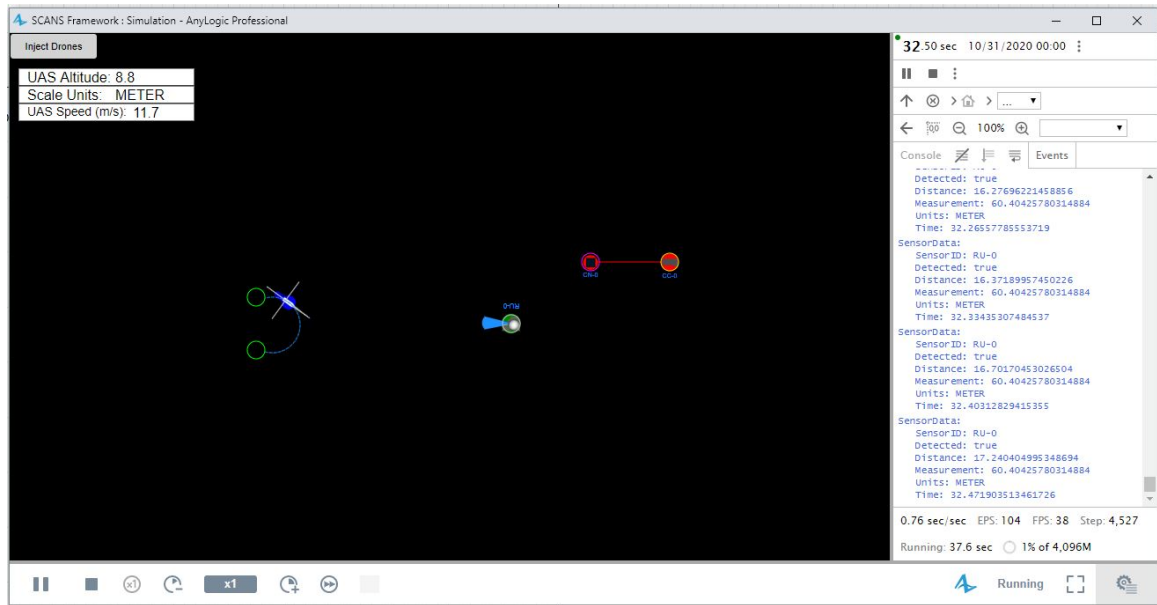


Figure 5.8. Radar Model Based on Parameters of Park et al. (2020)

Figure 5.8. Using the data provided by Park et al., Table 5.7 list the various *Radar* input parameters of 1 W, using the 2.4 GHz ISM band, and according to their results a minimum received power of approximately -80 dBm, or 0.00000000001 W (p.15). Park et al. record that the radar detected the DJI Phantom UAS at 17 meters.

The results of the experiment showed the *Radar* agent capable of detecting the *UAS* agent at a range of 60.4 meters. This is a concerning finding as Park et al. reported the DJI Phantom UAS was only detected up to 17 meters, while their flight path covered a 40 meter space directly in front of the radar. This could

Table 5.7.

Radar Agent Input Parameters for Park et al. (2020)

Parameter	Value
transmitPower	1 W
minimumReceivePower	0.00000000001 W (-80 dBm)
frequency	2.44 GHz
antennaGain	10 dBi
systemLoss	1 dB
fieldOfView	30°

potentially be explained by their system having a high amount of system loss, or if there was a high amount of environmental noise.

The only mention of system loss Park et al. mentioned was due high amounts of heat generated by the system, however this was addressed by adding a heat block to the bottom of the radar components. In terms of environmental noise, Park et al. measured between approximately -9 and -45 dBm. This is presumably caused by Wi-Fi systems in the area operating on the same frequency. By configuring the model’s *systemLoss* parameter to 46 dBm (environmental noise + 1 dB of system loss), the potential detection distance for the *Radar* agent becomes 16.8 meters. This is inline with the experiment of Park et al.

5.3 Camera Experiments

Farlik et al. also tested the capabilities of the ReTOB military optical air surveillance device. The ReTOB uses a MATIS HH thermal camera and has an angle of view of 9 x 6° (WxH) (Farlik et al., 2019, p.18) and a picture resolution of 384x256 (Broekaert & Budin, 2003, p.234). Farlik et al. also used a second device called FLIR ThermoVision™ A40M/Researcher. This IR camera has an angle of view of 24 x 18° (WxH) and a resolution of 320x240. The team found the FLIR thermal imaging camera capable of UAS detection at 140 meters, while the MATIS HH camera detected UAS at 1.8 km. It is important to note however that for this

test, Farlik et al. used a DJI Spreading Wings S900 hexacopter, which is over six times larger than the DJI Phantom 2 UAS other studies have been using. The specifications for the S900 are provided in Table 2.1.

As the *Radar* agent and *Camera* agent function similarly, at least in the SCANS Framework, the model layout has been reused from Figure 5.8. The *Radar* agent in the center of the model was replaced with an IR *Camera* agent and the scale of the model was increased from 20 meters to 150 meters for the FLIR camera test and to 1800 meters for the MATIS HH test.

From Farlik et al., each camera’s FOV is given. As this is given, neither the resolution of the cameras, nor the px/ft (PPF) threshold for UAS detection is needed. Using Equations 2.4 and 2.3, the lens angles for each imaging solution can be calculated and these parameters can be used in the model to verify proper operation. If the model’s calculations are functioning properly, then the MATIS HH test should return a maximum detection range of 1.8 kilometers and the FLIR test should return 140 meters. The results of these calculations are recorded in Table 5.8.

Table 5.8.

Infrared Camera Agent Tests For Farlik et al. (2019)

Parameters	MATIS HH	FLIR ThermoVision
fieldOfView	9	24
lensAngle	0.087	3
Max Distance	1.806 km	139.71 m

While not a complicated evaluation, the success shown in Table 5.8 helps to verify the calculations performed by the *Camera* agent. The *Camera* agent also signaled a detection only when the *UAS* agent was properly within the designated FOV. Verification of the *Camera* agent can be taken further by calculating the PPF ensuring Equation 2.2 has been implemented properly, as well. The PPF is calculated by dividing the resolution width by FOV. That results in a PPF of 42.66 for the MATIS HH and 13.33 for the FLIR. The parameters entered and the calculated maximum detection ranges are recorded in Table 5.9.

Table 5.9.

Infrared Camera Agent Tests Using Calculated PPF

Parameters	MATIS HH	FLIR ThermoVision
resolutionWidth	384	320
lensAngle	0.087	3
pixelsPerFoot	42.66	13.33
Max Distance	1.806 km	139.71 m

Compared to Qi et al. (2018), referenced in Section 2.1.2, with a PPF of 4 and a camera resolution of 460 x 520, the PPF for the MATIS HH and FLIR cameras appeared too large. However, given the area of the side profile for the DJI S900, the pixels required per camera are only 0.144% of the total resolution for the MATIS HH, and 0.057% for the FLIR camera. For comparison, Qi et al. used a DJI Phantom 2 and machine learning algorithms for a detection range of 87 meters. This requires 0.003-0.0038% of the image’s resolution.

5.4 PassiveRF Experiments

DeDrone has become a popular company in the CUAS space with their DroneTracker software. Farlik et al. (2019) also tested this system and using DeDrone’s now deprecated passive RF scanner, the RF-100, tested DeDrone’s claimed 2 kilometer detection range. Farlik et al. measured positive detections for the DJI Phantom 2 and the 3DR Y6 at up to 1.7 kilometers. This is theoretically accomplished provided the UAS transmitted with the 2.4 GHz frequency, at 100 mW of power and a 2.5 dBi antenna. The RF-100 would then need only a 5 dBi antenna, if the environmental noise floor was -80 dB, which it appears to have in DeDrone’s published images. The specifications for DeDrone’s antenna could not be found. The assumptions listed for the UAS radio transmission are common within the wireless network space, even on deprecated hardware such as the Cisco 1242 access point (Cisco, 2009).

Figure 5.9 shows the test model and Table 5.10 provides the input parameters and results of the test. As expected, with the stated parameters, RF detection was possible at the 2 km mark, with a noise floor of -80 dB. This result, and the results shown in the right pane of Figure 5.9, verify the calculations are performing as expected. However, the value is generous due to the lack of system noise.

A more conservative *noiseFloor* value for simulation purposes may be +3 dB to any given value. This would place the *noiseFloor* at -77 dB, which in turn lessens the detection distance to 1670 meters. A noise floor of -78 dB lessens the distance to 1875 meters, which may still be too generous according to the results of Farlik et al. (p.13). The DJI MAVIC Pro UAS was unable to be detected at 1740 meters, but was able to be both detected and identified at 1400 meters. This could indicate that a detection distance near 1670 meters exists at the specified -77 dB noise floor.

Table 5.10.

Input Parameters for PassiveRF and UAS Agent

Agent	Parameter	Value
UAS	transmitPower1	20 dBm
UAS	radioFrequency1	2400 MHz
UAS	antennaGain1	2.5 dBi
PassiveRF	frequency	2400 MHz
PassiveRF	antennaGain	5 dBi
PassiveRF	noiseFloor	-80 dBm
PassiveRF	sensitivity	-80 dBm
Result	RSS (2075m)	-78 dBm

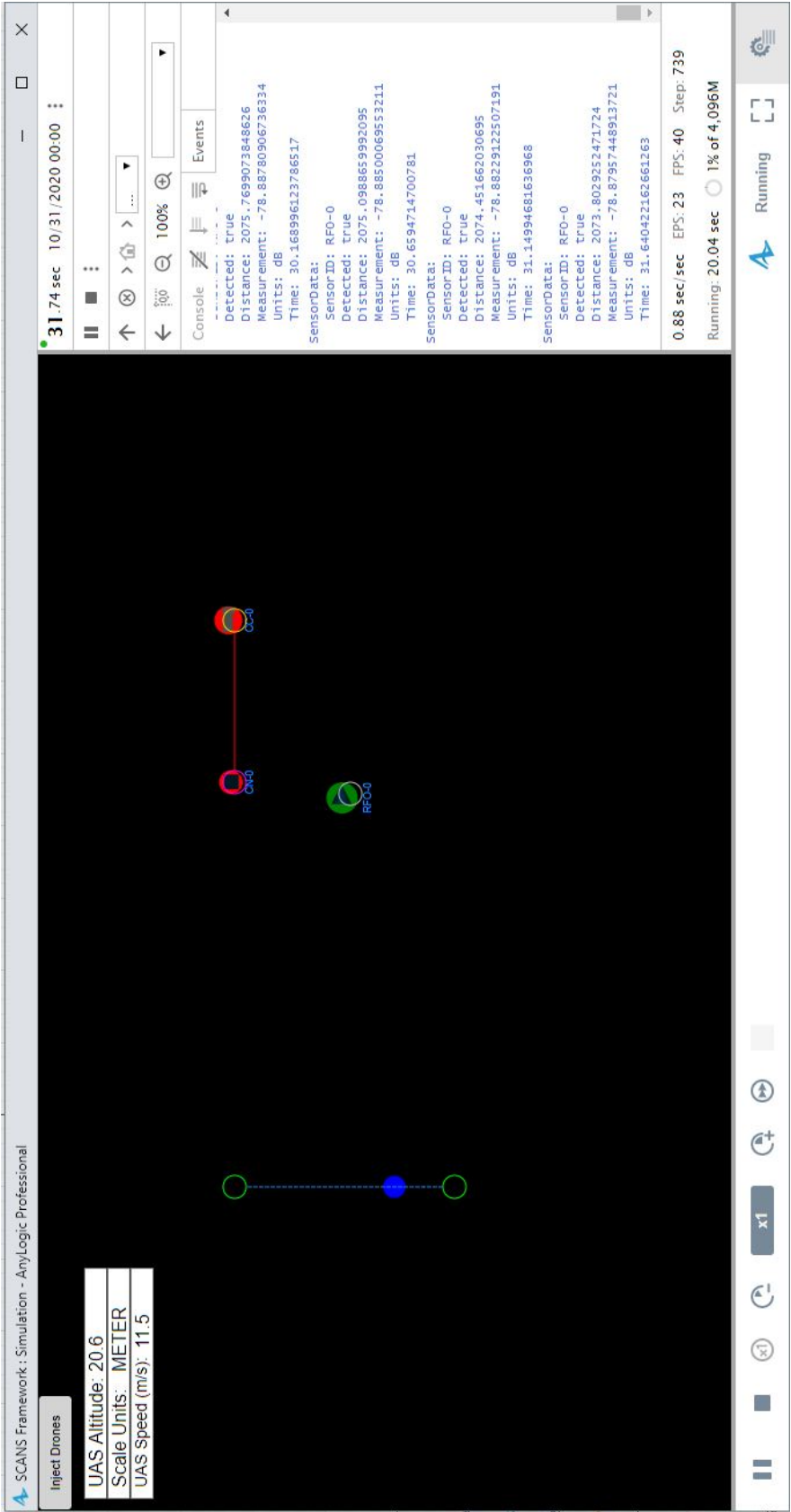


Figure 5.9. *PassiveRF* Agent Configured as DeDrone’s RF-100

CHAPTER 6. SUMMARY

This work has presented and detailed a novel framework for the modeling of CUAS systems. The framework, Simulation of CUAS Networks and Sensors (SCANS), has been designed to use mathematical formulas for determining key sensor performance characteristics, instead of the traditional “maximum sensor range” approach. Along with the formulas, the AEIOU framework was used as a springboard for documenting and identifying various activities, environments, interactions, objects, and users necessary for such a system. This led to an object hierarchy broken into three categories, *UAS*, *sensors*, and *communication devices*.

Under the *sensors* category, *Acoustic*, *Radar*, *Camera*, and *PassiveRF* agent types were developed. Each had common input parameters and individual input parameters, as well as various functions and variables. The *communication devices* category housed the *CommunicationNode* and *CommandAndControl* agents. The *CommunicationNode* acts as the glue between sensors and the *CommandAndControl* agent, passing the custom datatype of *SensorData* in message format between agents. Finally, the *CommandAndControl* agent was tasked with logging any received messages. This restriction of data flow allows the system to respond similarly to a real-world system, whereby information must reach a central processing location before further action can be taken. Chapter 3 detailed each agent’s functions and parameters, while Chapter 4 provided an implementation example of the framework using the AnyLogic simulation software.

Chapter 5 provided tests and results of several published studies and from this, the AnyLogic’s implementation was analyzed for calculation accuracy. In addition to the analysis presented, the SCANS Framework also reported the SPL measured by the *Acoustic* agents, the RSS measured by the *PassiveRF* agents, and the maximum detection range the *Radar* and *Camera* agents were capable of.

From this, it was found that the calculations for the *Radar* agent may have been too simplistic as the maximum range measured in Farlik et al. (2019) was never achieved. A potential change to the Framework would be to report the reflected power signal for the *Radar* agent, and the measured pixels per foot for the *Camera* agent, so as to provide more actionable information. These changes would affect the `calculateMeasure()` function of both agents.

6.1 Future Work

Additional future changes made to the framework should work to address current model limitations. One such limitation is the inability to consider a sensor's vertical FOV in relation to the height of a *UAS* agent. The model currently takes 3D space into account, however the model fails to consider the pitch angle of a sensor when calculating distances. Currently, this results in the model assuming the sensor agent is aimed correctly in the vertical axis orientation.

Another limitation of the framework is the absence of a LiDAR agent. Future work needs to more closely examine the requirements of modeling a LiDAR system and its many complexities. Unfortunately, at this time, not enough information could be gathered to accurately simulate a LiDAR sensor's performance. It is potentially feasible to model the LiDAR system as a flash LiDAR, instead of a traditional sweep LiDAR. This would eliminate needing to model the laser beam performance, however flash LiDAR systems are considered to have their own unique performance equations. This may pose more problems than are solved.

The next major enhancement of the model should be the networking interfaces. Additional routing protocols, queues, and advanced communication protocols would enhance the model's capability to produce realistic results in terms of data transmission. Wired communication capabilities could also be added, either as a separate communication agent, or as a switch within the *CommunicationNode*

agent. Provisions should be made to simulate the performance of less traditional modulation schemes, such as Chirp Spread Spectrum (CSS) in LoRaWAN.

From the results presented in Section 5.2, a more robust examination of the mathematical formulas surrounding the *Radar* agent are needed so as to better understand the discrepancies created by the range calculations. While Equation 2.9 is foundational in understanding the relationship between signal strength and maximum range, perhaps additional information is needed, or an unfound implementation error in the program code is present.

6.2 Closing Thoughts

What is necessary to develop and validate an extensible and general use case CUAS simulation model, with a focus on sensor simulation and a detection-alert network communication system? That was the question this project worked to answer. The SCANS Framework provides a novel way of modeling the CUAS space using a limited number of agent types. With this, each sensor agent is capable of free movement, and creates detection-events based on mathematical formulas, instead of relying upon the manufacture published effective distance rates. Furthermore, given the decision to name functions similarly and the use of helper methods for invocation, functionality can be added or changed with minimal alterations to the rest of the Framework. The functionality and versatility of the framework was then shown by modeling and testing several experiments involving a host of sensor types and UAS.

This work presents a foundation on which to expand the SCANS Framework and its functionality in the future. Continued validation and adaptation of the model should be done by using the tool to first design, and then implement a CUAS system. Comparisons of the results from empirical studies to that of the model should then follow. It is of my opinion that this tool is most useful when used for

planning and preliminary CUAS system design, where estimates can be obtained for real-world performance.

LIST OF REFERENCES

LIST OF REFERENCES

- Ahmad, K. A., Salleh, M. S., Segaran, J. D., & Hashim, F. R. (2018). Impact of foliage on LoRa 433MHz propagation in tropical environment. *AIP Conference Proceedings, 1930*(February), 1–7. doi: 10.1063/1.5022903
- Atherton, K. (2016). *Hobbyist Flies Drone To 11,000 Feet*. Retrieved 2019-03-24, from <https://www.popsci.com/hobbyist-flies-drone-to-11000-feet>
- Augustin, A., Yi, J., Clausen, T., & Townsley, W. M. (2016). A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors, 16*(9). Retrieved from <https://search-proquest-com.ezproxy.lib.purdue.edu/technology1/docview/1882248452/BECAF79264F74955PQ/7?accountid=13360> doi: 10.3390/s16091466
- Broekaert, M., & Budin, J. (2003). Pixel fusion and superresolution for Matis handheld thermal imager. *Infrared Technology and Applications XXIX, 5074*(October 2003), 233. doi: 10.1117/12.497131
- Busset, J., Perrodin, F., Wellig, P., Ott, B., Heutschi, K., Rühl, T., & Nussbaumer, T. (2015, oct). Detection and tracking of drones using advanced acoustic cameras. In E. M. Carapezza, P. G. Datskos, C. Tsamis, L. Laycock, & H. J. White (Eds.), (Vol. 9647, p. 96470F). International Society for Optics and Photonics. Retrieved from <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2194309> doi: 10.1117/12.2194309
- Cabell, R., McSwain, R., & Grosveld, F. (2016). Measured Noise from Small Unmanned Aerial Vehicles. In *New England NoiseCon*. Hampton: NASA/Langley Research Center.
- Cattani, M., Boano, C., & Römer, K. (2017, jun). An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication. *Journal of Sensor and Actuator Networks, 6*(2), 7. Retrieved from <http://www.mdpi.com/2224-2708/6/2/7> doi: 10.3390/jsan6020007
- Cisco. (2009). Cisco Aironet 1130AG Series IEEE 802 . 11A / B / G Access Point. , 1–8.
- Cline, T. L., & Dietz, J. E. (2020). Agent based modeling for low-cost counter UAS protocol in prisons. *International Journal of Aviation, Aeronautics, and Aerospace, 7*(2). doi: 10.15394/IJAAA.2020.1462
- Clover, J. (2019). *AT&T and Sprint Settle Lawsuit Over Misleading '5GE' Label for AT&T's 4G Network*. Retrieved 2019-08-18, from <https://www.macrumors.com/2019/04/22/sprint-att-5ge-lawsuit-settled/>

- Coleman, D., & Westcott, D. (2018). *CWNA Certified Wireless Network Administrator Study Guide* (Fifth ed.; J. Sleeva, Ed.). Indianapolis, Indiana: John Wiley & Sons, Inc.
- Dawaliby, S., Bradai, A., & Pousset, Y. (2017). In depth performance evaluation of LTE-M for M2M communications. *IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 1–8. doi: 10.1109/wimob.2016.7763264
- De Clercq, G. (2018). *Greenpeace crashes Superman-shaped drone into French nuclear plant*. Retrieved 2019-03-17, from <https://www.reuters.com/article/us-france-nuclear-greenpeace/greenpeace-crashes-superman-shaped-drone-into-french-nuclear-plant-idUSKBN1JT1JM>
- Dedrone, Inc. (n.d.). Retrieved 2019-08-09, from <https://www.dedrone.com/>
- DJI. (n.d.). *Phantom 2 - The Spirit Of Flight*. Retrieved 2020-11-08, from <https://www.dji.com/phantom-2>
- Drafts, B. (2001). Acoustic Wave Technology Sensors. *IEEE Transactions on Microwave Theory and Techniques*, 49(4 II), 795–802. doi: 10.1109/22.915466
- Drone Swarm Detection Capabilities*. (2018). Retrieved 2019-03-24, from <http://www.uavexpertnews.com/2018/09/drone-swarm-detection-capabilities/>
- Drozdowicz, J., Wielgo, M., Samczynski, P., Kulpa, K., Krzonkalla, J., Mordzonek, M., ... Jakielaszek, Z. (2016, may). 35 GHz FMCW drone detection system. In *2016 17th International Radar Symposium (IRS)* (pp. 1–4). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7497351/> doi: 10.1109/IRS.2016.7497351
- Drug delivery drone crashes in Mexico*. (2015). Retrieved 2019-03-17, from <https://www.bbc.com/news/technology-30932395>
- Farlik, J., Kratky, M., Casar, J., & Sary, V. (2019). Multispectral detection of commercial unmanned aerial vehicles. *Sensors (Switzerland)*, 19(7). doi: 10.3390/s19071517
- Federal Aviation Administration. (n.d.).
- Federal Aviation Administration. (2018). *Fact Sheet – Small Unmanned Aircraft Regulations (Part 107)*. Retrieved 2019-08-11, from https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=22615
- Federal Aviation Administration. (2019). *Unmanned Aircraft System Detection-Technical Considerations*.
- Federal Emergency Management Agency. (1995). Explosive blast. *Buildings*, 1–14.

- Finn, R. L., & Wright, D. (2012, apr). Unmanned aircraft systems: Surveillance, ethics and privacy in civil applications. *Computer Law & Security Review*, 28(2), 184–194. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0267364912000234> doi: 10.1016/J.CLSR.2012.01.005
- Fungineers. (2016). *How much weight can a drone lift?* YouTube. Retrieved from <https://www.youtube.com/watch?v=UoVrXuM5lXk>
- Gallagher, S. (2013). *German chancellor's drone "attack" shows the threat of weaponized UAVs — Ars Technica*. Retrieved 2019-03-17, from <https://arstechnica.com/information-technology/2013/09/german-chancellors-drone-attack-shows-the-threat-of-weaponized-uavs/>
- Ganti, S. R., & Kim, Y. (2016, jun). Implementation of detection and tracking mechanism for small UAS. In *2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016* (pp. 1254–1260). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7502513/> doi: 10.1109/ICUAS.2016.7502513
- Ghoraishi, M., Takada, J.-i., & Imai, T. (2016). Radio Wave Propagation Through Vegetation. *Intech, i(tourism)*, 13. doi: <http://dx.doi.org/10.5772/57353>
- Hammer, M., Borgmann, B., Hebel, M., & Arens, M. (2019, may). UAV detection, tracking, and classification by sensor fusion of a 360 lidar system and an alignable classification sensor. In M. D. Turner & G. W. Kamerman (Eds.), *Laser Radar Technology and Applications XXIV* (Vol. 11005, p. 10). SPIE. Retrieved from [https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11005/2518427/](https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11005/2518427/UAV-detection-tracking-and-classification-by-sensor-fusion-of-a/) doi: 10.1117/12.2518427
- Haneda, K., Omaki, N., Imai, T., Raschkowski, L., Peter, M., & Roivainen, A. (2016, may). Frequency-Agile Pathloss Models for Urban Street Canyons. *IEEE Transactions on Antennas and Propagation*, 64(5), 1941–1951. Retrieved from <http://ieeexplore.ieee.org/document/7421994/> doi: 10.1109/TAP.2016.2536170
- Hauzenberger, L., & Holmberg Ohlsson, E. (2015). Drone Detection using Audio Analysis. Retrieved from <https://lup.lub.lu.se/student-papers/search/publication/7362609>
- Hennigan, W. (2017). *Islamic State's deadly drone operation is faltering, but U.S. commanders see broader danger ahead*. Retrieved 2019-08-24, from <https://www.latimes.com/world/la-fg-isis-drones-20170928-story.html>
- Holloway, C. L., Koepke, G., Camell, D., Young, W. F., & Remley, K. A. (2014). Propagation measurements before, during, and after the collapse of three large public buildings. *IEEE Antennas and Propagation Magazine*, 56(3), 16–36. doi: 10.1109/MAP.2014.6867680
- How Much Weight Can A Drone Lift?* (n.d.). Retrieved 2019-03-24, from <https://www.uavsystemsinternational.com/how-much-weight-can-a-drone-lift/>

- Jamrogowicz, K. (n.d.). 2 and 5 GHz Real World Propagation. Retrieved from <https://maipn.org/wp-content/uploads/2017/04/propagation.pdf>
- Kim, B. H., Khan, D., Choi, W., & Kim, M. Y. (2019). Real-time counter-UAV system for long distance small drones using double pan-tilt scan laser radar. (May 2019), 8. doi: 10.1117/12.2520110
- Klepal, M., Mathur, R., McGibney, A., & Pesch, D. (n.d.). Influence of people shadowing on optimal deployment of WLAN access points. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004* (Vol. 6, pp. 4516–4520). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/1404934/> doi: 10.1109/VETECF.2004.1404934
- Koettl, C., & Marcolini, B. (2018). *A Closer Look at the Drone Attack on Maduro in Venezuela - The New York Times*. Retrieved 2019-03-18, from <https://www.nytimes.com/2018/08/10/world/americas/venezuela-video-analysis.html>
- Labbe, C., & Rose, M. (2014). *France investigates mystery drone activity over nuclear plants — Reuters*. Retrieved 2019-03-17, from <https://www.reuters.com/article/us-edf-drones/france-investigates-mystery-drone-activity-over-nuclear-plants-idUSKBN0IJ0ZI20141030>
- Laurenzis, M., Rebert, M., Schertzer, S., & Christnacher, F. (2019). Tracking and prediction of small unmanned aerial vehicles' flight behavior and three-dimensional flight path from laser gated viewing images. (May 2019), 9. doi: 10.1117/12.2519273
- Lauridsen, M., Gimenez, L. C., Rodriguez, I., Sorensen, T. B., & Mogensen, P. (2017, mar). From LTE to 5G for Connected Mobility. *IEEE Communications Magazine*, 55(3), 156–162. Retrieved from <http://ieeexplore.ieee.org/document/7876975/> doi: 10.1109/MCOM.2017.1600778CM
- Lauridsen, M., Kovacs, I. Z., Mogensen, P., Sorensen, M., & Holst, S. (2016, sep). Coverage and Capacity Analysis of LTE-M and NB-IoT in a Rural Area. In *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)* (pp. 1–5). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7880946/> doi: 10.1109/VTCFall.2016.7880946
- Lee, D. (2019). *Amazon to deliver by drone 'within months'*. Retrieved 2019-08-22, from <https://www.bbc.com/news/technology-48536319>
- Li, C. J., & Ling, H. (2017). An Investigation on the Radar Signatures of Small Consumer Drones. *IEEE Antennas and Wireless Propagation Letters*, 16, 649–652. doi: 10.1109/LAWP.2016.2594766
- Li, S. (2019). Applying multi agent system to track uav movement. (December).
- Liu, H., Qu, F., Liu, Y., Zhao, W., & Chen, Y. (2018). A drone detection with aircraft classification based on a camera array. *IOP Conference Series: Materials Science and Engineering*, 322(5). doi: 10.1088/1757-899X/322/5/052005

- Liu, H., Wei, Z., Chen, Y., Pan, J., Lin, L., & Ren, Y. (2017, apr). Drone Detection Based on an Audio-Assisted Camera Array. In *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)* (pp. 402–406). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7966780/> doi: 10.1109/BigMM.2017.57
- LoRa Alliance. (2015). *LoRaWAN - What is it? A technical overview of LoRa® and LoRaWAN™* (Tech. Rep.). San Ramon, California, US: LoRa Alliance Technical Marketing Workgroup. Retrieved from https://docs.wixstatic.com/ugd/ecccc1a_acef1a0dbad649bc894a372cf8ff6beb.pdf
- LoRa Alliance. (2017). *LoRaWAN™ 101, A technical introduction*.
- Lu, J. S., Bertoni, H. L., Remley, K. A., Young, W. F., & Ladbury, J. (2014). Site-specific models of the received power for radio communication in urban street canyons. *IEEE Transactions on Antennas and Propagation*, 62(4), 2192–2200. doi: 10.1109/TAP.2013.2297164
- Martin, B., & Hanington, B. (2012). *Universal Methods of Design*. Beverly ,MA: Rockport Publishers.
- Mathur, R., Klepal, M., McGibney, A., & Pesch, D. (n.d.). Influence of people shadowing on bit error rate of IEEE802.11 2.4GHz channel. In *1st International Symposium on Wireless Communication Systems, 2004.* (pp. 448–452). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/1407287/> doi: 10.1109/ISWCS.2004.1407287
- Matolak, D. W., Remley, K. A., Gentile, C., Holloway, C. L., Wu, Q., & Zhang, Q. (2014, oct). Peer-to-Peer Urban Channel Characteristics for Two Public-Safety Frequency Bands. *IEEE Antennas and Propagation Magazine*, 56(5), 101–115. Retrieved from <http://ieeexplore.ieee.org/document/6971921/> doi: 10.1109/MAP.2014.6971921
- Mavic 2 - Specifications, FAQs, Videos, Tutorials, Manuals.* (n.d.). Retrieved 2019-03-24, from <https://www.dji.com/mavic-2/info>
- McManamon, P. F., Banks, P., Beck, J., Fried, D. G., Huntington, A. S., & Watson, E. A. (2017). Comparison of flash lidar detector options. *Optical Engineering*, 56(3), 031223. doi: 10.1117/1.oe.56.3.031223
- McManamon, P. F., & McManamon, P. F. (2019). Introduction to LiDAR. *LiDAR Technologies and Systems*, 1–18. doi: 10.1117/3.2518254.ch1
- Mezei, J., Fiaska, V., & Molnar, A. (2015, nov). Drone sound detection. In *2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 333–338). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7382945/> doi: 10.1109/CINTI.2015.7382945

- Mezei, J., & Molnar, A. (2016, may). Drone sound detection by correlation. In *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)* (pp. 509–518). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7507430/> doi: 10.1109/SACI.2016.7507430
- Michel, A. H. (2018). Counter-Drone Systems. *Counter*(February), 23.
- Moore, B. C. J. (1995). *Hearing* (Second ed.; B. C. J. Moore, Ed.). San Diego, CA: Academic Press. Retrieved from <https://books.google.com/books?id=0ywDx9pxCMYC&pg=PA11#v=onepage&q&f=false>
- Murray, N. (2016). *DJI Phantom 3 and 4 maximum lifting weight*. YouTube. Retrieved from <https://www.youtube.com/watch?v=E6zZW5WhC8s>
- National Oceanic and Atmospheric Administration NOAA Coastal Services Center. (2012). Lidar 101 : An Introduction to Lidar Technology , Data , and Applications. *NOAA Coastal Services Center*(November), 76. doi: 10.5194/isprsarchives-XL-7-W3-1257-2015
- Nave, C. R. (1999). *Sound Intensity*. Retrieved 2020-06-14, from <http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/intens.html>
- Next Generation Mobile Networks Alliance 5G Initiative. (2015). 5G White Paper. *NGMN Alliance*, 124. Retrieved from https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf
- Nguyen, P., Ravindranatha, M., Nguyen, A., Han, R., & Vu, T. (2016). Investigating Cost-effective RF-based Detection of Drones. In *Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use - DroNet '16* (pp. 17–22). New York, New York, USA: ACM Press. Retrieved from <http://dl.acm.org/citation.cfm?doid=2935620.2935632> doi: 10.1145/2935620.2935632
- Park, S., Kim, Y., Lee, K., Smith, A. H., Dietz, J. E., & Matson, E. T. (2020). Accessible real-time surveillance radar system for object detection. *Sensors (Switzerland)*, 20(8), 1–19. doi: 10.3390/s20082215
- Park, S., Kwon, R., Yun, S., Ganser, J., Kim, H., & Anthony, S. (2018). Forestry monitoring system using lora and drone. *ACM International Conference Proceeding Series*. doi: 10.1145/3227609.3227677
- Petajajarvi, J., Mikhaylov, K., Roivainen, A., Hanninen, T., & Pettissalo, M. (2015, dec). On the coverage of LPWANs: range evaluation and channel attenuation model for LoRa technology. In *2015 14th International Conference on ITS Telecommunications (ITST)* (pp. 55–59). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7377400/> doi: 10.1109/ITST.2015.7377400
- Phillip, A. (2014). *Delivery drone carrying marijuana, cellphones and tobacco crashed outside a S.C. prison - The Washington Post*. Retrieved 2019-03-17, from <https://www.washingtonpost.com/news/post-nation/wp/2014/07/>

31/a-delivery-drone-carrying-marijuana-cell-phones-and-tobacco-crashed-outside-of-a-s-c-prison/?noredirect=on&utm_term=.5c21c207d2a5

- Qi, S., Zhang, W., & Xu, G. (2018). Detecting consumer drones from static infrared images by fast-saliency and HOG descriptor. *ACM International Conference Proceeding Series*, 62–66. doi: 10.1145/3290420.3290426
- Ray, B. (2017). *What is LTE-M?* Retrieved 2017-07-26, from <https://www.link-labs.com/blog/what-is-lte-m>
- Recreational Fliers & Modeler Community-Based Organizations*. (2019). Retrieved 2019-03-24, from https://www.faa.gov/uas/recreational_fliers/
- Remley, K. A., Koepke, G., Holloway, C., Camell, D., & Grosvenor, C. (2009, jun). Measurements in harsh RF propagation environments to support performance evaluation of wireless sensor networks. *Sensor Review*, 29(3), 211–222. Retrieved from <https://www.emeraldinsight.com/doi/10.1108/02602280910967620> doi: 10.1108/02602280910967620
- Riegsecker, A. (2018). *Measuring Environmental Effects on LoRa Radios in Cold Weather Using 915 MHz*. Unpublished master's thesis, Purdue University.
- Rocadenbosch, F. (2007). Introduction to LIDAR (laser radar) Remote Sensing Systems. (C), 30.
- Rohde & Schwarz. (n.d.). *LTE-M Testing*. Retrieved 2019-04-26, from https://www.rohde-schwarz.com/us/solutions/test-and-measurement/wireless-communication/iot-m2m/lte-m/lte-m-theme_234034.html
- Sargent, R. G. (2011). Advanced Tutorials: Verification and Validation of Simulation Models. *Proceedings of the 2011 Winter Simulation Conference*, 183–198. Retrieved from <https://www.informs-sim.org/wsc11papers/016.pdf>
- Sathyamoorthy, D. (2015). A Review of Security Threats of Unmanned Aerial Vehicles and Mitigation Steps. *The Journal of Defence and Security*, 6(1), 81–97.
- Schmidt, M., & Shear, M. (2015). *A Drone, Too Small for Radar to Detect, Rattles the White House - The New York Times*. Retrieved 2019-03-17, from <https://www.nytimes.com/2015/01/27/us/white-house-drone.html>
- Semtech SX1276*. (n.d.). Retrieved 2017-07-26, from <http://www.semtech.com/wireless-rf/rf-transceivers/sx1276/>
- Shin, D.-H., Jung, D.-H., Kim, D.-C., Ham, J.-W., & Park, S.-O. (2017, feb). A Distributed FMCW Radar System Based on Fiber-Optic Links for Small Drone Detection. *IEEE Transactions on Instrumentation and Measurement*, 66(2), 340–347. Retrieved from <http://ieeexplore.ieee.org/document/7762208/> doi: 10.1109/TIM.2016.2626038

- Skolnik, M. I. (1980). *Introduction to radar systems* (Vol. 3). McGraw-hill New York.
- Solomitckii, D., Gapeyenko, M., Semkin, V., Andreev, S., & Koucheryavy, Y. (2018, jan). Technologies for Efficient Amateur Drone Detection in 5G Millimeter-Wave Cellular Infrastructure. *IEEE Communications Magazine*, 56(1), 43–50. Retrieved from <https://ieeexplore.ieee.org/document/8255736/> doi: 10.1109/MCOM.2017.1700450
- Torija, A. J., Li, Z., & Self, R. H. (2020). Effects of a hovering unmanned aerial vehicle on urban soundscapes perception. *Transportation Research Part D: Transport and Environment*, 78(0), 1–51. doi: 10.1016/j.trd.2019.11.024
- Ullrich, A., Pfennigbauer, M., & Rieger, P. (2013). How to read your LIDAR spec – a comparison of single-laser-output and multi-laser-output LIDAR instruments. *Riegl*, 1–21. Retrieved from <https://mail.google.com/mail/u/0/%5Cnpapers2://publication/uuid/9F4502B4-07AA-41AF-8893-1D7CDD344950>
- U.S. Department of Homeland Security. (n.d.). *Bomb Threat Stand-Off Chart*. U.S. Department of Homeland Security. doi: 10.1002/oby.20937
- U.S. Department of Homeland Security. (2019). *Counter-Unmanned Aircraft Systems* (Tech. Rep. No. September). New York, New York, USA: National Urban Security Technology Laboratory.
- U.S. National Park Service. (n.d.). *Unmanned Aircraft in the National Parks*. Retrieved 2019-08-14, from <https://www.nps.gov/articles/unmanned-aircraft-in-the-national-parks.htm>
- Vos, G. (n.d.). *The Opportunity for LTE-M / Cat-M1 Cat-M1 is a LPWA Technology*. Sierra Wireless.
- Wagoner, A. R., Schrader, D. K., & Matson, E. T. (2017). Towards a vision-based targeting system for counter unmanned aerial systems (CUAS). *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications, CIVEMSA 2017 - Proceedings*, 237–242. doi: 10.1109/CIVEMSA.2017.7995333
- Watson, B. (2018). *Against the Drones: How to Stop Weaponized Consumer Drones*. Retrieved 2019-03-18, from <https://www.defenseone.com/feature/against-the-drones/>
- White, B. A., Shin, H.-S., & Tsourdos, A. (2011). UAV Obstacle Avoidance using Differential Geometry Concepts. *IFAC Proceedings Volumes*, 44(1), 6325–6330. doi: 10.3182/20110828-6-IT-1002.02344
- Yang, B. (2019). *UAV Detection System with Multiple Acoustic Nodes Using Machine Learning Models*. Master of science, Purdue University. Retrieved from https://hammer.figshare.com/articles/thesis/UAV_DETECTION_SYSTEM_WITH_MULTIPLE_ACOUSTIC_NODES_USING_MACHINE_LEARNING_MODELS/7975991

- Yim, D., Chung, J., Cho, Y., Song, H., Jin, D., Kim, S., ... Riegsecker, A. (2018). An Experimental LoRa Performance Evaluation in Tree Farm. In *IEEE Sensors Applications Symposium*. Seoul, Korea: IEEE.
- Young, W. F., Holloway, C. L., Koepke, G., Camell, D., Becquet, Y., & Remley, K. A. (2010, apr). Radio-Wave Propagation Into Large Building Structures—Part 1: CW Signal Attenuation and Variability. *IEEE Transactions on Antennas and Propagation*, 58(4), 1279–1289. Retrieved from <http://ieeexplore.ieee.org/document/5406066/> doi: 10.1109/TAP.2010.2041142
- Young, W. F., Remley, K. A., Holloway, C. L., Koepke, G., Camell, D., Ladbury, J., & Dunlap, C. (2014). Radiowave propagation in urban environments with application to public-safety communications. *IEEE Antennas and Propagation Magazine*, 56(4), 88–107. doi: 10.1109/MAP.2014.6931660
- Zipf, R. K., Kenneth, P. E., & Cashdollar, L. (2010). Explosions and Refuge Chambers: Effects of blast pressure on structures and the human body. Retrieved from <https://www.cdc.gov/niosh/docket/archive/pdfs/NIOSH-125/125-ExplosionsandRefugeChambers.pdf>

APPENDICES

A. CUSTOM SENSORDATA DATA STRUCTURE

```

1  /**
2  * SensorData
3  */
4  public class SensorData implements Serializable {
5
6      private String ID = "";
7      private boolean detected = false;
8      private boolean inRange = false;
9      private double distance = 0.0;
10     private double measure = 0.0;
11     private String units = "";
12     private double time = 0.0;
13     private boolean message_received = true;
14     private String last_com_node = "";
15
16     public SensorData(String sensorID, boolean isDetected,
17                     boolean withinRange, double distanceFromDrone,
18                     double measurement, String unitOfMeasure,
19                     double modelTime){
20         ID = sensorID; //Originating sensor
21         detected = isDetected;
22         inRange = withinRange;
23         distance = distanceFromDrone;
24         measure = measurement; //sensor data
25         units = unitOfMeasure; //"yard", "meter", "dB", etc.

```

```
26         time = modelTime; //seconds within the model
27         //that this data was collected
28     }
29
30     public String getSensorID(){
31         return ID;
32     }
33
34     public boolean getDetection(){
35         return detected;
36     }
37
38     public boolean getWithinRange() {
39         return inRange;
40     }
41
42     public double getDistance(){
43         return distance;
44     }
45
46     public double getMeasurement(){
47         return measure;
48     }
49
50     public String getUnits(){
51         return units;
52     }
53
54     public double getTime() {
55         return time;
56     }
```

```
57
58     public boolean getReceived() {
59         return message_received;
60     }
61
62     public String getLastComNode() {
63         return last_com_node;
64     }
65
66     public void setSensorID(String sensorID){
67         ID = sensorID;
68     }
69
70     public void setDetection(boolean isDetected){
71         detected = isDetected;
72     }
73
74     public void setWithinRange(boolean withinRange) {
75         inRange = withinRange;
76     }
77
78     public void setDistance(double distanceFromDrone){
79         distance = distanceFromDrone;
80     }
81
82     public void setMeasurement(double measurement){
83         measure = measurement;
84     }
85
86     public void setUnitOfMeasure(String unitOfMeasure){
87         units = unitOfMeasure;
```

```
88     }
89
90     public void setTime(double modelTime) {
91         time = modelTime; //time in seconds
92     }
93
94     public void setMessage_Received(boolean received) {
95         message_received = received;
96     }
97
98     public void setLastComNode(String comNodeID) {
99         last_com_node = comNodeID;
100     }
101
102 }
```

B. DATABASE TABLE STRUCTURE

B.1 Database Table: *model_runs*

The *model_runs* table records the start and end time of a model run as well as the number and type of agents used within the model.

Table B.1.

Database Structure for the model_runs Table

Column Name	Type
start_time	String
end_time	String
acoustic_omni	Integer
acoustic_uni	Integer
radar_omni	Integer
radar_uni	Integer
visual_camera	Integer
infrared_camera	Integer
com_nodes	Integer
command_nodes	Integer
run_name	String

B.2 Database Table: *sensor*

The *sensor* table records all agents, except the *UAS* agent, and the parameters of those agents at the start of the model. The *id* column is the agent's framework-assigned *ID* value. Each input parameter name is recorded to the

param_name column, while the value is recorded to the *param_value* column. The *real_time* column records the real-world time in order to place the parameters with a specific model run.

Table B.2.

Database Structure for the sensor Table

Column Name	Type
id	String
param_name	String
param_value	String
real_time	String

B.3 Database Table: *sensor_data*

The *sensor_data* table records the data generated by the various agents when a detection event is triggered. The *originating_sensor_id* refers to the sensor agent responsible for originally generating the message. A true positive detection for an individual sensor agent is marked by the columns *detected* and *in_range* recording a *true* value. The *distance* column records the distance between the *UAS* agent being detected and the sensor agent performing the detection. Columns *measure* and *sensor_units* are the calculated measures and units of said measures, as defined within a sensor's `calculatedMeasures()` function. The *model_time* and *real_time* record the time of detection in both the model simulation time and real-world time, respectively. Finally, the *message_received* and *last_com_node* column record whether the message was successfully received by the recording *CommandAndControl* agent. If *message_received* is *false*, the message failed to be received by either a *CommunicationNode* or the *CommandAndControl* node, while the *last_com_node* records the agent *ID* of the agent that failed to receive the message.

Table B.3.

Database Structure for the sensor_data Table

Column Name	Type
originating_sensor_id	String
detected	Boolean
in_range	Boolean
distance	Double
measure	Double
sensor_units	String
model_time	String
real_time	String
messsage_received	Boolean
last_com_node	String

C. DATA PROCESSING SCRIPTS

C.1 PowerShell: Sort and Reformat AnyLogic Output

The purpose of this script is to take the direct output from the *sensor_data* table in AnyLogic and reorganize the data for the consumption of the *pColorMesh* graph. As a prerequisite, the *model_time* parameter must be rounded to one tenth of a second. This performs a basic grouping function on the data which would otherwise have only unique model times recorded. The *model_time* column is then sorted and unique values are added to the PowerShell *\$uniqueTimes* variable. Using these times as an index, each unique sensor from the AnyLogic output is added as a column head. For every item (*\$line*) in *\$uniqueTimes*, the script queries for all records from the AnyLogic data. If a record exists for a given sensor and time, and that record's *detected* property and *message_received* property are *true*, the value for the cell becomes "1", otherwise the value for that cell becomes "0". If a sensor did not record a detection at the model time being queried, the cell value is marked with a "0".

The final output of the script is a data table structure with the number of *\$uniqueTimes* rows and the number of *\$uniqueSensors* columns. The value for each data cell is either a "1" or "0". This output is written to a CSV file which is then ingested by the Section C.2 Python script.

```

1  #=====
2  #   DATA REFORMATTER
3  #=====
4
5  $path = $1 #First passed in argument

```

```

6 $outputFile = $2 #Second passed in argument
7 $csv = Import-CSV -path $path
8 $times = @()
9 $output = @()
10
11 #Time comes in as a string, need to turn it into a double
12 foreach($line in $csv){
13     #move all time entries to another array
14     $times += [double]$line.model_time
15 }
16
17 #sort the time array and grab only unique values
18 $uniqueTimes = $times | sort -unique
19
20 #make this script more generalized and grab unique sensors from the data.
21 $uniqueSensors = $csv.SensorID | Sort -Unique
22
23 #For every timestamp
24 foreach ($line in $uniqueTimes){
25
26     #Get every sensor ID and Detected for that timestamp
27     $data = $csv | Where-Object {$_.model_time -eq $line
28         -and $_.messageReceived -ne "FALSE"}
29
30     #Create a blank custom object. Very important line.
31     $record = [pscustomobject]@{}
32
33     #Add the time as a property
34     $record | Add-Member -MemberType NoteProperty
35         -Name time -Value $line
36

```

```

37     #For a given timestamp, work through every record
38     #(should be one per sensor per time)
39     foreach ($sensor in $uniqueSensors){
40         $value = 0
41
42         #If a sensor ID is found, use the data
43         #from the Detected field.
44         if ($data.SensorID -contains $sensor){
45             if ($($data | Where-Object {$_.SensorID
46                 -eq $sensor}).Detected -eq "TRUE"){
47                 $value = 1
48             }
49         } #If the sensor ID is not found,
50         #set the data to False (see line 40)
51
52         #Add the property using the sensorID as the
53         #column header and the Detected as the value
54         $record | Add-Member -MemberType NoteProperty
55             -Name $sensor -Value $value
56     }
57     #Add the psCustomObject to a more general list for output.
58     $output += $record
59 }
60 $output | Export-CSV -path $outputFile
61 #=====
62 #   END: DATA REFORMAT
63 #=====

```

C.2 Python: Generate pColorMesh Graphs

This Python script inputs the data structure generated by the PowerShell script in Section C.1 and outputs a pColorMesh graph.

```

1  #=====
2  #   CREATE PCOLORMESH GRAPH
3  #=====
4  import matplotlib.pyplot as plt
5  import numpy as np
6  import pandas as pd
7
8  def graphDetectionPlot(df, plotTitle, x_label,
9      y_label, transpose = False):
10
11     #fig adjust the entire graph, ax adjust individual graphs
12     fig, ax = plt.subplots()
13     ax.set_title(plotTitle)
14     ax.set_xlabel(x_label)
15     ax.set_ylabel(y_label)
16     if (transpose == True):
17         plt.pcolormesh(df.T)
18     else:
19         plt.pcolormesh(df.iloc[1:,1:])
20     plt.show()
21
22 path = $inputFile #Output from PowerShell data formatter script
23 df = pd.read_csv(path) #read csv into a dataframe
24 df.set_index("time", inplace=True)
25
26 graphDetectionPlot(df, "Experiment_1:_Trial_1",

```

```

27         "Detection_Records", "Sensor", True)
28 #=====
29 #   END: CREATE PCOLORMESH GRAPH
30 #=====

```

C.3 PowerShell: Quick Sort by SensorID for AnyLogic Output

This PowerShell script sorts the *sensor_data* table output from AnyLogic by the *originating_sensor_id* column which was renamed to *SensorID* here. This enables easier data processing within Excel, or another program, for determining the number of true positives, false negatives, averages, and so on.

```

1 #=====
2 #   DATA SORTER
3 #=====
4 $path = $1 #First passed in argument
5 $outputPath = $2 #Second passed in argument
6 $csv = Import-CSV -path $path
7 $output = @()
8
9 #make this script more generalized and
10 #grab unique sensors from the data.
11 $uniqueSensors = $csv.SensorID | Sort -Unique
12
13 #For every timestamp
14 foreach ($sensor in $uniqueSensors){
15     if ($sensor -eq "" -or $sensor -eq $null){
16         continue
17     }

```

```
18         $data = $csv | Where-Object {$_.SensorID -eq $sensor}
19
20         #Get every sensor ID and Add the psCustomObject
21         #to a more general list for output.
22         $output += $data
23     }
24     $output | Export-CSV -path $outputFile$
25     #=====
26     #   END: DATA SORTER
27     #=====
```

VITA

VITA

Austin J. Riegsecker

Education

- Ph.D., Technology
Purdue University, West Lafayette, IN, USA
December 2020
- M.S., Computer and Information Technology
Purdue University, West Lafayette, IN, USA
May 2018
- B.S., Computer and Information Technology
Purdue University, West Lafayette, IN, USA
December 2016

Work Experience

- Virtualization Solutions Architect
General Dynamics Information Technology, Odon, IN, USA
October 2020 - Present
- Senior Software Developer
General Dynamics Information Technology, Odon, IN, USA
April 2020 - September 2020
- Wireless Network Consultant
West Lafayette, IN, USA
May 2019 - August 2019

- International Cybersecurity Boot Camp Instructor
Purdue University, West Lafayette, IN, USA
May 2019 - August 2019
- Head Teaching Assistant
Purdue University, West Lafayette, IN, USA
January 2019 - May 2019
- GenCyber Course Instructor
Purdue University, West Lafayette, IN, USA
May 2018 - August 2018
- Teaching and Research Assistant
Purdue University, West Lafayette, IN, USA
January 2017 - December 2020
- Network Administrator
Korean Software Square, West Lafayette, IN, USA
January 2017 - December 2020
- Software Developer Intern
Crowe Horwath - Applied Technology Department, Indianapolis, IN, USA
May 2016 - August 2016
- Software Developer Intern
CEC Controls, Warren, MI, USA
May 2013 - May 2016

Publications

- Cline, T. L., Ho, K. E., Hood, C., Riegsecker, A., & Dietz, J. E. (2020). A case for standardized communications for point of distribution operations. *Journal of Emergency Management*, 18(6). <https://doi.org/10.5055/jem.2020.0490>
- Gilbert, A. K., Riegsecker, A., & Dietz, J. E. (2018). Assessing Security Perimeters for Large Events using Agent-based modeling. In *International Association of Journals and Conferences*. Orlando, FL, USA: International.

- Riegsecker, A. (2018). *Measuring Environmental Effects on LoRa Radios in Cold Weather Using 915 MHz*. Purdue University.
- Yim, D., Chung, J., Cho, Y., Song, H., Jin, D., Kim, S., ... Riegsecker, A. (2018). An Experimental LoRa Performance Evaluation in Tree Farm. In *IEEE Sensors Applications Symposium* (p. 6). Seoul, Korea: IEEE.

Service

- IEEE IRC Program Committee Member (2020)
- IEEE IRC Peer Reviewer (2019 & 2020)
- ETRI Journal Peer Reviewer (2018)
- ICPADS Peer Reviewer (2017)
- Korean Software Square Research Team Advisor (2017-2019)