



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**ADVANCING COTS UAV CAPABILITY
TO PROVIDE VISION-BASED SA/ISR DATA**

by

Wei Shun Teo

September 2018

Thesis Advisor:
Second Reader:

Oleg A. Yakimenko
Fotis A. Papoulas

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 2018	3. REPORT TYPE AND DATES COVERED Master's thesis		
4. TITLE AND SUBTITLE ADVANCING COTS UAV CAPABILITY TO PROVIDE VISION-BASED SA/ISR DATA			5. FUNDING NUMBERS	
6. AUTHOR(S) Wei Shun Teo				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Unmanned systems are gaining popularity in many modern-day applications, and their growth potential in unmanned technologies is infinite. These systems have created research and development opportunities for enabling autonomous behavior to reduce human workload and involvement in tedious operations. This thesis assesses autonomy-enabling technologies for conducting search-and-rescue (SA) operations and intelligence, surveillance, and reconnaissance (ISR) missions using a small unmanned system (sUAS). These technologies include an electro-optical sensor, onboard processor, and computer-vision (CV) algorithms. In a previous master's thesis by Wee Kiong Ang, a commercial off-the-shelf (COTS) quadcopter sUAS was integrated with a suite of hardware and multiple-moving-target-detection software. Building upon that work, this thesis aims to advance the system's capabilities by exploring the applicability of the aforementioned three technologies on an sUAS. Using the systems engineering approach, the baseline system deficiencies are identified first. Next, a technology enabler review is conducted to explore the relevant COTS products and paradigms. Then, through the implementation of a set of changes, the baseline system architecture is reassessed and consequently redesigned, followed by an assessment of state-of-the-art CV algorithms. After being tested in a field experiment based on SA/ISR-type mission scenarios, the developed prototype was found to be successful.				
14. SUBJECT TERMS unmanned aerial system, UAV, computer vision, object detection, optical-flow, payload sensor, manifold, GPU			15. NUMBER OF PAGES 161	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**ADVANCING COTS UAV CAPABILITY TO PROVIDE VISION-BASED SA/ISR
DATA**

Wei Shun Teo
Civilian, DSO National Laboratories, Singapore
BEng, Nanyang Technological University, 2009
MS, National University of Singapore, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2018**

Approved by: Oleg A. Yakimenko
Advisor

Fotis A. Papoulas
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned systems are gaining popularity in many modern-day applications, and their growth potential in unmanned technologies is infinite. These systems have created research and development opportunities for enabling autonomous behavior to reduce human workload and involvement in tedious operations. This thesis assesses autonomy-enabling technologies for conducting search-and-rescue (SA) operations and intelligence, surveillance, and reconnaissance (ISR) missions using a small unmanned system (sUAS). These technologies include an electro-optical sensor, onboard processor, and computer-vision (CV) algorithms. In a previous master's thesis by Wee Kiong Ang, a commercial off-the-shelf (COTS) quadcopter sUAS was integrated with a suite of hardware and multiple-moving-target-detection software. Building upon that work, this thesis aims to advance the system's capabilities by exploring the applicability of the aforementioned three technologies on an sUAS. Using the systems engineering approach, the baseline system deficiencies are identified first. Next, a technology enabler review is conducted to explore the relevant COTS products and paradigms. Then, through the implementation of a set of changes, the baseline system architecture is reassessed and consequently redesigned, followed by an assessment of state-of-the-art CV algorithms. After being tested in a field experiment based on SA/ISR-type mission scenarios, the developed prototype was found to be successful.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	AUTONOMY ENABLERS.....	1
B.	PROBLEM FORMULATION AND RESEARCH QUESTIONS	3
C.	THESIS CHAPTER OUTLINE	4
II.	REVIEW OF BASELINE SYSTEM.....	5
A.	SYSTEM ARCHITECTURE	5
B.	STANDARD M100 HARDWARE CONFIGURATION	7
C.	BASELINE SYSTEM HARDWARE ADDITIONS	11
D.	SOFTWARE CONFIGURATION OF BASELINE PLATFORM	14
E.	THREAT DETECTION ALGORITHM.....	16
F.	HIGHLIGHTS OF THE BASELINE SYSTEM FLIGHT TESTS	17
G.	IDENTIFIED PROBLEMS OF THE BASELINE PLATFORM	20
1.	Sensitivity of the Classifier Parameters	20
2.	Overlap in the Bounding Box	21
3.	Rolling Shutter Effects	22
III.	TECHNOLOGY ENABLERS AND AREAS FOR BASELINE SYSTEM IMPROVEMENTS.....	23
A.	PROPOSED CHANGES	23
B.	EO SENSOR TECHNOLOGIES	25
1.	Imaging Considerations	25
2.	Sensor Technologies.....	26
3.	Visual Considerations.....	29
C.	COMPUTING HARDWARE TECHNOLOGIES	34
1.	Hardware Technologies.....	34
2.	Performance-Power Considerations	38
D.	ADVANCEMENT IN CV ALGORITHMS	40
1.	Convolutional Neural Networks	42
2.	Object Detection Algorithm	47
3.	Performance	53
IV.	ADVANCED SYSTEM DESIGN.....	55
A.	SYSTEM ARCHITECTURE	55
B.	HARDWARE CONFIGURATION	56
C.	SOFTWARE CONFIGURATION.....	59

1.	Software Integration	61
2.	User Applications	67
3.	State-of-the-Art Deep Learning–Based CV Algorithm	70
V.	T&E OF THE ADVANCED SYSTEM	77
A.	EXPERIMENT SETUP.....	77
B.	FIELD TRIALS RESULTS	80
1.	3D Mapping	80
2.	2D Aerial Imagery.....	84
3.	Simulation of a Search Mission	88
4.	Simulation of a Surveillance Mission	93
VI.	CONCLUSIONS AND RECOMMENDATIONS.....	97
A.	SUMMARY	97
B.	RECOMMENDATIONS FOR FUTURE WORK.....	98
	APPENDIX A. DJI MANIFOLD SETUP	101
A.	CODE FOR CUDA INSTALLATION	101
B.	CODE FOR OPENCV INSTALLATION	101
C.	CODE FOR ROS INDIGO INSTALLATION.....	102
	APPENDIX B. DJI ONBOARD SDK	105
A.	INSTALLING THE SDK CORE	105
B.	INSTALLING THE SDK ROS NODES.....	105
C.	CONFIGURATION OF THE DJI SDK.....	106
D.	EXECUTION OF THE SDK PROGRAM.....	106
	APPENDIX C. DJI SDK ROS WRAPPER.....	109
A.	OVERVIEW	109
B.	SUBSCRIBED TOPICS	109
1.	Flight Control Topics.....	109
2.	Gimbal Control	110
C.	PUBLISHED TOPICS.....	110
D.	SERVICES.....	112
E.	PARAMETERS.....	117
F.	DETAILS ON FLIGHT CONTROL SETPOINT	117
	APPENDIX D. DJI MANIFOLD CAMERA	119
A.	BUILDING THE DJI_CAM_TRANSPORT	119
B.	RUNNING THE DJI_CAM_TRANSPORT	119

C.	CODES FOR DJI_CAM_TRANSPORT.....	120
APPENDIX E. YOLO		125
A.	INSTALLING THE BASE SYSTEM	125
B.	COMPILING WITH CUDA AND OPENCV	125
C.	SETUP OF PRE-TRAINED MODEL INTO ALGORITHM	126
LIST OF REFERENCES		129
INITIAL DISTRIBUTION LIST		135

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Baseline M100-Based Platform. Adapted from Ang (2017).	5
Figure 2.	System View of the Baseline Platform. Source: Ang (2017).	6
Figure 3.	Matrice 100 Base Unit. Adapted from DJI (2017).	7
Figure 4.	Overview of the Added Payload Electronics. Adapted from Ang (2017).	11
Figure 5.	Software Architecture of Baseline System	14
Figure 6.	Overview of Purdue/NPS CV Algorithm. Source: Ang (2017).	16
Figure 7.	Varying Classifier Parameters in the Field Test Environment. Source: Ang (2017).	17
Figure 8.	Varying Classifier Parameters in the Urban Test Environment. Source: Ang (2017).	18
Figure 9.	Post-flight Experimental Testing with Tuned Classifier Parameters. Source: Ang (2017).	19
Figure 10.	Unwanted Noise from the Oversensitive Classifier Parameter. Source: Ang (2017).	20
Figure 11.	Extended Bounding Box over Detected Objects. Source: Ang (2017).	21
Figure 12.	Example of Image Quality under Various Conditions. Adapted from Ang (2017).	22
Figure 13.	Examples of Rolling Shutter Effects. Adapted from Jonen (2007) (left); Adler (2016) (right).	24
Figure 14.	Sensor Architecture of CCD Sensor (left) and CMOS Sensor (right). Adapted from Qimaging (2014).	26
Figure 15.	Illustration of Exposure with Global Shutter (left) and Rolling Shutter (right). Source: Lappenküper (2018).	27
Figure 16.	Image Distortion for Global Shutter (top) versus Rolling Shutter (bottom). Source: Lappenküper (2018).	28
Figure 17.	Relationship of Parameters in GSD. Adapted from PIX4D (2017).	29

Figure 18.	Snapshot of the Ground Distance from UAV (50 m Altitude)	31
Figure 19.	Representation of Pixels Count for Human and Small UAV Detection	32
Figure 20.	Block Diagram for Odroid-XU4. Source: Hardkernel (2014).	35
Figure 21.	Block Diagram of DJI Manifold. Adapted from Berkeley Design Technology Inc (2014).	36
Figure 22.	Diagram Illustrating the Package-on-Package Setup of the Memory and Processor. Source: Schiesser (2012).	37
Figure 23.	Plot of CPU Performance Score. Adapted from Triggs (2014).	39
Figure 24.	Plot of GPU Performance Score. Adapted from Triggs (2014).	40
Figure 25.	Comparison of Image Classification, Object Detection, and Instance Segmentation. Source: Ouaknine (2018).	41
Figure 26.	Illustration of the Regular Neural Network versus Convolution Neural Network. Adapted from Li (2017).	43
Figure 27.	Pipeline of a Convolutional Neural Network. Source: Geitgey (2016).	43
Figure 28.	Convolution Process. Source: Dertat (2017).	44
Figure 29.	Visual Representation of Convolution Process Layer. Source: Fergus (2015).	45
Figure 30.	Pooling Process. Adapted from Dertat (2017).	46
Figure 31.	Example of CNN Architecture. Source: Li et al. (2017).	46
Figure 32.	Representation of CNN Architecture. Source: Gandhi (2017).	47
Figure 33.	Visualization of the Selective Search Method. Source: Uijlings et al. (2013).	48
Figure 34.	Architecture of R-CNN. Source: Girshick et al. (2014).	49
Figure 35.	Architecture of Fast R-CNN. Source: Girshick (2015).	50
Figure 36.	Architecture of Faster R-CNN. Source: Xu (2017).	51
Figure 37.	Architecture of YOLO Algorithm. Source: Redmon et al. (2016).	52

Figure 38.	Overview of Advanced System	55
Figure 39.	Interconnection for Flight Controller, Zenmuse X3 Gimbal Mount, and DJI Manifold	56
Figure 40.	Schematic Overview of DJI Manifold Interface (top) and M100 Reserved Ports (bottom). Adapted from DJI (2017).....	59
Figure 41.	Software Architecture for Refined DJI Matrice 100.....	60
Figure 42.	Hierarchy of DJI SDK Application. Source: DJI (2017).....	63
Figure 43.	Successful Execution of the DJI SDK ROS Server	64
Figure 44.	DJI M100 Simulator Environment.....	66
Figure 45.	Example of Flight Control Coding. Source: ROS WIKI (2017).	67
Figure 46.	Successful Execution of ROS Server (left) and Publishing of ROS Image Node (right).....	68
Figure 47.	Visualizer of ROS Image Node (top) and Video Transmission Display on DJI Go Application (bottom).....	70
Figure 48.	Decomposition of the YOLO (CV Algorithm) Command	71
Figure 49.	Successful Execution of the YOLO Algorithm	72
Figure 50.	Predictions for Objects Detection on Sample Image	73
Figure 51.	Image Data Set for the Quadcopter UAV	74
Figure 52.	Data Annotation of Quadcopter in a Single Image	74
Figure 53.	Fort Ord’s Impossible City. Source: Google Earth (top).....	79
Figure 54.	Illustration of Area Mapping in Pix4DCapture	81
Figure 55.	3D Point Cloud Model of Impossible City	82
Figure 56.	3D Mesh Model of Impossible City.....	83
Figure 57.	2D Aerial Imagery of Impossible City (in X-Orientation)	84
Figure 58.	2D Aerial Imagery of Impossible City (in Y-Orientation)	85
Figure 59.	Targets Detection on the 2D Aerial Imagery Map.....	86

Figure 60.	Target Detection on the 2D Aerial Imagery for X- and Y-Axis Orientation	87
Figure 61.	Representation of Probability Road Map for Impossible City.....	88
Figure 62.	Screen Capture of the Target Search Mission Waypoints	89
Figure 63.	Viewing Scene of the UAV during Target Search.....	91
Figure 64.	Snapshot of the Target Detection in YOLO Algorithm.....	92
Figure 65.	Representation of Video Resolution Based on Generic Pictorial. Source: DJI (2017).....	93
Figure 66.	Plot for Ground Area versus Flight Altitude.....	94
Figure 67.	Snapshot of Impossible City at Different Flight Altitudes	95
Figure 68.	Successful Execution of the DJI SDK Demo Client.....	107
Figure 69.	Output Data from the Algorithm.....	126
Figure 70.	Prediction Data of the Sample Image. Source: Redmon et al. (2016).	127

LIST OF TABLES

Table 1.	Key Specifications of the Matrice 100 UAV. Source: DJI (2017).	9
Table 2.	Key Specifications of DJI Zenmuse X3 Gimbal Camera. Adapted from DJI (2017).	10
Table 3.	Key Specifications of the Odroid Processor. Adapted from Hardkernel (2014).	12
Table 4.	Key Specifications of Logitech C920 Webcam. Adapted from Logitech (2016).	13
Table 5.	Potential Object Characteristics that Form Images. Source: Harney (2013).	25
Table 6.	Summary of Payload Hardware. Adapted from DJI (2014); HardKernel (2017).	39
Table 7.	Performance of Algorithm Model. Adapted from Ouaknine (2017).	53
Table 8.	Summary of DJI System Software.....	62
Table 9.	Commands for Running the DJI SDK Server.....	63
Table 10.	Commands for Running the DJI Demo Client.....	65
Table 11.	Commands for Running the Zenmuse X3 Software Application	68
Table 12.	Command for Running the ROS Image Visualizer	69
Table 13.	Command for Running the YOLO Algorithm.....	72

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AI	artificial intelligence
ARM	Advanced RISC Machine
BEC	Battery Eliminator Circuit
CCD	Charge Coupled Device
CCW	Counter Clockwise
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
ConvNet	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
CW	clockwise
DC	Direct Current
DJI	Da Jiang Innovations
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
eMMC	Embedded Multi-Media Controller
EO	Electro-Optical
ESC	Electronic Speed Control
GCS	Ground Control Station
GB	Gigabytes
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSD	Ground Sample Distance
HD	High Definition
INS	Inertial Navigation Systems
IoT	Internet of Things
IP	Internet Protocol
ISR	Intelligence, Surveillance, and Reconnaissance
LiDAR	Light Detection and Ranging
LPDDR3	Low Power Double Data Rate version 3

M100	Matrice 100
NPS	Naval Postgraduate School
OV	Operational View
POP	Package-on-Package
RAM	Random Access Memory
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RoI	Region of Interests
ROM	Read-Only Memory
ROS	Robot Operating System
RPN	Region Proposal Network
RTSP	Real-Time Streaming Protocol
SAR	Search and Rescue
SDK	Software Developmental Kit
SLAM	Simultaneous Localization and Mapping
SoC	System-on-Chip
SV	System View
T&E	Test and Evaluation
UART	Universal Asynchronous Receiver Transmitter
UAV	unmanned aerial vehicle
UGV	unmanned ground vehicle
USB	Universal Serial Bus
vSMP	Variable Symmetric Multi-Processing
YOLO	You Only Look Once

EXECUTIVE SUMMARY

Unmanned systems are gaining popularity for various commercial and military applications. The hype about civilian unmanned aerial vehicle (UAV) applications such as photography, agricultural monitoring, and inspection is driving sales demand and mass production volume, which leads to lower unit costs and greater potential market growth. On the other hand, production volume for military-specific UAVs is much less robust, which results in higher ownership costs for defense spending. According to the *Unmanned Aircraft Systems Roadmap 2005–2030*, the U.S. Department of Defense is exploring the concept of using commercial off-the-shelf (COTS) UAVs as “consumable logistic” and allowing the commercial market to drive the research and development (R&D). In addition, the COTS system will provide a 50-percent solution tomorrow to meet the military’s immediate needs, as compared to waiting for a full-scale development to achieve a 70–80-percent solution in three years (Office of the Secretary of Defense, 2005). As these current COTS UAVs have limited performance to support the military mission, any R&D will need to include autonomous behavior using vision as a means to advance the capabilities of COTS UAVs. The key elements required for vision are: a) a physical sensor, b) the computer-vision (CV) algorithm, and c) the computing platform on which the algorithms run.

This thesis elaborates on the past Naval Postgraduate School (NPS) master’s thesis titled “Assessment of an Onboard EO [Electro-Optical] Sensor to Enable Detect-and-Sense Capability for UAVs Operating in a Cluttered Environment” by Wee Kiong Ang (2017). Through the utilization of the baseline system developed in his thesis work, this thesis research focuses on problem rectification through system improvements to advance the capabilities of a COTS UAV system. To address the problems, this thesis aims to fulfill the following research objectives:

- Evaluate and determine the limitation of existing EO sensors on the baseline system to support the CV algorithm. Consider whether the setup is sufficient to meet the requirements for continuous detection and tracking.

- Perform an assessment of existing CV algorithms and check whether they are adequate to support the detection and tracking requirements. Determine the recommended improvements or approaches can be made to the algorithms.
- Evaluate the capabilities of the UAVs in supporting or fulfilling a simulated operational scenario.

In the initial assessment of the physical sensor, the rolling-shutter based EO sensor (Logitech C920) produced images with object distortion when the object or the capturing platform was moving. Even though the sensor has a high-resolution detector, the image did not appear crisp and clear for image processing usage. The thesis studied the effects of rolling shutter and global shutter, and established that the object distortion in images was attributed to the mode of capturing the image (i.e., how shuttering controlled image capture). Since the rolling shutter EO will capture its imagery by row-by-row exposure, the distorted image will occur for objects that were moving in the capture process. Hence, the result shows that the new EO (Zemuse X3) with a global shutter is a better alternative for capturing moving objects, as well as for system autonomy development.

For the CV algorithm, the existing algorithm (Purdue and NPS algorithm) had multiple occurrences of false positive target detection in the field experiment. Based on the thesis research, it was discovered that the false positive results were due to how detection has been defined. Since the Purdue/NPS algorithm was designed for moving target detection, the capturing sensor is assumed to be stationary. As such, the slight movement of the UAV will result in angular difference of the object in a given scene, and the algorithm will identify it as a target. Hence, this thesis studied and implemented a state-of-the-art object detection based algorithm for the system. Unlike the typical object detection algorithm, the new CV algorithm (You Only Look Once, or YOLO) is less straining to the computational requirements. The YOLO represents a feasible solution for providing real-time object detection on embedded processors.

As for the computing platform, the change in EO sensor and CV algorithm makes the Odroid-XU4 obsolete. In order to keep pace with the relevant technology, the DJI

Manifold system was selected as the computing platform to support the algorithm. The DJI Manifold is based on an Nvidia CPU+GPU framework that accelerates the computation-intensive convolution network layer in the YOLO algorithm. Based on the testing, the Nvidia devices were able to process a sample image (with three objects—a horse, a dog, and a person) with multiple convolution layers and high confidence in approximately 0.3 seconds.

Finally, a field test was conducted to demonstrate the capabilities of the system to perform a search and rescue (SAR) mission. In the SAR, the mission is broken down into three phases (mapping, search, and surveillance). The results show that the COTS UAV was able to perform the operations as required by the mission. In the target search phase, the EO's imagery data was processed by the YOLO algorithm, and it was able to work well to detect the objects (a person and a car) for the four defined scenarios. Hence, this thesis demonstrated advancements in COTS UAV capability through the use of an EO sensor and CV algorithm to fulfill mission needs. With the improvements pursued in this thesis, the M100 platform (COTS UAV) can be quickly turned around for military missions, which is likely a key factor in modern warfare.

References

- Ang, Wee Kiong. 2017. "Assessment of an Onboard EO Sensor to Enable Detect and Sense Capability for UAVs Operating in a Cluttered Environment." Master's thesis, Naval Postgraduate School. <https://calhoun.nps.edu/handle/10945/56165>.
- Office of the Secretary of Defense. 2005. *Unmanned Aircraft Systems Roadmap, 2005 – 2030*. Washington, DC: U.S. Department of Defense. <http://www.dtic.mil/dtic/tr/fulltext/u2/a445081.pdf>.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The journey of this thesis has been a tiring, yet enriching adventure. Although this thesis is an individual work, the thesis would not have been possible without the special people who supported, challenged, and believed in me along the way.

I am extremely fortunate to have Professor Oleg Yakimenko as my thesis advisor. His depth of knowledge in unmanned systems has provided me with many valuable insights, guidance, and support to overcome the obstacles in the thesis work. I am extremely grateful to him for his trust and giving thoughtful feedback to help me move forward.

I would like to express my gratitude to Professor Fotis Papoulias for providing valuable feedback on this thesis despite his busy schedule. His feedback has helped to improve the content of this thesis.

I would like to thank my NPS SE technical writing senior lecturer Barbara Berlitz, NPS Graduate Writing Coach Cheryldee Huddleston, Thesis Processing Office instructor Aileen Houston, and editor Meg Beresik for their coaching and editing of this thesis. Through their assistance to improve the flow and structure of the writing, the thesis has become more professionally written.

Reflecting on the decision to pursue my master's degree with NPS, it would not have been possible without the support of DSO National Laboratories, Singapore. I am truly grateful to my mentors who recommended me for the scholarship to sponsor my studies, giving me the opportunity to embark on a unique learning experience at NPS. I would also like to thank my colleagues for following up with my work.

Finally, I would like to express my gratitude to my family for their encouragement and support, which helped me in completion of this paper. Special gratitude to my supportive wife, Valentina, and son, Dylan, for enduring the extended period away from Singapore for the master's degree studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

With the advancement in technology, the use of unmanned systems such as the unmanned aerial vehicle (UAV) and unmanned ground vehicle (UGV) has gained popularity in various military and civilian applications. Such applications include intelligence, surveillance, and reconnaissance (ISR) operations, strike missions, search and rescue operations, agricultural activities, and hobbyists' usage. According to the U.S. Department of Defense (DoD) *Unmanned Systems Integrated Roadmap FY2013–2038* publication,

The prevalence and uses of unmanned systems continue to grow at a dramatic pace. The past decade of conflict has seen the greatest increase in unmanned aircraft systems, primarily performing ISR missions. Use of unmanned systems in the other domains is growing as well. The growth of unmanned systems use is expected to continue across most domains. Unmanned systems have proven they can enhance situational awareness, reduce human workload, improve mission performance, and minimize overall risk to both civilian and military personnel, and all at a reduce cost. (2013, 20)

Hence, the growing demand for unmanned system applications creates opportunities in research and development for autonomous behavior, particularly autonomous behavior involving the use of computer vision. The key elements required for vision are: a) a physical sensor, b) computer vision (CV) algorithms, and c) the computing platform on which the algorithms are run.

A. AUTONOMY ENABLERS

In the most deployed unmanned systems, the first element, the physical sensor, is usually the electro-optical (EO) sensor. The EO sensor provides situational awareness to the system, and its imagery data contributes to fulfilling the autonomous system requirements. These imagery data can complement the remote sensing operation by providing eyes on target. In addition, algorithm can be designed to allow operations such as target identification, threat assessment, and fire support for tactical weaponry for numerous military operations (Wilson 2016). Nonetheless, EO sensors form only the hardware behind the technology enablers to autonomous behavior in UAV. The software

algorithm shaping the CV techniques has been researched and developed to work with the EO sensors to increase the autonomy of the UAV.

One typical application of unmanned system autonomy is the search and rescue (SAR) mission, which can be a tedious and tiresome task for operators as it requires flight precision and long operation times to screen through the camera coverage of a given area (e.g., a disaster zone). In the event of such emergency situations, the operator needs to locate the potential survivors who might require medical attention. Thus, such a mission is of utmost importance and can be time sensitive. Therefore, unmanned systems can operate autonomously and execute the mission from take-off to landing, and designated to collect imagery footage of every square meter of an area of interest.

The second key element, the CV algorithm, plays a key role in providing perspective of the surroundings environment. The algorithm provides interpretation of the data and enables a response to the situation. Visual Simultaneous Location and Mapping (SLAM) is an example of a potential CV algorithm to yield reasonably accurate local navigation results based on image sequences from an EO sensor. Significantly, the EO sensor and CV algorithm support the navigation element as local referencing in the autonomous system behavior. Together with the data for global referencing (Global Positioning System [GPS] and Inertial Navigation System [INS]), the complete navigation solutions are supplied to the mission computers. The flight dynamics (guidance and control) are constantly updated with the navigation data and the results from the CV algorithm. The system benefits from these inputs to plan the best route for flight advancement, while avoiding the potential threats and the predefined boundaries (Office of the Secretary of Defense 2013).

While most CV algorithms that are developed using desktop computers execute well beyond the requirements, the limited computational capacity of commercial-of-the-shelf (COTS) UAVs precludes the use of modern CV algorithms on them. Hence, the third element, the computing platform, must consider the resources requirements of the CV algorithm. With the improvement in embedded processors through the Internet-of-Things (IoT), system developers must be ready to leverage these highly efficient embedded devices for military applications. The IoT processor is expected to lead to changes in future

military unmanned system design. Furthermore, the IoT-enabled system enables a highly connected operating environment (i.e., swarm UAV operation) and increases the potential for situational awareness via a networked entity (Tortonesi et al. 2017).

B. PROBLEM FORMULATION AND RESEARCH QUESTIONS

This thesis elaborates on a previous Naval Postgraduate School (NPS) master's thesis titled "Assessment of an Onboard EO Sensor to Enable Detect-and-Sense Capability for UAVs Operating in a Cluttered Environment" by Wee Kiong Ang (2017), of the Singapore Army. In his thesis research, he integrated and demonstrated the use of EO sensors to achieve a detect and track capability for multi-UAV operations. Ang discovered, however, that the existing EO sensor (a Logitech webcam) had a rolling shutter mechanism, which degrades the full performance of the CV algorithm. Furthermore, the embedded onboard processor (Odroid) lacked computing power for computationally-intensive algorithm implementation. In addition, the hardware setup was unable to support data sharing and interfacing between the payload and flight controller.

Hence, the present research focuses on the development, integration, and evaluation of the required algorithm as well as the hardware implementation to achieve the desired capability for a COTS UAV to perform onboard decision making upon continuous detection and tracking of a target. To address the aforementioned problems, this thesis aims to fulfill the following research objectives:

- Evaluate and determine the limitation of existing EO sensors on the baseline system to support the CV algorithm. Consider whether the setup is sufficient to meet the requirements for continuous detection and tracking.
- Perform an assessment of existing CV algorithms, and check whether they are adequate to support the detection and tracking requirements. Determine the recommended improvements or approaches that can be made to the algorithms.
- Evaluate the capabilities of the UAVs in supporting or fulfilling a simulated operational scenario.

The thesis study adopts both the qualitative and the quantitative approaches to look into the key performance differences between the rolling shutter and global shutter imaging sensors in providing real-time video stream to the CV algorithm application. It is essential to look at the embedded computing hardware to process the imagery data from the EO sensor, and to execute CV algorithms. Since the CV algorithm (software) is one of the key elements in autonomy research, the existing algorithm needs to be evaluated and possibly revised. This report reviews the current state-of-the-art open source algorithm supporting target detection and tracking. Finally, this thesis research looks into how COTS UAVs can be adapted to support a typical military mission such as search and rescue.

C. THESIS CHAPTER OUTLINE

This remainder of the thesis is organized as follows:

- Chapter II reviews the architecture, including the hardware and software configuration, of the baseline system. In addition, the test results from the experimental test case and the potential problems are identified and briefly discussed.
- Chapter III discusses system considerations relevant to this thesis and the proposed changes based on the current available technologies.
- Chapter IV presents the architecture, including the hardware and software configuration, of the advanced system. The modifications in terms of the hardware and software are described.
- Chapter V discusses the results of the test and evaluation of the proposed system in a typical military mission, namely a SAR.
- Chapter VI summarizes the thesis findings and suggests recommendations for future research.

II. REVIEW OF BASELINE SYSTEM

This chapter reviews the existing system architecture, key details of hardware and software solutions, test results, and potential problems that were identified in the past NPS master's thesis titled "Assessment of an Onboard EO Sensor to Enable Detect-and-Sense Capability for UAVs Operating in a Cluttered Environment" (Ang 2017).

A. SYSTEM ARCHITECTURE

The UAV system utilized in Ang's thesis is the Matrice 100 (M100) platform by the Da Jiang Innovation (DJI) Science and Technology Company Ltd. The design hierarchy of the M100 allows a highly flexible architecture, where vision-based payload electronics can be integrated onto the system and perform the additional tasks on top of the features provided by the baseline unit. Figure 1 shows the M100 system setup that was developed. The modules listed in the red textbox are the baseline hardware required by the M100 to provide essential flight capability, and the modules listed in the green textbox are the payload to enable user-defined functions (object detection and tracking).

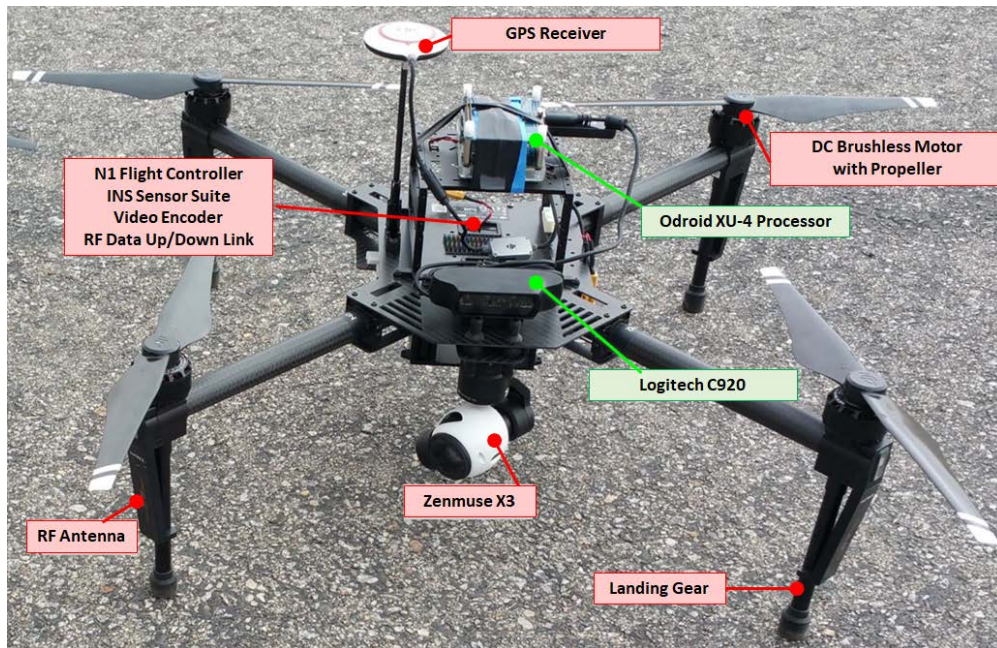


Figure 1. Baseline M100-Based Platform. Adapted from Ang (2017).

Ang's (2017) thesis utilized the Department of Defense Architecture Framework (DoDAF) as the systems engineering approach to define the operational view (OV) and system view (SV) of the system. The OV diagram focuses on the behaviors and functions describing the enterprise mission aspects, while the SV diagram describes the systems and services supporting the operational mission activities (Beery 2017). Figure 2 presents the SV of the baseline platform.

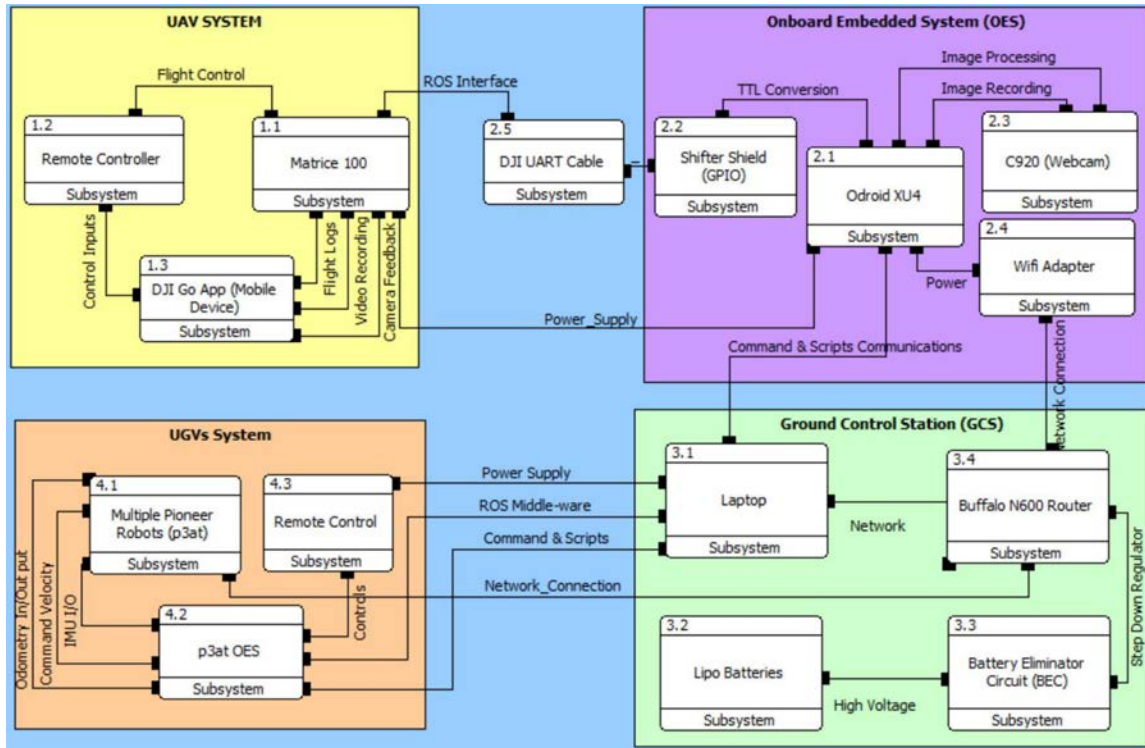


Figure 2. System View of the Baseline Platform. Source: Ang (2017).

B. STANDARD M100 HARDWARE CONFIGURATION

The M100 baseline hardware consists of the minimal essential items required for the UAV to operate in a human-in-the-loop flight environment. The M100 base unit is a quadcopter configuration with one direct-current (DC) brushless motor and one propeller attached to the end of its carbon-fiber rod. The M100's N1 flight controller receives and processes the data from the local sensor suite (i.e., gyroscope, compass, barometer) and GPS receiver prior to sending the control information to each individual motor via its electronic speed control (ESC) circuits, which are designed to control the motor's thrust, revolutions per minute (RPM), and direction. The M100 also contains a gimbal camera (Zenmuse X3), which provides video feed to the human operator via the radio frequency (RF) up/down link. Figure 3 illustrates the Matrice 100 base unit, with the arrows representing the direction for the propellers in flight.

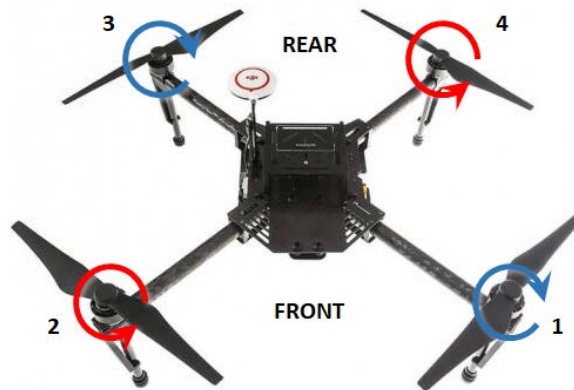


Figure 3. Matrice 100 Base Unit. Adapted from DJI (2017).

Basically, the movement of the M100 quadcopter UAV originates from the remote control stick; the signals are sent via RF data up/down link and are passed to the N1 flight controller to execute the desired movement by directing the ESC and motors to increase or decrease speed. In order for the quadcopter to have a vertical lift, a force must be created that must equal or exceed the force of gravity. The spinning of the quadcopter propellers forces air downward, and a reaction force with equal size and opposite direction pushes

upward onto the UAV platform. Hence, the faster the propellers spin, the greater the lift force will be, and vice versa. In the vertical plane, the UAV is designed to:

- Hover—Zero net force. The quadcopter UAV produces an upward force that is equal to the gravitational force.
- Ascend—Positive net force. The quadcopter increases the thrust of the propeller to generate an upward force that is greater than the gravitational force.
- Descend—Negative net force. The quadcopter reduces the thrust, resulting in an upward force that is less than the gravitational force.

Next, in order to move forwards, backwards, and sideways or to rotate during flight, the quadcopter produces an angular momentum by varying the speed of its motor configuration. As shown in Figure 3, the quadcopter motor configuration shows that motors 2 and 4 are rotating counterclockwise (CCW motors) and motors 1 and 3 are rotating clockwise (CW motors). Hence, for the UAV platform to execute:

- Yaw—Rotation of the system to either its right or left. This requires a decrease in the spin of CW motors 1 and 3, with an increase in the spin of CCW motors 2 and 4. This produces a positive angular momentum to rotate counterclockwise.
- Pitch—Movement of the system either forward or backward. For forward pitch movement, the motors 3 and 4 located at the rear of the system must increase in spin rate, while the motors 1 and 2 (front motors) decrease in spin. The greater force from the back of the system tilts the system forward, pushing the system into forward flight motion.
- Roll—Movement of system to either the right or left. For right movement, a similar concept to pitch movement applies; the motors 1 and 4 increase their spin rate, while motors 2 and 3 decrease in spin. The greater force from the right side of the system creates a roll effect of the system to its right.

The M100 system is powered by a single TB47D / TB48D, 6S LiPo battery with voltage rated at 22.8V, and capacity of 4700 mAh / 5700 mAh. The M100 has its own power distribution circuitry to provide regulated power to the base unit hardware modules. The M100 power circuitry has two additional ports for 22.8V unregulated voltage, which can be supplied to additional experimental hardware units. Table 1 summarizes the key specifications of the M100 system.

Table 1. Key Specifications of the Matrice 100 UAV. Source: DJI (2017).

Parameters	Values
Performance	
Hovering Accuracy (P-Mode with GPS)	Vertical: 0.5m, Horizontal: 2.5m
Max. Angular Velocity	Pitch: 300°/s , Yaw: 150°/s
Max. Tilt Angle	35°
Max. Speed of Ascent	5 m/s
Max. Speed of Descent	4 m/s
Max. Wind Resistance	10 m/s
Max. Speed	22 m/s (ATTI mode, no payload) 17 m/s (GPS mode, no payload)
Battery Voltage/Capacity	TB47D : 22.8V / 4500 mAh TB48D : 22.8V / 5700 mAh
Hovering Time w/o payload (with Zenmuse X3)	19 mins with TB47D 23 mins with TB48D
RF Data Up/Down Link	
Operating Frequency	5.725 ~ 5.825 GHz (Video) 2.400 ~ 2.483 GHz (Data)
Estimated Transmission Distance (Line-of-sight)	CE: 3.5 km FCC: 5 km
Structure	
Diagonal Wheelbase	650 mm
System Weight	2355 g with TB47D 2431 g with TB48D
Maximum Takeoff Weight	3600 g
Expansion Bay Weight	45 g
Zenmuse X3 Gimbal Camera	247 g

The M100 is integrated with a Zenmuse X3 gimbal camera system for live video feed during the flight. This X3 camera contains a Sony complementary metal oxide semiconductor (CMOS) sensor with 12.4M pixels, which provides 4K / FHD / HD quality video recording to the user. The three-axis gimbal controller receives data from the N1 flight controller to compute the required angular motion correction to the camera for video stabilization during flight, and to make control changes to point the camera according to user-defined inputs. The Zenmuse X3 has also its own proprietary interfacing protocols that enable efficient data transfer to the DJI products. Table 2 summarizes the key specifications of the DJI Zenmuse X3 gimbal camera.

Table 2. Key Specifications of DJI Zenmuse X3 Gimbal Camera.
Adapted from DJI (2017).

Parameters	Values
Model	Zenmuse X3 (FC250)
Sensor	Sony EXMOR 1 / 2.3" CMOS
Shutter Type	Global Shutter
Lens	Field of View (FOV): 94° Focal Length (35 mm Equivalent): 20 mm Aperture: F/2.8
Video Recording	<u>UHD (4K):</u> 4096 x 2160 3840 x 2160: <u>FHD (1080p):</u> 1920 x 1080 <u>HD (720p)</u> 1280 x 720
File Format	Photo: JPEG, DNG Video:MP4 in .MOV
Photography Modes	Storage on MicroSD Card Single Shot, Burst (3, 5, 7 frames per sec)
Interface	Proprietary of DJI. Undisclosed.

C. BASELINE SYSTEM HARDWARE ADDITIONS

The additional payload electronics are placed on the UAV system to perform image processing functions. In the existing setup, the system utilized an onboard embedded processor (Odroid-XU4) fitted with supporting peripherals (i.e., WiFi adapters) that are capable to handle the computationally intensive processes of the CV algorithm in a small form-factor solution. In addition, the Logitech webcam (C920) is connected to the Odroid processor via USB3.0 ports to provide the imagery data. This setup simplifies the communication between base unit and CV electronics to a universal asynchronous receiver transmitter (UART) interface. Figure 4 shows the overview of the payload electronics used in the UAV system developed by Ang (2017).

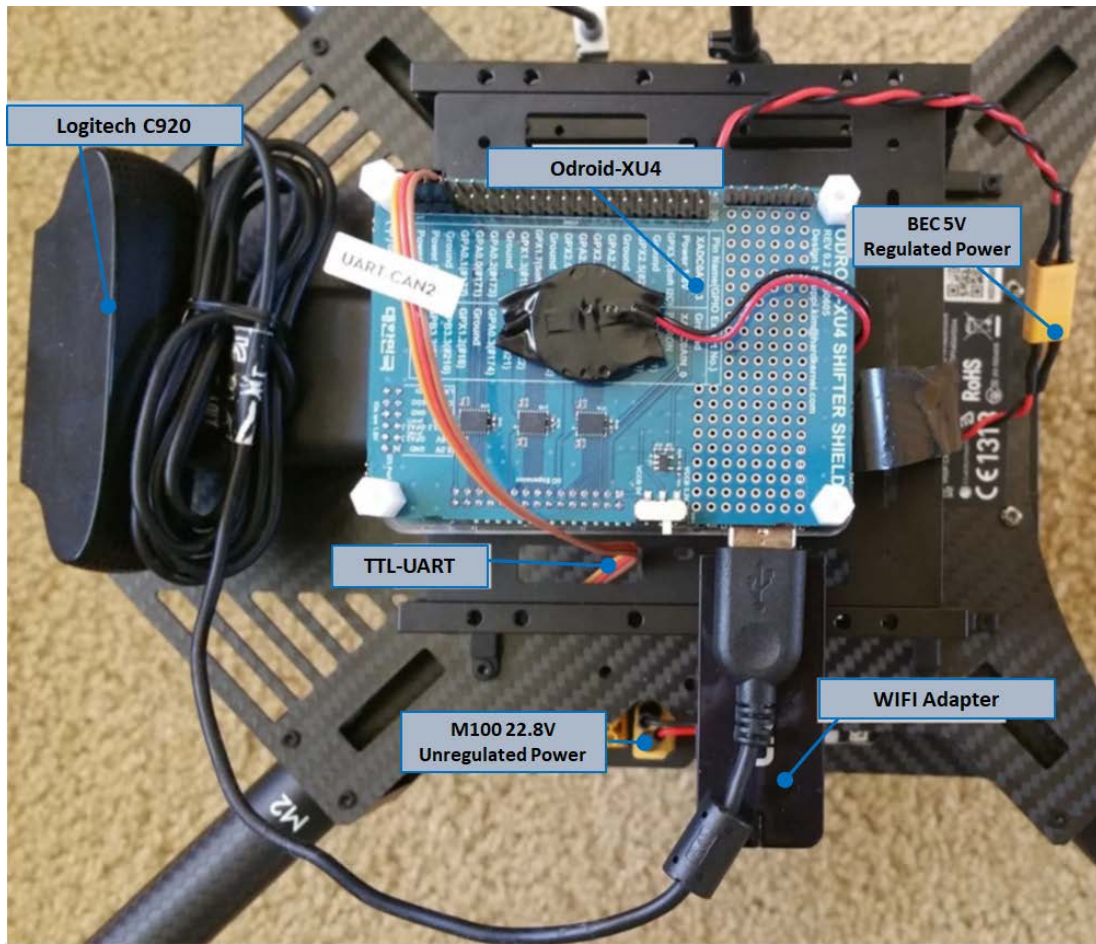


Figure 4. Overview of the Added Payload Electronics. Adapted from Ang (2017).

The main processor embedded within the Odroid-XU4 is a Samsung Exynos 5422 central processing unit (CPU) element, with quad-core of ARM-Cortex-A15 clocked at 2 GHz and quad-core of ARM-Cortex-A7 clocked at 1.2 gigahertz (GHz) (Samsung 2014). The processor has also two gigabytes (GB) low power double data rate 3 (LPDDR3) random access memory (RAM) mounted as a package-on-package (POP) on top of the CPU element to reduce its footprint on the printed circuit board form factor. The Odroid-XU4 receives its regulated power through a battery eliminator circuit (BEC), which taps into the one of the 22.8V unregulated power sources provided by the M100 power circuitry. The BEC is programmed to down-convert the 22.8V unregulated voltage to 5V constant regulated voltage for the Odroid-XU4. The Odroid-XU4 contains internal power circuitry to supply the Logitech C920 webcam and the WiFi adapter with the correct voltage through the universal serial bus (USB) connection. Table 3 summarizes the key specifications of the payload processor.

Table 3. Key Specifications of the Odroid Processor. Adapted from Hardkernel (2014).

Parameters	Values
CPU	Samsung Exynos 5422 Octa-core CPU - 4x Cortex-A15 2 GHz - 4x Cortex-A7 1.2 GHz
3D Accelerator	Mali-T628 MP6 (Support OpenGL and OpenCL)
RAM	2 GB LPDDR3 RAM
Memory	eMMC 5.0 HS400 Flash Storage
Interface	USB 3.0 – Logitech C920 Webcam USB 3.0 – Wifi Adapter USB 2.0 – Available Gigabit Ethernet Port – Available
Power In	4.0A @ 5V (Peak Consumption) 2.7A @ 5V (Nominal Loading)
Miscellaneous	GPIO Interface - +1.8V LVTLL Logic

The added payload also includes a USB 2.0 webcam (Logitech C920) as its main video input feed into the autonomous detection and tracking algorithm executed by the Odroid-XU4 CPU element. The resolution of the video feed is kept and interchangeable between 640-by-480p or 800-by-600p to achieve balance between the FOV, data transfer between devices, and bandwidth to process between frames. The Logitech C920 has a rolling shutter sensor mechanism to reduce the complexity of the electronic module and its cost of production. In his thesis, Ang (2017) suggested that the rolling shutter produces image artifacts or distortion when capturing moving objects at a frequency higher than the shutter rate. Since the Logitech C920 is mounted as a fixed structure on the system, it will look at a certain facing direction with a FOV of 78 degrees. Table 4 summarizes the key specifications of the Logitech C920 webcam.

Table 4. Key Specifications of Logitech C920 Webcam. Adapted from Logitech (2016).

Parameters	Values
Model	Logitech C920
Sensor	Aptina 1/3" CMOS
Shutter Type	Rolling Shutter
Lens	FOV: 78° Focal Length (35 mm Equivalent): 3.67 mm Aperture: Not Available
Video Recording	<u>FHD (1080p):</u> 1920 x 1080 <u>HD (720p)</u> 1280 x 720
File Format	Video stream in H264 compression
Photography Modes	No Storage Capability
Interface	Hi-Speed USB2.0

D. SOFTWARE CONFIGURATION OF BASELINE PLATFORM

The software configuration for the M100 system can be broken down into the software within the base unit, the N1 flight controller, and the added payload electronics, which is mainly the Odroid-XU4 hardware. The simplified software architecture of the M100 system, depicted in Figure 5, shows that DJI had limited developer access to their products. In fact, through the research efforts and discussions with a collaborating research team at the University of Missouri – Kansas City (UMKC), it was concluded that DJI provided these “blackbox” solutions to protect their intellectual property rights and avoid modification of their software program. Therefore, the experimental hardware must be integrated as a payload and communicate with the N1 flight controller via the UART interface and robot operating system (ROS) protocol standard. The DJI software development kit (SDK) ROS wrapper (see Appendix C) for communicating with the N1 flight controller is discussed in a later section.

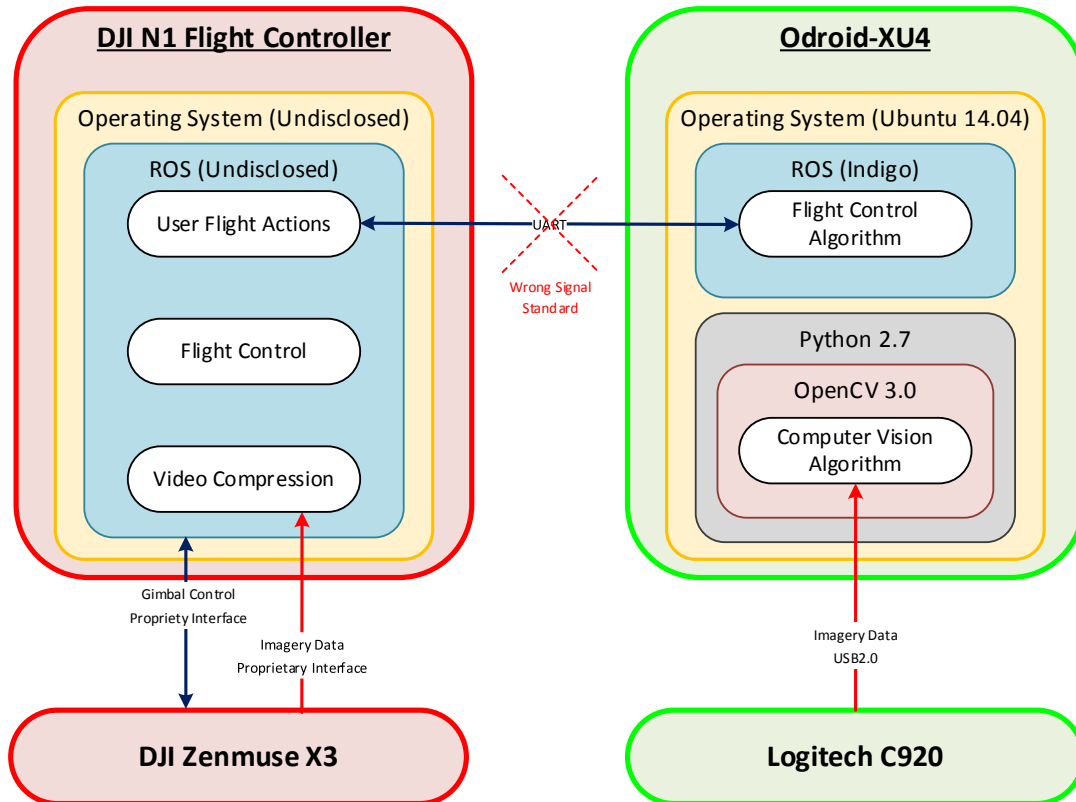


Figure 5. Software Architecture of Baseline System

The M100 system is controllable via the DJI Go application, which can be run from a mobile device (i.e., a tablet or smartphone) connected to the M100 remote controller. The remote controller is embedded with the RF data up or down link to communicate with the M100 system. All data exchanges between the M100 platform system and the remote controller are handled seamlessly by the DJI application. In the research, the system utilized firmware v1.3.100 for the M100 platform, v1.8.100 for the Zenmuse X3, 1.8.0 for the DJI remote controller, and v3.1.6 for the DJI Go application (Ang 2017).

As mentioned earlier, the software within the N1 flight controller is built on a proprietary architecture controlled by DJI. Hence, the M100 system requires the user to integrate additional payload to communicate and control the operation of the M100 system. DJI has released information and coding examples via the DJI SDK on their developer webpage. In order to implement any user-defined solution, the payload must be installed with DJI SDK and use Linux ROS as the means for communication.

The Odroid-XU4 hardware has Ubuntu 14.04 LTS as its main operating system software, loaded with ROS-Indigo (version name), and the OpenCV is cross-compiled as an add-on package to Python 2.7 in a virtual workspace environment. The ROS-Indigo provides the environment in which the DJI SDK is to be installed and the workspace for the flight control code (if any) to be implemented.

The Python2.7-OpenCV cross-compilation provides the framework for the autonomous detection and tracking algorithm code to be executed. The Python scripts are used for extracting data (via the USB interface) from the Logitech C920 (visual sensor), and for performing the required image processing to identify the moving objects within the image dataset (Ang 2017). The OpenCV enables an open source image processing library to be utilized, and for taking advantage of the multi-core processing and hardware acceleration capability of the mathematical functions to execute the computationally-intensive application.

E. THREAT DETECTION ALGORITHM

The CV algorithm used in the baseline system (shown in Figure 5) is the result of a joint collaboration between Purdue University and the Naval Postgraduate School (Li et al. 2016). Among the different techniques to support the detection and tracking of moving objects, the Purdue/NPS CV algorithm utilizes the motion-based approach, which involves a combination of background subtraction and optical flow techniques to process the image dataset. The background subtraction technique removes the pixels with the same brightness constancy over time, and thus enables the moving object to be identified (Nuria et al. 1999). This technique, however, will experience poor accuracy in a moving system when the background is ever-changing in the image dataset. The optical flow technique detects the moving object by identifying the local motion vectors through the examination of various sequential frames (Brox and Malik 2011). The accuracy of detection for this technique depends upon the image quality; hence, distorted images reduce the overall accuracy of this technique. Figure 6 illustrates the key components overview of the Purdue/NPS CV algorithm.

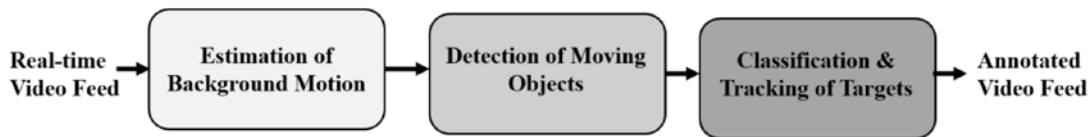


Figure 6. Overview of Purdue/NPS CV Algorithm. Source: Ang (2017).

F. HIGHLIGHTS OF THE BASELINE SYSTEM FLIGHT TESTS

The baseline system was tested and evaluated at McMillan Airfield of Camp Roberts. Test and evaluation (T&E) exercises were conducted with two distinct backgrounds simulating the field and urban test environments. The flight altitude of the UAV was capped at 30 meters (m), and placed in a hovering flight mode. While the UAV was placed in flight mode, the Purdue algorithm was executed on the CV hardware to detect and track the multiple moving objects.

Figure 7 shows the sampled screenshots of the video with varying classifier parameters of the field test environment. Ang's (2017) thesis highlighted observations made from the entire video footage, whereby the classifier parameter for small targets resulted in enhanced sensitivity as compared to the average targets. The average targets classifier yielded more misdetections as the CV algorithm was inefficient in discerning the slower moving UGVs and smaller UAVs.

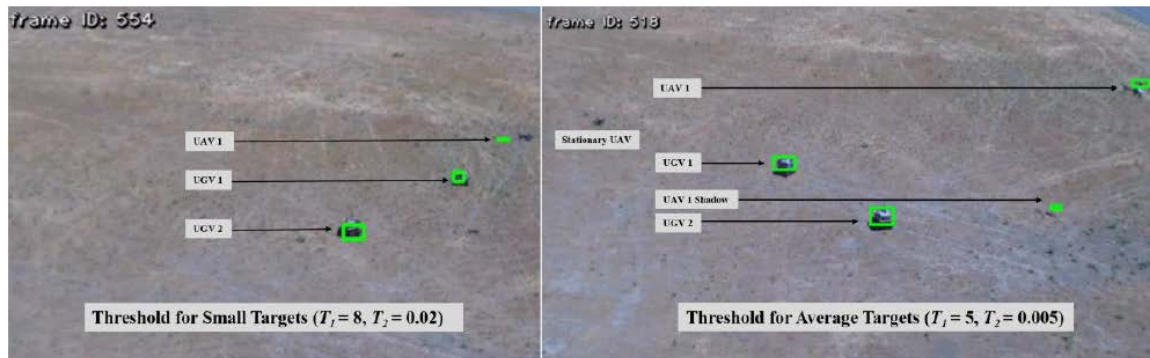


Figure 7. Varying Classifier Parameters in the Field Test Environment. Source: Ang (2017).

Figure 8 shows the sampled screenshots of the video with varying classifier parameters from the urban test environment. Similarly, Ang's (2017) thesis repeated the field experimentation with the visual sensor on the M100 system. The sensor was placed to capture video feed of an urban test environment, which is relatively noisier with background clutter due to the multiple stationary objects (i.e., parked cars, buildings, etc.). Based on Ang's observation, he highlighted that the classifier for small targets had better performance compared to the average targets. The average targets classifier yielded more detection errors.



Figure 8. Varying Classifier Parameters in the Urban Test Environment. Source: Ang (2017).

For the urban test environment, the background clutter of objects proved to be a challenge for the Purdue/NPS CV algorithm and resulted in multiple instances of misdetections. Therefore, this amounted to poor consistency in detection and tracking efficiency. Using the recorded video from the urban test environment, the classifier parameters were fine tuned to enhance their sensitivity. The threshold for clustering and grouping of points was lowered to capture and maintain tracking of the minute movements of the targets in a noisy background (Ang 2017). Figure 9 shows the sampled screenshot of the video with tuned classifier parameters to detect and track human subjects, UGVs, and UAVs simultaneously.



Figure 9. Post-flight Experimental Testing with Tuned Classifier Parameters.
Source: Ang (2017).

G. IDENTIFIED PROBLEMS OF THE BASELINE PLATFORM

This section presents on the problems of the baseline platform based on the test data collected during the field experimental

1. Sensitivity of the Classifier Parameters

The sensitivity of the classifier parameter defines the threshold for the detection of the moving objects in the input video feed. When an oversensitive classifier parameter is used, it results in false positives or unwanted noise in the data. When the UAV was moved during Ang's experiment, the undesired results from an oversensitive classifier parameter of the Purdue/NPS CV algorithm became worse. Hence, Ang (2017) suggested that there is a need to fine tune the sensitivity of the algorithm to improve the capture result of the objects while blocking out the unwanted noise. Figure 10 shows the unwanted noise effects that resulted from the oversensitive classifier parameters.



Figure 10. Unwanted Noise from the Oversensitive Classifier Parameter. Source: Ang (2017).

2. Overlap in the Bounding Box

The version of the Purdue/NPS CV algorithm used in Ang's (2017) thesis research occasionally produces large extended bounding boxes (depicted in green in Figure 11) when the detected moving objects cross paths and overlap each other. The appearance of large extended bounding boxes affects system performance, reducing effectiveness in distinguishing the individual objects.

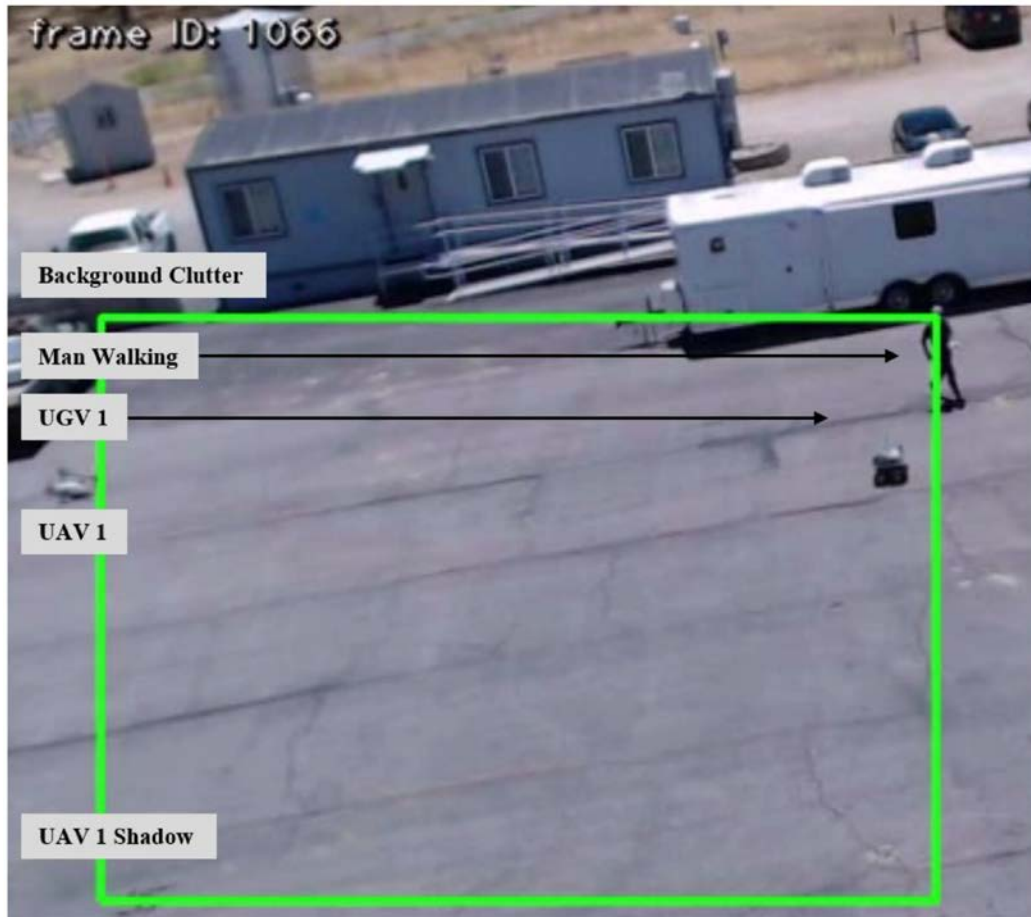


Figure 11. Extended Bounding Box over Detected Objects. Source: Ang (2017).

3. Rolling Shutter Effects

Due to the movement of a UAV in flight mode or in a strong wind, the video quality experiences rolling shutter effects, depicted in Figure 12. The rolling shutter effects of the video degrade the effectiveness of the Purdue/NPS CV algorithm in detection and tracking of the objects (Ang 2017). Figure 12(a) illustrates where the algorithm lost track of UAV 1, following the tilted planar axis of the system in strong wind. Figure 12(b) highlights distortion in image quality due to the rolling shutter effects, following the tilted planar axis of the system in strong wind. Figure 12(c) highlights another example of rolling shutter effect on image quality as the M100 transited in flight. Figure 12(d) shows a positive example of the image without rolling shutter effect. Furthermore, there is no available post image processing technique to correct for these rolling shutter effects in the degraded imagery.

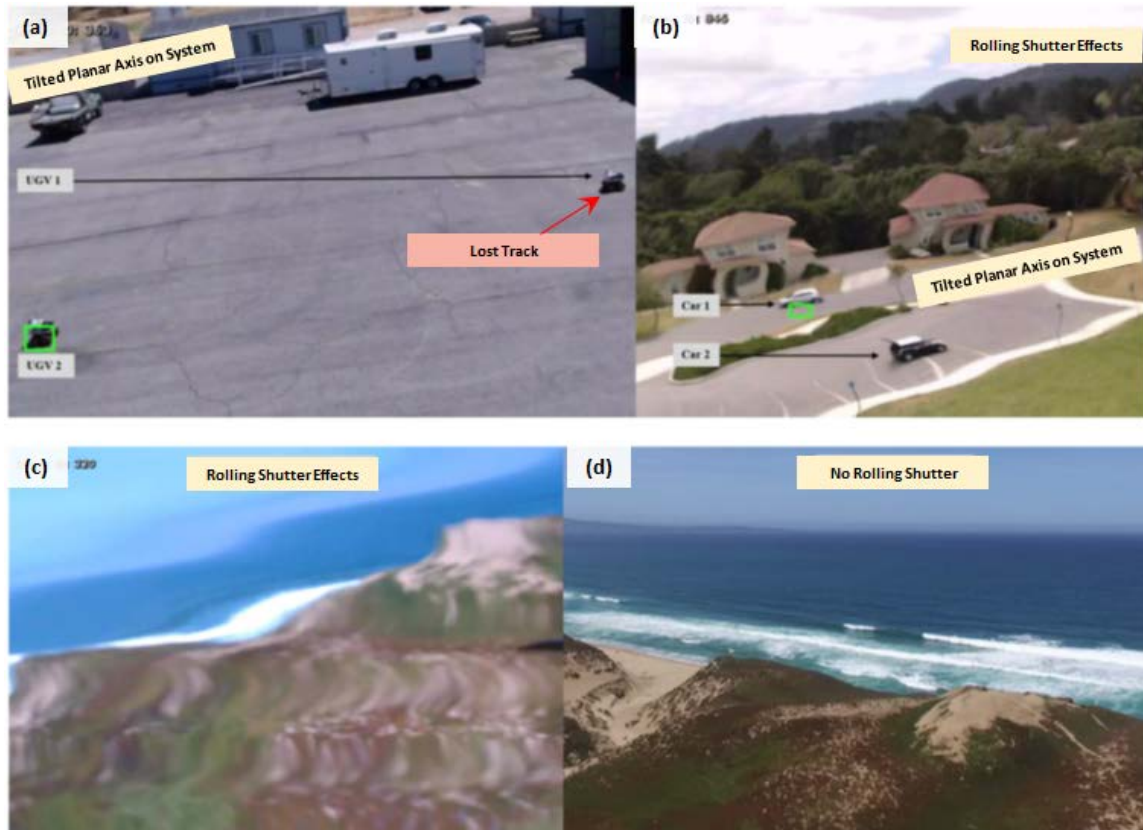


Figure 12. Example of Image Quality under Various Conditions. Adapted from Ang (2017).

III. TECHNOLOGY ENABLERS AND AREAS FOR BASELINE SYSTEM IMPROVEMENTS

This chapter addresses the proposed means to mitigate the deficit of the baseline system reviewed in the previous chapter.

A. PROPOSED CHANGES

Based on the summary of identified problems, discussed in Section II.G, it is evident that rolling shutter effects contribute negatively to the test outcome. Rolling shutter is a shutter mechanism, electronic or mechanical, for capturing images in which the scenes exposed on the sensor are read out in either a vertical or horizontal manner (Tirosch 2012). In other words, the top and bottom points of the scene image are not captured at the same moment. This is why images might appear skewed under certain circumstances.

The appearance of skewed objects in an image can also be referred to as geometric distortions. This phenomenon occurs when images are captured from a vibrating EO sensor, UAV in-flight moving motion, or fast moving objects within the scene. Depending on the circumstances, multiple effects might occur at the same time. In Figure 13(a), the fast spin rate of the helicopter motors causes a “jello effect” on the helicopter propellers, where they appear to be wobbling. Next, for Figure 13(b), the captured images appear to have a skewed effect on the building, where all the buildings appear to lean in a common direction.

Hence, it is evident that the rolling shutter effects influence the video quality available from the Purdue/NPS CV algorithm on the CV hardware. Furthermore, the degraded video can lead to multiple instances of false positives and noise. Therefore, the change in EO sensor from a rolling sensor mechanism to a global shutter mechanism generates significant improvement in the performance of the CV hardware.

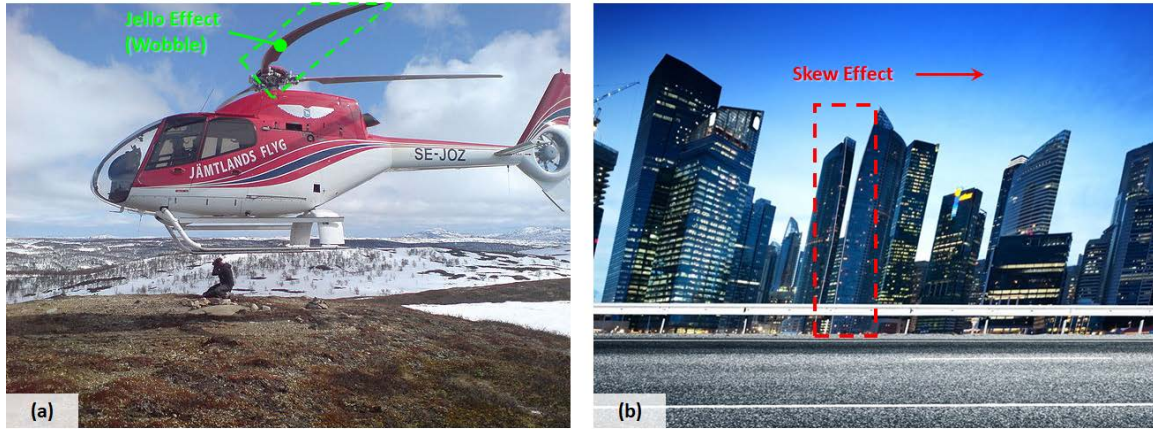


Figure 13. Examples of Rolling Shutter Effects. Adapted from Jonen (2007) (left); Adler (2016) (right).

Beside the hardware changes, a review of the algorithm that provides the object detection and tracking is significant. The current version of the Purdue/NPS CV algorithm has some flaws, which require improvements. Since the research students working on the algorithm have graduated, there is a lack of support for improvements in the algorithm coding. As the objective of this thesis research is to be able to perform system integration and evaluation of the CV algorithm, the approach taken is to mitigate the lack of support for the Purdue/NPS CV algorithm by creating a generic system architecture to enable different open-source CV coding to be evaluated.

Hence, the proposed changes to the baseline system focus on the payload electronics. A new payload processor, DJI Manifold, has been introduced to replace the Odroid-XU4. Recalling that the DJI Zenmuse X3 has a proprietary interface, the DJI Manifold has been designed to decode this proprietary interface. This enables the use of the Zenmuse X3 (a gimbal and global shutter EO solution) in the development of this thesis.

B. EO SENSOR TECHNOLOGIES

Depending on the application, the EO sensors can be separated into two different classes (i.e., active sensors and passive sensors). Active sensors usually possess their own energy source, which is used to illuminate the object. The radiation energy from the source is reflected from the object and measured by the sensor. In comparison, the passive sensor uses naturally occurring radiation, without emitting any energy. This section focuses on the passive EO sensor (i.e., the Zenmuse X3) found on the M100 baseline system.

1. Imaging Considerations

Since images contain the information required for the CV algorithm application, it is important to understand the definition of an image. According to Robert C. Harney (2013), an image can be defined by multiple spatial dimensions that are related to the physical properties of an object or scene. Table 5 shows the listed of potential spatial dimensions and physical properties that characterize the image. Hence, the image from the DJI Zenmuse X3 can be seen as a distribution of reflected intensity over azimuth and elevation angles (Harney 2013).

Table 5. Potential Object Characteristics that Form Images. Source: Harney (2013).

Potential Spatial Dimension	Potential Physical Properties
Azimuth (or Bearing)	Color
Elevation Angle	Reflectivity
Range	Reflected Intensity
Cartesian Coordinates (x, y, z)	Radiance
Depth (or Altitude)	Concentration
Map Coordinates	Transmittance (or Absorptance)
Cross-Range	Velocity
	Temperature
	Range
	Radar Cross Section

2. Sensor Technologies

The two imagery sensor types widely used in EO sensors are the Charge Coupled Devices (CCD) and the Complementary Metal Oxide Semiconductors (CMOS). The major distinction between the two sensor technologies is how the sensor reads the individual pixel signal formed within the sensor, as illustrated in Figure 14.

Typically, the CCD sensor employs a global shutter mode. In the global shutter mode, every pixel on the detector is exposed simultaneously to the reflected intensity of the object (Qimaging 2014). Due to this exposure mechanism, the global shutter detector has an advantage in capturing changing scenes. Nevertheless, one of the drawbacks of the global shutter is the slower frame rate due to the readout of the pixel data. The readout rate is dependent on the analog-to-digital (A/D) converter, which receives and digitizes the analog value of each individual pixel. In addition, the CCD sensors do not build a large detector (high pixel count) to prevent further slowing down of the frame rate.

For CMOS technologies, the sensor uses an A/D converter for each and every column of pixels on the sensor, which is known as a rolling shutter (Qimaging 2014). The workload to digitize the pixels is shared among the parallel A/D converters in each column. Hence, this results in a small delay for each row's readout and contributes to a faster frame rate.

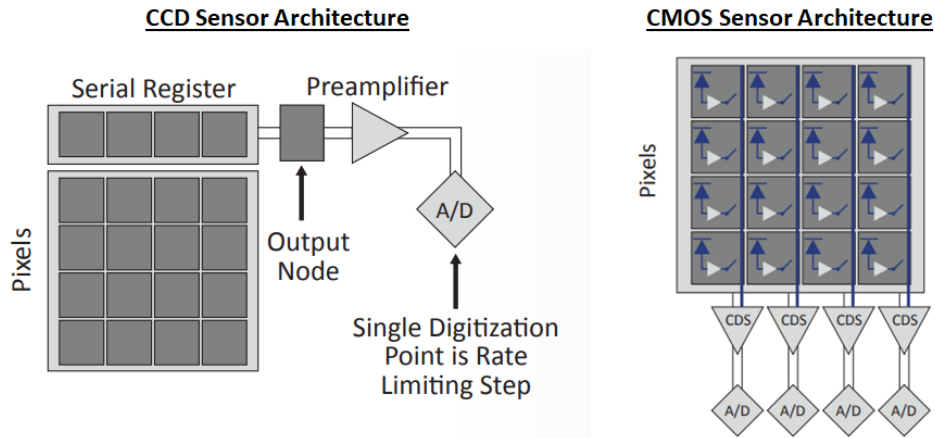


Figure 14. Sensor Architecture of CCD Sensor (left) and CMOS Sensor (right).
Adapted from Qimaging (2014).

Since the shutter mechanism of an EO sensor determines how and when light is recorded during exposure, the two fundamental types of electronic shutter are the global shutter and the rolling shutter.

The sensor exposure with a global shutter is either on or off during exposure (a single simultaneous exposure). Depending on the readout rate of the sensor, a moving object will be illuminated in a rapid sequence. Unlike with the global shutter, the sensor exposure with a rolling shutter is a series of exposures when the capturing function is triggered. Each row of the sensor is exposed in succession, line after line. However, for moving objects, the rolling shutter might create image distortions.

The exposure sequence for global shutter and rolling shutter sensors is illustrated in Figure 15. While the rolling shutter sensor provides the advantages of fast frame rate, it is not without flaws. The overlapping and time delay between each row's exposure contributes to the geometric distortion of moving objects within the captured image.

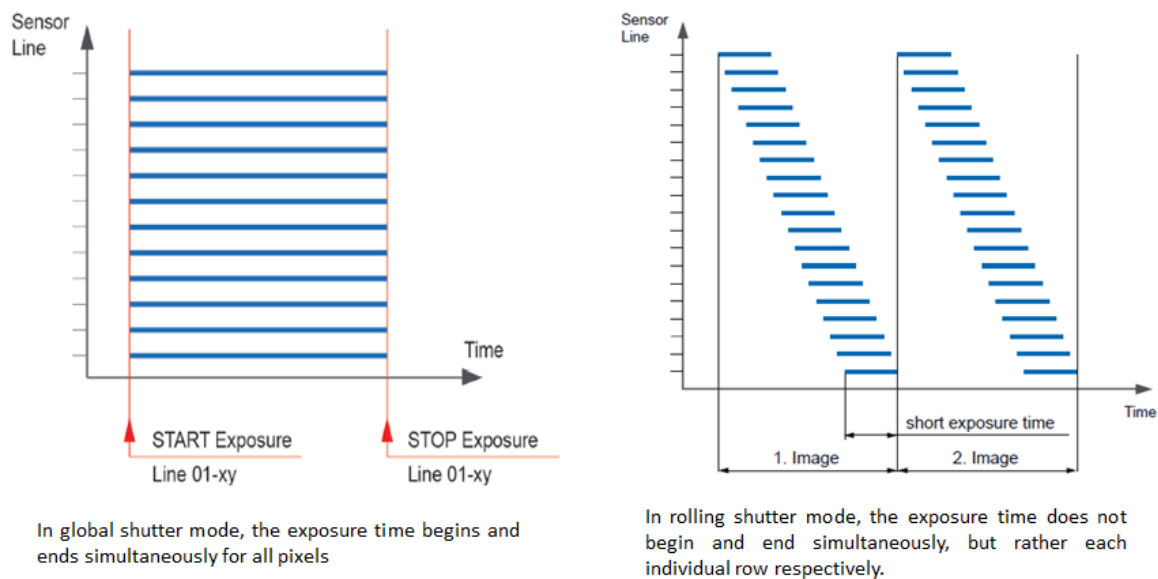


Figure 15. Illustration of Exposure with Global Shutter (left) and Rolling Shutter (right). Source: Lappenküper (2018).

As discussed earlier, the main rolling shutter effect on images is the geometric distortion of an object. The distortion occurs if the object or sensor is moved at a rate faster than the exposure and readout time of the sensor. As the images are reconstituted in a sequential manner of the exposure on individual rows, the rate of the readout will determine the impact of rolling shutter effect on the image. Assuming sensors with global shutter and rolling shutter are used for traffic monitoring, the rolling shutter sensor will likely result in some distortion. In the example depicted in Figure 16, the moving cars appear to be skewed in their geometric dimension for the sensor using a rolling shutter mechanism.

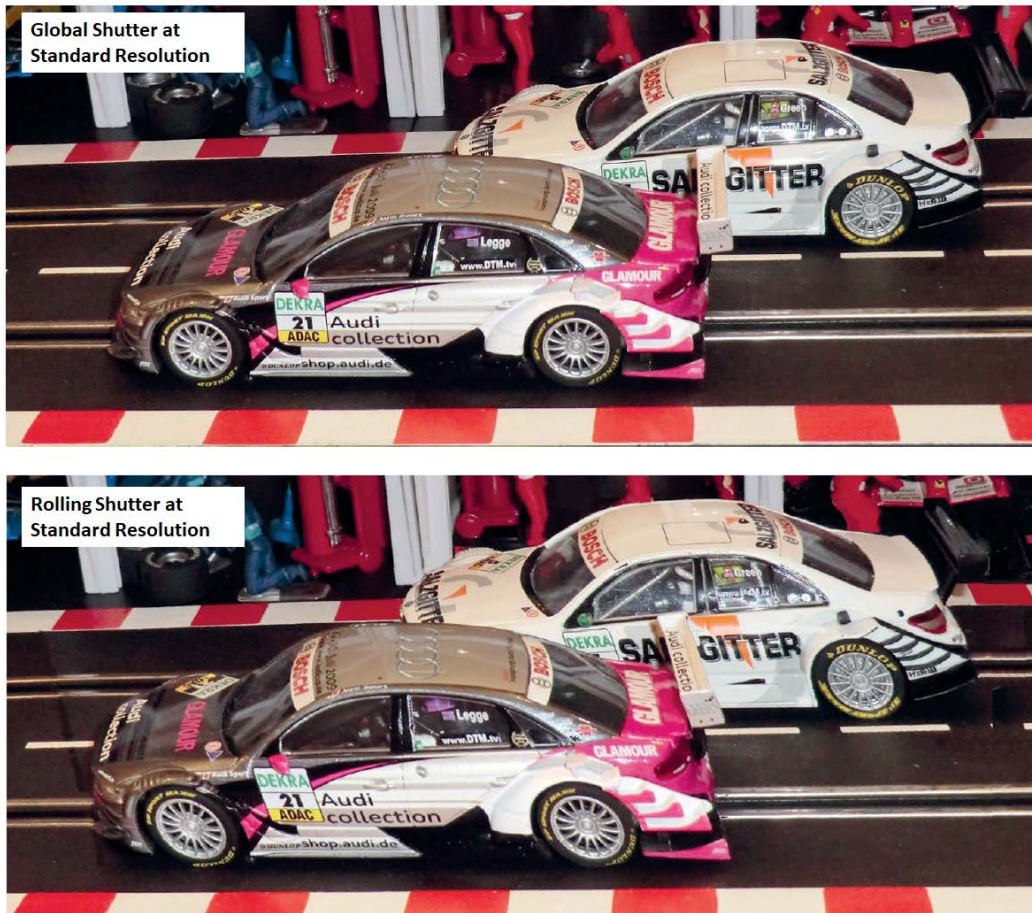


Figure 16. Image Distortion for Global Shutter (top) versus Rolling Shutter (bottom). Source: Lappenküper (2018).

3. Visual Considerations

For the new EO sensor, we know that the DJI Zenmuse X3 uses a Sony EXMOR CMOS 1 / 2.3-inch sensor, with effective pixels of 12.4 megapixels and a global shutter mechanism. Hence, to achieve optimum results in the CV algorithm application, it is important to consider the visual coverage and performance of the EO sensor.

a. *Ground Sample Distance*

The visual coverage of the EO sensor is the actual distance covered in a single image. Considering the M100 system is flying at a certain flight altitude with the EO sensor looking down at the ground, the image captured by the sensor is composed of the object in the foreground and the actual distance covered in the background. By defining the Ground Sample Distance (GSD) as the ground distance, we can compute the flight altitude H that is required to meet the GSD. The computation depends on the camera focal length, the camera sensor width (in millimeters, mm), and the image width (pixels). Figure 17 shows the relationship of the parameters used to find the GSD.

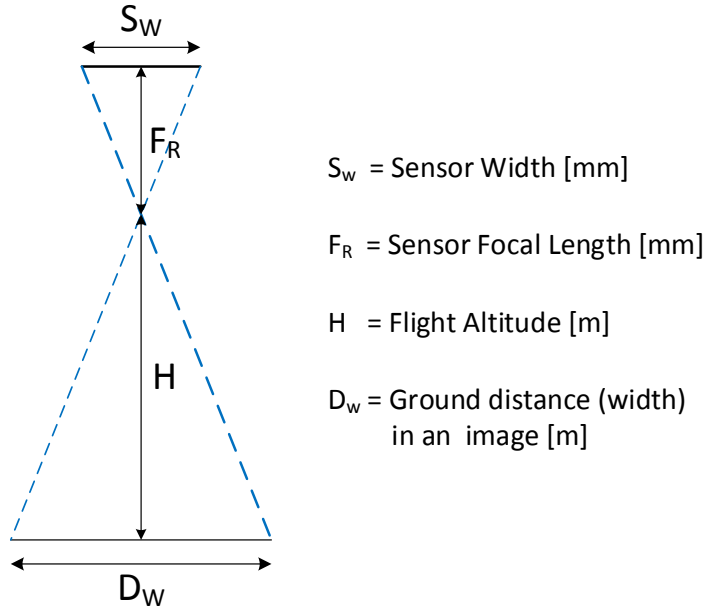


Figure 17. Relationship of Parameters in GSD. Adapted from PIX4D (2017).

From the specification of the Sony EXMOR, the sensor has a dimension width by height of 6.16 mm * 4.62 mm. The real focal length F_R is related to the 35 mm equivalent focal length by:

For 4:3 Sensor Ratio:

$$F_R(mm) = \frac{F_{35} \times S_W}{34.6}, \quad (1)$$

where:

- F_{35} = focal length that corresponds to the 35mm equivalent 20 mm
- S_W = real sensor width 6.16 mm.

Therefore, the real focal length is:

$$F_R(mm) = \frac{F_{35} \times S_W}{34.6} = \frac{20 \text{ mm} \times 6.16 \text{ mm}}{34.6},$$

$$F_R(mm) = 3.56069 \text{ mm}.$$

Assuming the flight altitude and the sensor image resolution is 4000 by 3000 pixels.

The GSD will be:

$$GSD = \frac{S_W \times H}{F_R \times Sensor_{image_wd}}, \quad (2)$$

$$GSD = \frac{6.16 \text{ mm} \times (50 \text{ m} \times 100)}{3.56069 \text{ mm} \times 4000 \text{ pixels}},$$

$$GSD = 2.16 \text{ cm/pixel}.$$

The result shows that when the Zenmuse X3 is operated at a height of 50 m, the camera can see 2.16 cm at ground sample distance per sensor image pixel. If the camera is operated as 4000 * 3000 pixels resolution, the width of a single image footprint on the ground will be:

$$D_w = \frac{GSD \times image_{wd}}{100} = \frac{2.16 \text{ cm/pixel} \times 4000}{100} = 86 \text{ m} .$$

And the height of a single image footprint on the ground will be:

$$D_h = \frac{GSD \times image_{height}}{100} = \frac{2.16 \text{ cm/pixel} \times 3000}{100} = 65 \text{ m} .$$

Figure 18 shows a snapshot recorded at 4000*3000 pixels of the ground coverage over an area of Fort Ord's Impossible City located at GPS coordinates of 36° 37' 11.20" N, 121° 44' 55.90" from the DJI Zenmuse X3 operated at 50 m flight altitude.

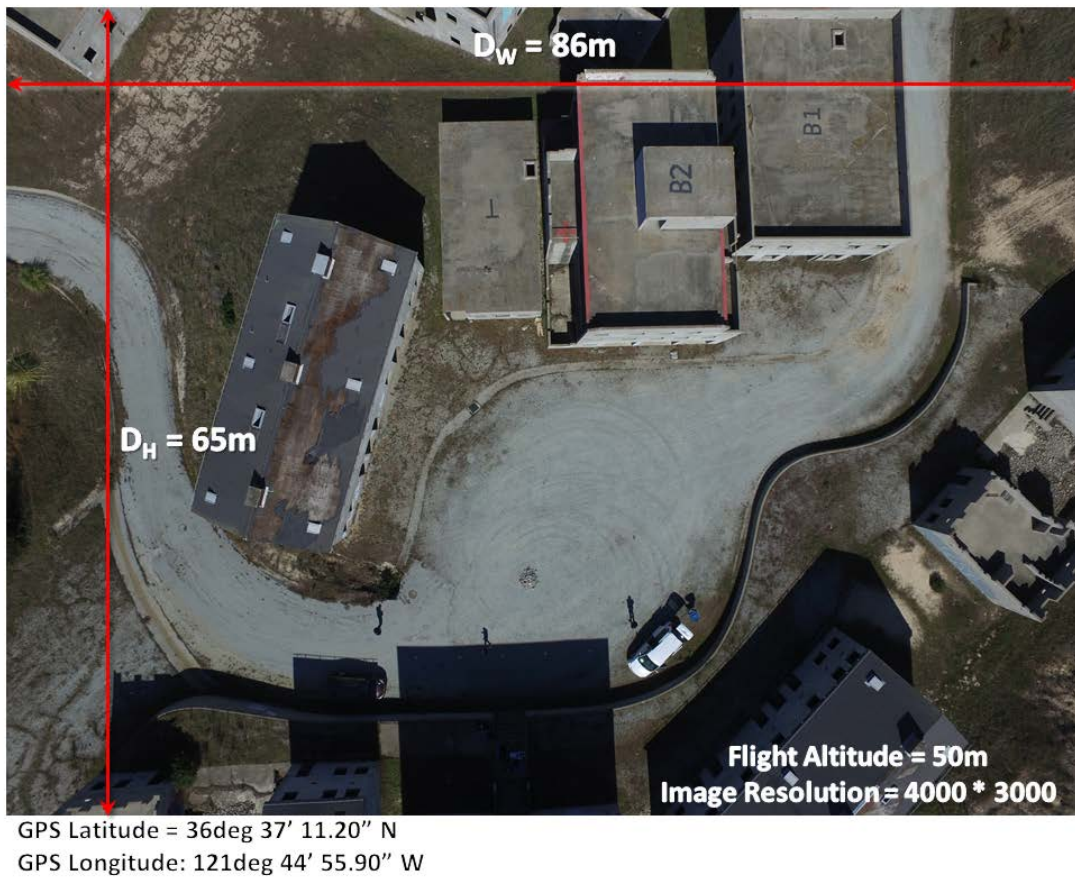


Figure 18. Snapshot of the Ground Distance from UAV (50 m Altitude)

b. Object Resolution

The primary objective for the EO imaging system is object perception (i.e., detection, recognition, identification, etc.) based on image characteristics. The GSD provides a linear resolution of the object that can be observed from the sensor. Since GSD is a function of flight altitude, EO sensor specifications, and image resolution, the measurement of GSD in terms of centimeters per pixel defines the number of pixels that the object (given its perpendicular surface area) represents in the image snapshot captured by the UAV EO sensor. Applying the concept from an earlier section, it is assumed the total surface area of the human body is approximately $1.7\text{m} \times 0.2\text{m} = 0.34\text{m}^2$, and the total surface area of a small UAV is approximately $0.2\text{m} \times 0.2\text{m} = 0.04\text{m}^2$. With the Zenmuse X3 sensor designed to capture images at 4000 by 3000 pixels, the number of pixels represented for the two objects changes according to the distance between the objects and the sensor, as depicted in Figure 19.

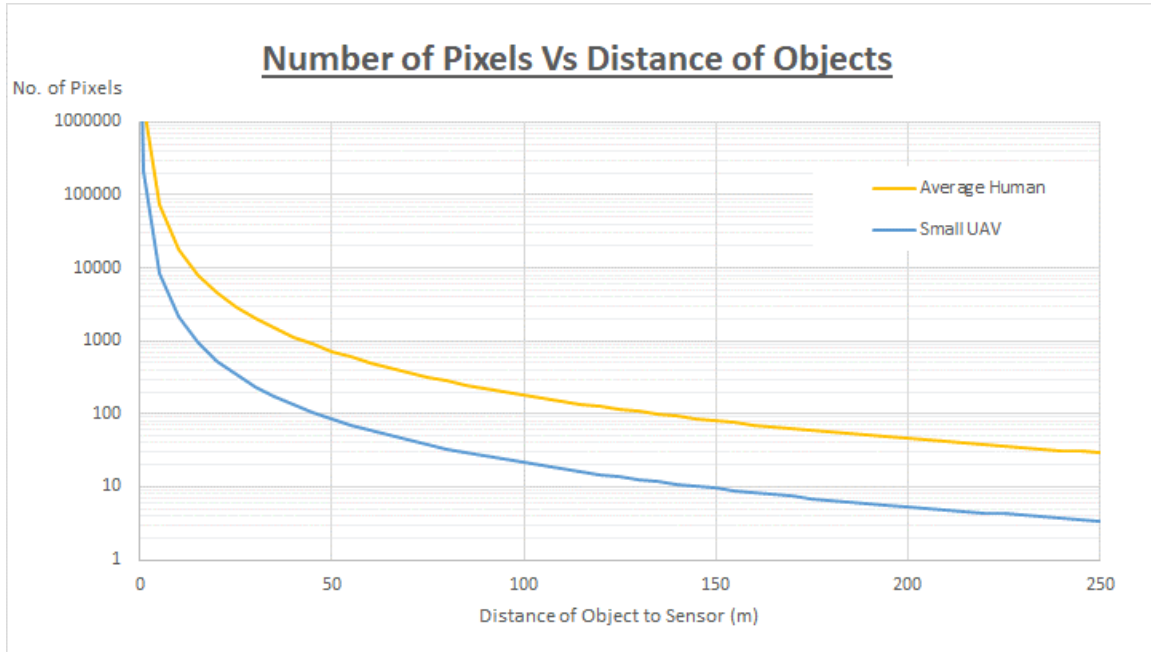


Figure 19. Representation of Pixels Count for Human and Small UAV Detection

c. Exposure Time

For the CV algorithm application, the distortion will affect the performance and accuracy of the algorithm. Hence, for the requirement to image a moving object, the object's size and velocity must be considered for its sampling time while avoiding motion blur of the objects in the image. Motion blur is created when the exposure time for a moving object is set too long causes motion blur. Regardless of the sensor type, the motion blur on objects will occur if the object's velocity is much greater than the exposure performance of the sensor. According to Qimaging (2014), if it is assumed that motion blur of no more than 10 percent is acceptable, the exposure time to image a moving object can be computed as:

$$T \leq \frac{\Delta x}{10v} , \quad (3)$$

where:

- T = exposure time
- Δx = object's length
- v = object's velocity.

For the DJI Zenmuse X3, the sensor has a range of shutter speeds from 8 sec to 1/8000 sec. Let us assume a small UAV with the length of 0.2 m, and flying at 10 m/s, the result shows that:

$$T \leq \frac{0.2 \text{ m}}{10 \frac{\text{m}}{\text{s}}} = 0.02 \text{ s} .$$

Hence, the minimum shutter speed required for the EO system is 1/50 sec. If the scene has sufficient brightness, however, the shutter speed can go even faster to reduce the motion blur in the image. Based on the EO (DJI Zenmuse X3) specification, the EO has the capability to meet the minimum shutter speed.

C. COMPUTING HARDWARE TECHNOLOGIES

The payload hardware functions as the processing element for the execution of the machine algorithm (i.e., CV algorithm and flight control) and to manage the communication interface between the EO sensor and the flight controller. The required onboard payload data processing capability is defined by the amount of data from the EO sensors and the desired level of autonomy. The greater the payload capability that can be integrated onto the UAV system, the more the flexibility in additional technology enablers that can be adapted to the specific needs. This will mean upgrading the EO sensors to a dual mode imaging system (Thermal and Visible), adding Light Detection and Ranging (LiDAR) technology, integrating different GPS systems, or even using different sensor systems to improve accuracy in the target identification and navigation process. However, the increase in payload leads to an increase in payload weight, and this will affect the system's flight performance (mainly in terms of its flight time). Therefore, it is important to achieve balance among the required performance, payload weight, and flight time.

1. Hardware Technologies

In this section, we focus on the proposed replacement payload hardware (DJI Manifold). The DJI Manifold is a computing platform on which the algorithms are run.

a. Processor Architecture

The framework for the processor hardware requires the computing architecture to be scalable and achieve energy efficiency in the overall system design. The general trend between processor performance and power consumption is a directly proportional relationship. The increase in demand for processor performance leads to higher power consumption. Yet, small UAV systems such as the M100 system have a limited power-to-weight ratio, which creates constraints for the payload processor type and weight. In the modern technology evolution of mobile devices, the processor element has delivered powerful computational performance and graphics outputs within a reasonable power budget (Norris 2014).

Among the available embedded processor architectures, the Advanced RISC Machine (ARM) is the market leader due to the demand for its technology. The ARM simply designs the architecture and instruction sets, which are then licensed to hardware partners (i.e., Apple, Samsung, Nvidia) to modify their own ARM processor variants (Norris 2014). The ARM architecture delivers relevant technology to drive system-on-chip (SOC) solutions in solving the challenges for embedded system design. These SOC solutions facilitate both the scalability of hardware to meet requirements of specific tasks and system efficiency for optimized software solutions (Steele 2016).

The existing payload processor (an Odroid-XU4) utilizes a Samsung Exynos chipset featuring quad-cores of ARM Cortex-A15 and quad-cores of ARM Cortex-A7, and integrating ARM's designed Mali Graphics Processing Unit (GPU) design. Since the Cortex-A15 cores have a higher demand in power consumption, the ARM implemented a system scheme called "big.LITTLE" whereby each high-performance core is shadowed by a lower-power, lower-performance core (the Cortex-A7) that takes over the system processing whenever the workload permits (Norris 2014). The implementation of big.LITTLE heterogeneous computing architecture enables the system to achieve savings in power usage according to dynamic computing needs. Figure 20 shows the block diagram for the Odroid-XU4 hardware.

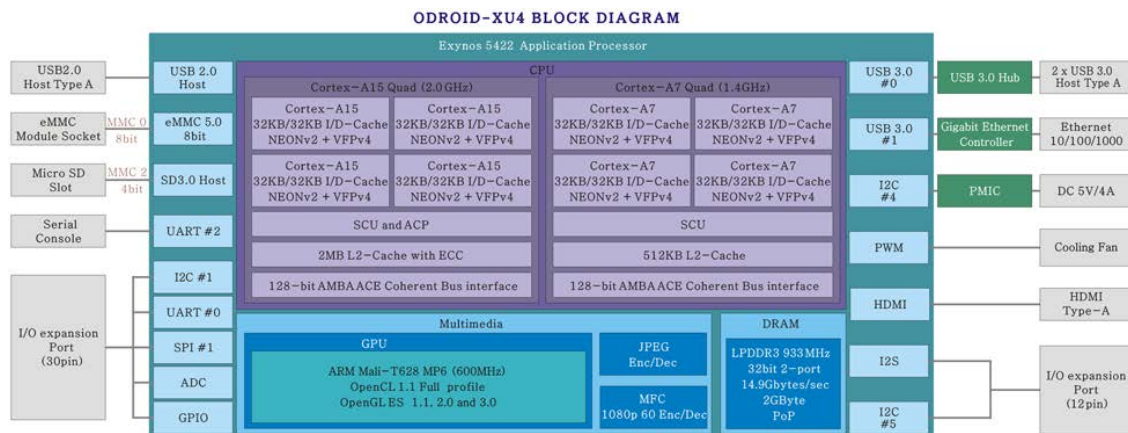


Figure 20. Block Diagram for Odroid-XU4. Source: Hardkernel (2014).

For the new proposed payload processor, the DJI manifold utilizes an Nvidia Tegra K1 (TK1) chipset featuring a quad-core of the ARM Cortex-A15 and single-core of the ARMv7 microcontroller, and 192 Nvidia proprietary CUDA cores (Kepler GPU Architecture). Similar to the Samsung strategies for power saving for embedded processors, the Nvidia TK1 chooses to design with the 4-PLUS-1 CPU core architecture and variable symmetric multiprocessing (vSMP) technology (Nvidia 2014). This architecture enables the management of the high-performance quad-core A15 CPU for complex and performance intensive tasks, and switches to the power optimized ARMv7 microcontroller to handle low performance tasks. Hence, the architecture achieves an intelligent power management scheme according to dynamic computing needs. In addition, the one clear advantage of the Nvidia TK1 is its integration of 192 Nvidia CUDA Kepler GPU cores as part of the co-processor element. The Kepler GPU architecture enables parallel computing applications and delivered high performance computing performance while maintaining efficient energy for mobile usage (Nvidia 2014).

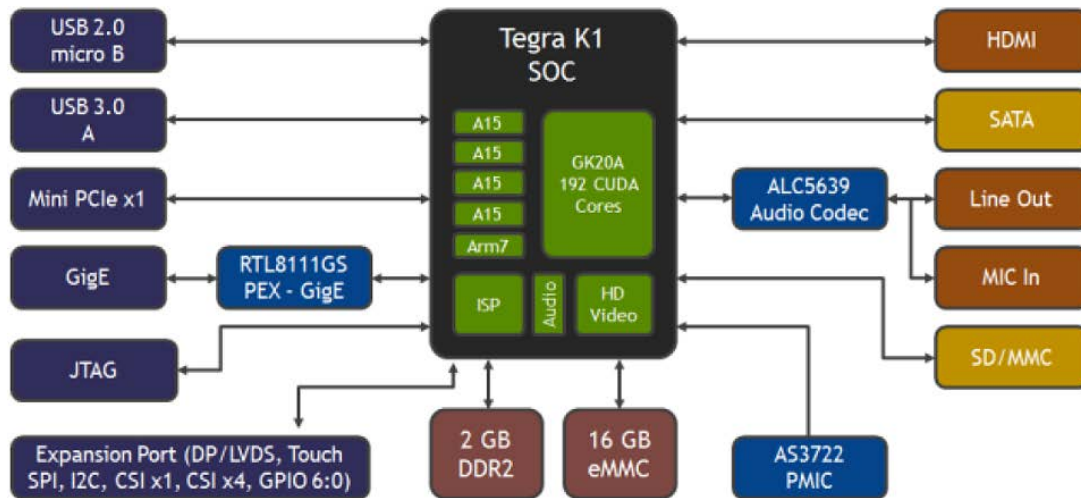


Figure 21. Block Diagram of DJI Manifold. Adapted from Berkeley Design Technology Inc (2014).

b. Memory Interface

The memory interfacing is an essential function to sustain standard operation within the embedded processor. The two critical memory components required by the embedded processor are the RAM and onboard storage, read-only memory (ROM). The RAM serves as temporary storage between the file system, which is stored on the ROM, and the processor. The RAM usually designed for an embedded processor is technically DRAM, with D meaning dynamic. The structure of DRAM stores a data bit on a capacitor cell, and its contents on the DRAM have the ability to be refreshed or changed quickly. In addition, the RAM is different from the flash-based ROM storage, whereby its contents will be lost upon power disconnect. Due to the volatile nature and its storage mechanism, RAM typically has a faster access time as compared to ROM (Schiesser 2012). Hence, RAM interfacing contributes to the overall performance of the embedded processor design. In the modern technology evolution, the RAM memory is designed to be mounted directly on top of the embedded processor (SoC), as depicted in Figure 22, for the most direct access, and its close proximity improves the signal integrity of high-speed signals and reduces the power consumption of devices.

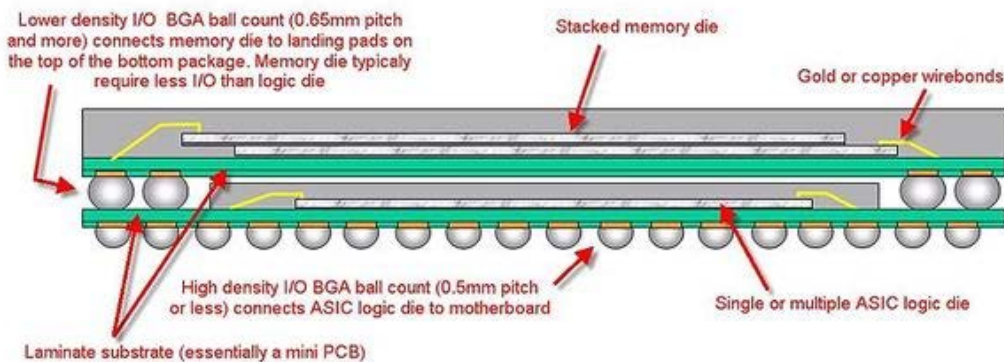


Figure 22. Diagram Illustrating the Package-on-Package Setup of the Memory and Processor. Source: Schiesser (2012).

ROM is part of internal storage and refers to the system files that are not accessible for end users to write on. Typically, the internal storage memory is partitioned into several sections for different purposes, such as system files, cache, application storage, and media

files, etc. The most widely used storage device employed in embedded device design is the embedded Multi-Media Controller (eMMC). The eMMC combines the flash controller, interface adapter, and flash memory arrays onto a single silicon die (Datalight 2018). With the flash controller integrated into the eMMC architecture, the storage memory interface design is simplified and the processor is freed from low-level flash memory management. Also, the interface adapter within the eMMC eliminates the need to develop interface software for different types of NAND memory and provides an easy-to-use memory solution for software programmers.

2. Performance-Power Considerations

For the new payload processor, we know that the DJI manifold utilizes the Nvidia TK1 as its main processor element. Hence, the selection of a new processor will bring changes to the performance parameters of the payload hardware and the power consumption of the hardware. In this section, the performance and power considerations for the Nvidia TK1 and Samsung Exynos 5422 are reviewed.

a. Performance

From the key specifications of the two payload hardware types summarized in Table 6, it is evident that both types have comparable parameters in terms of their hardware specifications. The most significant difference in the processor architecture is evident in the GPU processor cores, where the Nvidia Tegra K1 had 192 CUDA cores that delivered excellent parallel computing performance. In the GPU-accelerated algorithm, the sequential workload will be executed on the CPU, which is excellent for a single-threaded task, and the computational hungry portion is run in parallel over the CUDA GPU cores. The CPU-GPU architecture improves the computing and processing of complex data.

Table 6. Summary of Payload Hardware. Adapted from DJI (2014); HardKernel (2017).

	DJI Manifold	Odroid-XU4
Processor Chipset	Nvidia Tegra K1	Samsung Exynos 5422
CPU	4x ARM Cortex-A15 @ 2.3GHz 1x ARMv7 controller core	4x ARM Cortex-A15 @ 2GHz 4x ARM Cortex-A7 @ 1.2GHz big.LITTLE Processing
GPU	Kepler GK20a (192-cores) @ 950 MHz	ARM Mali-T628 MP6 @ 533 MHz
RAM	2 GBtyes DDR3L	2 GBtyes LPDDR3
Memory	eMMC 16GBtyes	eMMC 16GBtyes
Interface	2x USB 3.0 / 2x USB 2.0 1x Gigabit Ethernet	2x USB 3.0 / 1x USB 2.0 1x Gigabit Ethernet
Interface with Zenmuse X3?	Yes	No

Figures 23 and 24 show the benchmark results of the CPU and the GPU performance, respectively. The benchmark results show consistency with the hardware specifications, when the CPU performance is relatively similar for single-core and multi-core computing, and the GPU performance for the Nvidia TK1 is two times better than that of the Samsung Exynos 5422 processor.

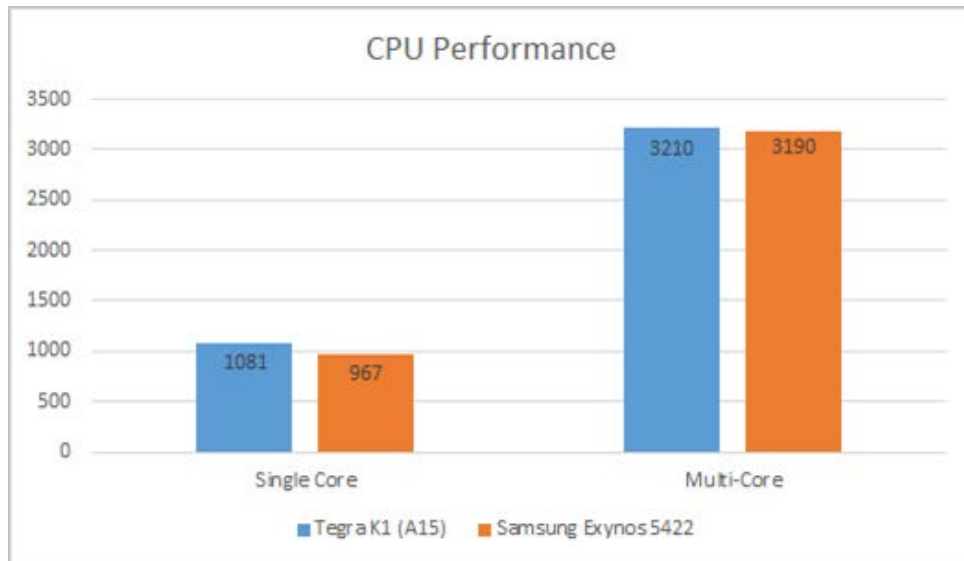


Figure 23. Plot of CPU Performance Score. Adapted from Triggs (2014).

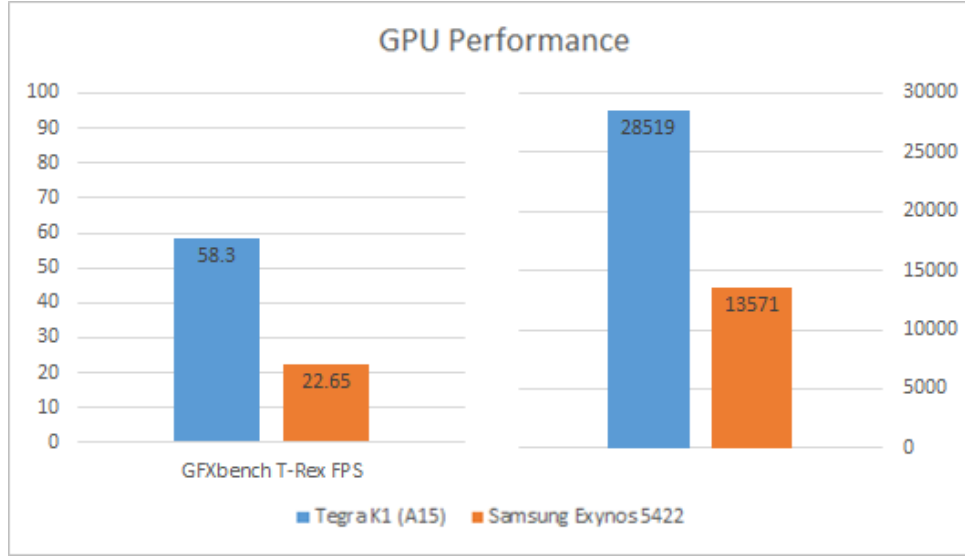


Figure 24. Plot of GPU Performance Score. Adapted from Triggs (2014).

b. Power

The DJI Manifold is designed with power management circuitry to handle the power distribution within the system. The specification of the DJI manifold required an input voltage in the range of 14 volts (V) to 26V. According to the Nvidia benchmark studies, the TK1 processor typically uses between 0.6 watts (W) to 4W during normal use, and reach a peak of 15W if the CPU, GPU, camera interface, and codec hardware are pushed to their limits (eLinux 2016).

A comparison of the results of the Nvidia benchmark studies against the power measurements for the Odroid-XU4 shows that the power consumption between the two payload processor alternatives is comparable, with the Odroid-XU4 and DJI Manifold drawing 20W and 15W at peak loading, respectively.

D. ADVANCEMENT IN CV ALGORITHMS

CV algorithms are attracting interest in both academia and real-world settings to harness the power of artificial intelligence (AI) for a wide range of applications. Among these applications, three different levels of processes are used in the identification of objects. Figure 25 illustrates the differences among image classification, object detection, and instance segmentation. In general, image classification models are used for classifying

images into a single category; object detection models are used in identifying multiple objects in a single image and providing some form of object localization in the process; and instance segmentation is the combination process of object detection and semantic segmentation that aims to group the relevant pixels of the objects together.

For most autonomous systems (e.g., self-driving cars and transportation surveillance), object detection and object tracking are the two major components required in the CV algorithm to enable the required level of autonomy in these applications (Zhu et al. 2018). Object detection identifies the target from the imagery data of the payload sensor, and object tracking supports the update of target location as it moves through the imagery data. The processed data output from the object tracker facilitates the navigation of the systems to accomplish the desired actions set.

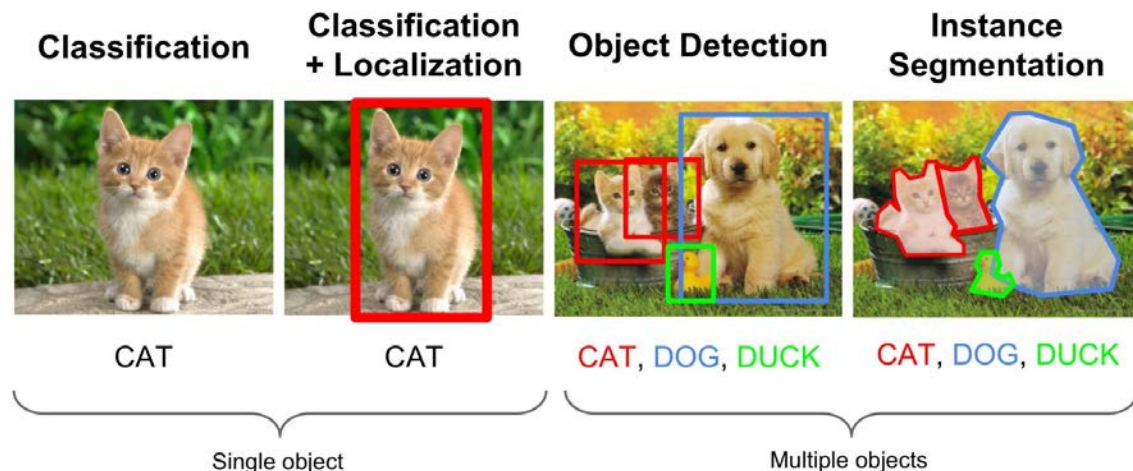


Figure 25. Comparison of Image Classification, Object Detection, and Instance Segmentation. Source: Ouaknine (2018).

Although there have been various approaches to handle object detection and tracking, the real game-changer has been through the use of deep machine learning for CV algorithm application. In machine learning, convolutional neural networks (ConvNets or CNNs) classification is the current state-of-art technique for efficient processing of the imagery data from the sensor.

1. Convolutional Neural Networks

CNN is a variant of artificial neural networks, having the ability to perform image recognition and object detection tasks by generating identification characteristics from a training dataset. The architecture of regular neural networks is explained in CS231n course materials, instructed by Dr. Fei-Fei Li, a renowned Stanford University professor working in the area of computer vision and Chief Scientist of Google Cloud AI and machine learning, as:

Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores. (Li et al. 2017)

Regular neural networks, however, do not perform well for larger images. Considering images from the CIFAR-10¹ dataset with a size of 32 x 32 x 3 (32 wide, 32 high, 3color – RGB), a single fully connected neuron of the first hidden layer within the regular neural network would have $32 \times 32 \times 3 = 3,072$ weights (Li et al. 2017). If a larger image is desired (i.e., a size of 200 x 200 x 3), the neuron will have $200 \times 200 \times 3 = 120,000$ weights. As such, it is evident that weights add up quickly, and the huge weight parameters would lead to overfitting and ultimately to inefficiency in processing (Li et al. 2017).

Therefore, researchers included a convolution process layer in addition to the regular neural network to resolve the issue related to larger image, where it helps to preserve the spatial relationship between the imagery data by learning features using smaller squares of image data (Karn 2016). In CNNs, the convolution process layers are organized in three dimensions: width, height, and depth. Additionally, the upper layer neurons do not connect to all the lower layer neurons, specifically to a small region of the lower layer. The output layer of the CNNs will be reduced to a single vector of probabilistic

¹ CIFAR-10 is a widely used dataset used to train and test machine learning and computer vision algorithms. The dataset consists of 60,000 32 x 32 color images in 10 different classes, with 6,000 images per class (Krizhevsky 2010).

scores, structured along the depth dimension (Li et al. 2017). Figure 26 illustrates the architecture of a regular neural network (left) versus a convolutional neural network (right).

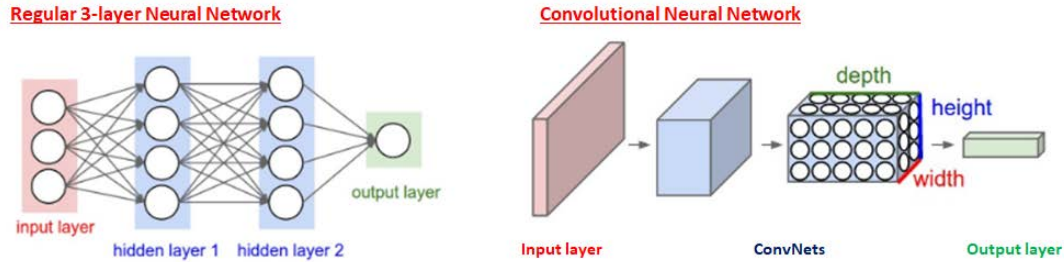


Figure 26. Illustration of the Regular Neural Network versus Convolution Neural Network. Adapted from Li (2017).

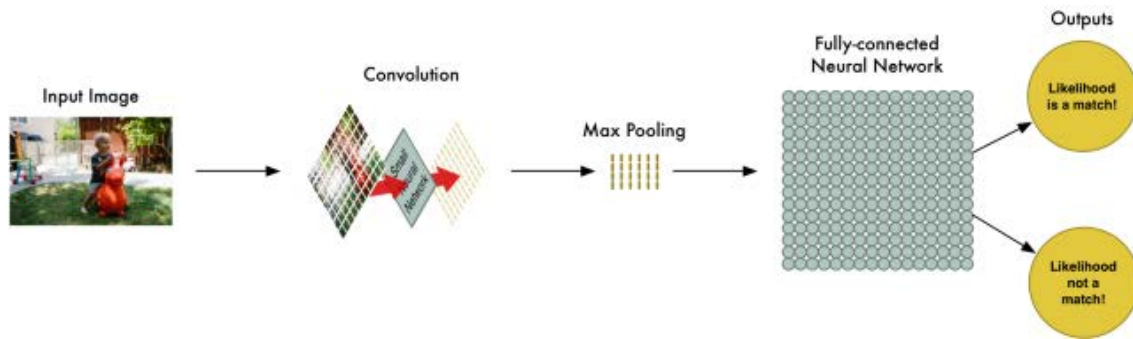


Figure 27. Pipeline of a Convolutional Neural Network. Source: Geitgey (2016).

Figure 27 illustrates the five-step pipeline overview of a CNN. It can be seen that the main building block of the CNN is the convolution layer. In mathematical terms, convolution refers to the integral of the amount of overlap between two functions to produce a third function (Weisstein 2018). Hence, the convolution process in a CNN will be performed on the input image with the use of a moving convolution filter, which slides over the image to produce a feature map. At each location on the image, the matrix multiplication is executed, and the results are summed to create a feature map.

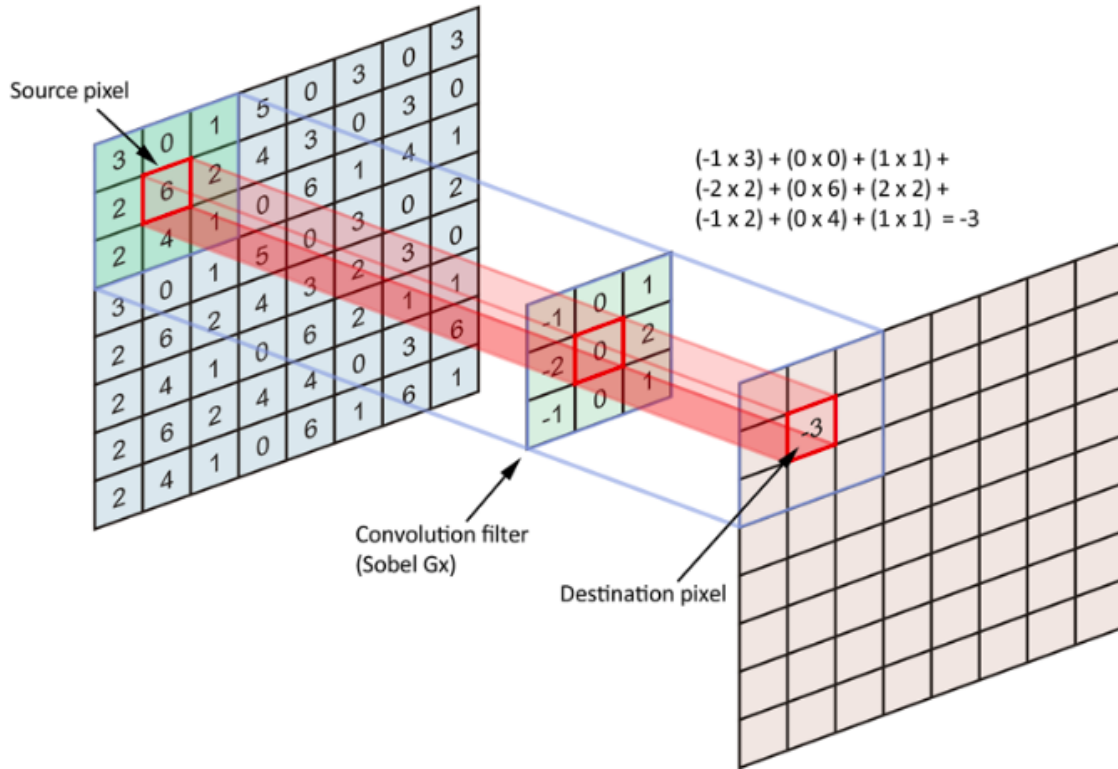


Figure 28. Convolution Process. Source: Dertat (2017).

This convolution process is repeated on the input image with different filters to generate different feature maps. These feature maps are passed through the rectified linear unit (ReLU) activation function to make them non-linear. The ReLU replaces the negative pixel value found in the feature map to zero. Subsequently, these rectified feature maps are stacked together to form the final output of the convolution layer. Figure 29 shows examples of the visual representation of the convolution process layer.

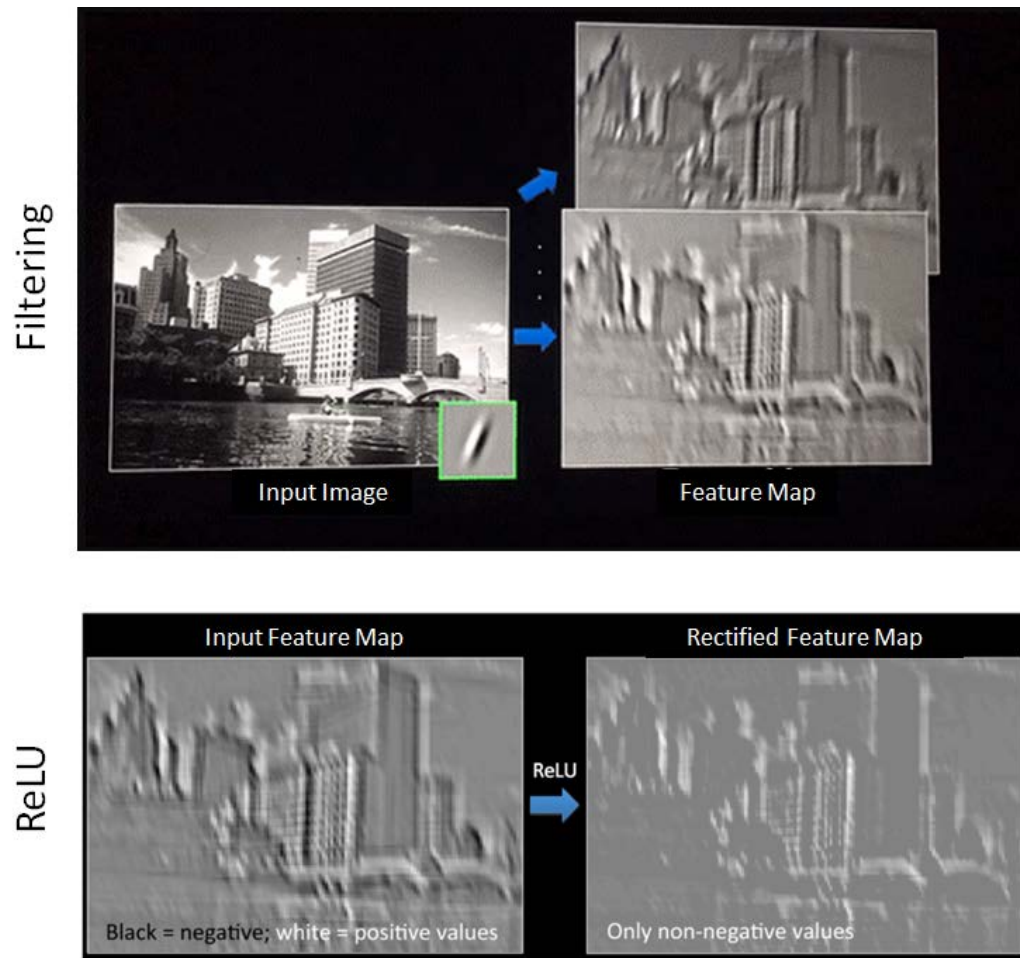


Figure 29. Visual Representation of Convolution Process Layer. Source: Fergus (2015).

The outputs from the convolution layer are then passed through the pooling layer. The function of this pooling layer is to reduce the dimensionality; hence, it is intended to achieve a reduction in the number of parameters, which shortens the training time and mitigates overfitting of the algorithm (Dertat 2017). CNNs typically use max pooling, which returns the max value in the pooling window. Hence, the pooling layer is able to down sample each feature map, leading to a reduction in height and width while keeping the depth. Figure 30 illustrates the pooling process, whereby the results of max pooling using a 2-by-2 window are passed over the feature map. The max pooling process decreases the feature map size without losing the significant information.

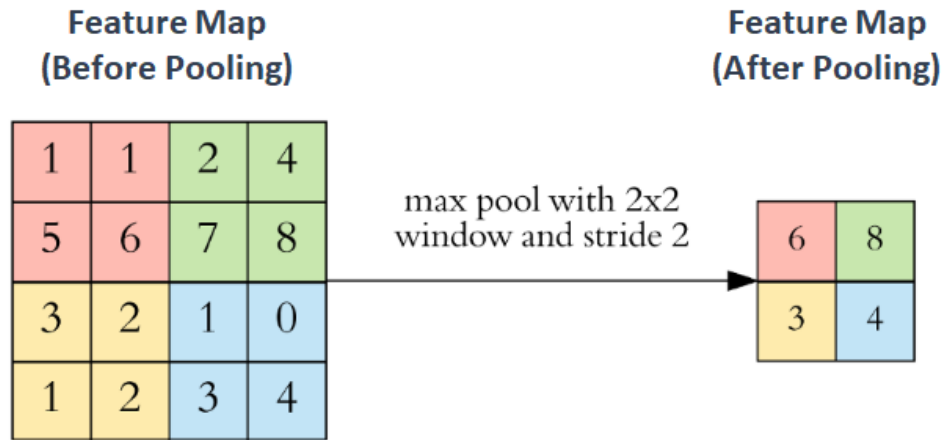


Figure 30. Pooling Process. Adapted from Dertat (2017).

After the convolution and pooling process layer, the fully connected neural network is used for classification process. Similar to the working principle of the regular neural network, the neurons are trained to classify and tell the probability of the object within the input image (Li et al. 2017). Figure 31 shows an example of a typical CNN architecture.

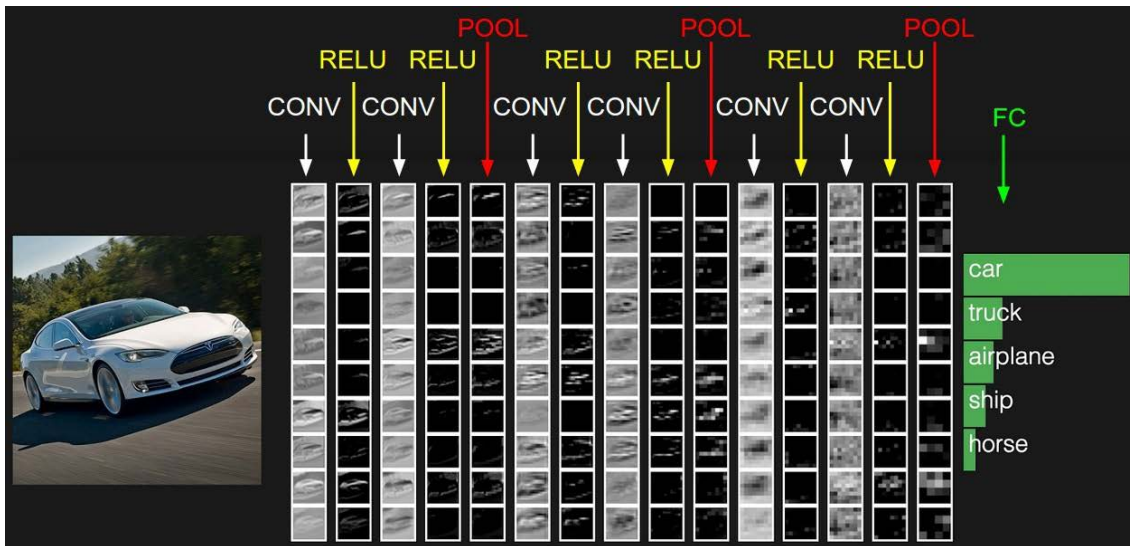


Figure 31. Example of CNN Architecture. Source: Li et al. (2017).

2. Object Detection Algorithm

The CNN provides the foundation for image classification. Nevertheless, in order to perform object detection, the algorithm should have the ability to locate the object of interest and define a bounding box containing it within the image. Furthermore, the ability to define many bounding boxes representing the different objects within the image is also required. Considering the classification of “duck” in Figure 32, the standard CNN will have issues processing and performing image classification, due to the multiple occurrences of the object (Gandhi 2017). In the native approach, the image is broken up into multiple regions, and the CNN will then perform the image classification task on each specific region. However, the objects of interest might have different spatial locations and aspect ratios within the image. Thus, a multiple number of regions with overlap will be defined, making the CNN computationally intensive and, therefore, less efficient. For this reason, algorithms like the Region-based Convolutional Neural Network (R-CNN) and the You Only Look Once (YOLO) algorithm are designed to search rapidly for these occurrences.

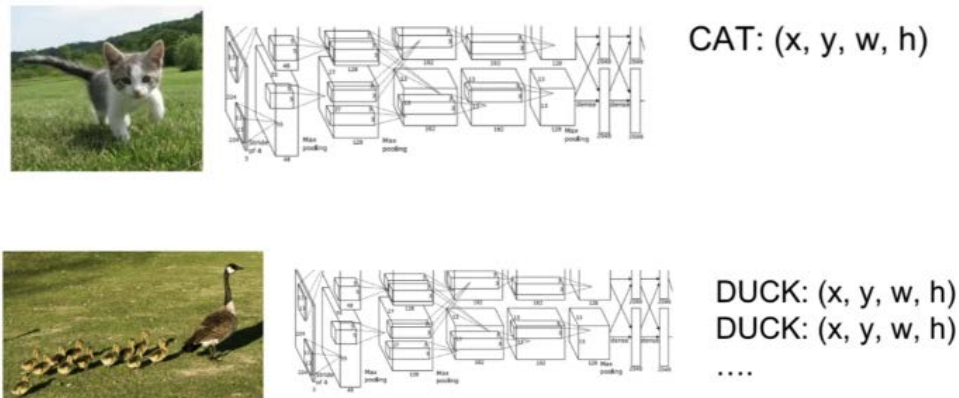


Figure 32. Representation of CNN Architecture. Source: Gandhi (2017).

a. *Region-Based Convolutional Neural Network (R-CNN)*

The R-CNN model proposed by Ross Girshick et al. (2014) was one of the earliest works to overcome the problem of multiple occurrences of the same object within an image. The R-CNN combines the selective search method to extract 2,000 regions and the

CNN to localize and classify the objects in the defined region. The selective search method proposed by Jasper Uijlings et al. (2013) begins by defining a small region in the image, before merging them according to the color spaces and similarity metrics within the hierarchical grouping. The output of this selective search method is a variety of region proposals containing the objects. Figure 33 illustrates the visualization of the selective search method, whereby the top shows the region proposals and bottom shows the results of object segmentation and localization.



Figure 33. Visualization of the Selective Search Method. Source: Uijlings et al. (2013).

Each of the region proposals is resized to the input requirement of the CNN, producing a 4,096-dimensional features vector. The CNN acting as the feature extractor passes the extracted feature to a support vector machine (SVM)² classifier to classify the

² According to Rohith Gandhi (2017), the objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N—the number of features) that distinctly classifies the data points.

presence and predict the probabilities of objects within the region proposal (Girshick et al. 2014). In addition, the algorithm predicts four values through a linear regressor to adapt the shapes and to increase the precision of the bounding box. This in turn reduces localization errors of the objects. Figure 34 shows the architecture of the R-CNN. Despite its success at more efficient image processing, the R-CNN still takes up significant computing resources to train the network, and real-time system performance requirements cannot be met.

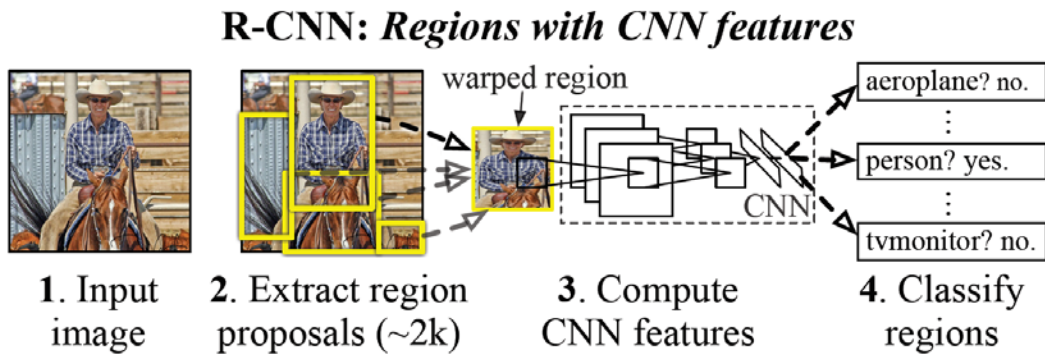


Figure 34. Architecture of R-CNN. Source: Girshick et al. (2014).

b. Fast R-CNN

The same author, Girshick, of the R-CNN paper improves the algorithm in his follow-on research to reduce the time consumption on processing a high number of models and analyzing all region proposals (Girshick 2015). Since the objective is to develop a faster object detection algorithm, the Fast R-CNN follows a similar approach as the R-CNN algorithm. Instead of extracting 2,000 region proposals from an input image, the image is passed through the CNNs to generate a convolutional feature map. Regions of Interest (RoI) are identified using the selective search method applied on the feature maps. Through the use of the RoI pooling layer, the size of the feature maps is reduced, and each RoI layer with fixed height is fed into the fully-connected neural network layer. A feature vector is created through this RoI pooling layer, which is used for prediction of the object class of the proposed region and adapting bounding box localizations (offset values for the

bounding box). Figure 35 shows the architecture of the Fast R-CNN. Although the Fast R-CNN is significantly faster in training and testing sessions than the R-CNN, the performance of the Fast R-CNN is affected by the generation of RoIs on the selective search method (Gandhi 2017). Therefore, generating RoIs / region proposals becomes a challenge to improve the algorithm performance.

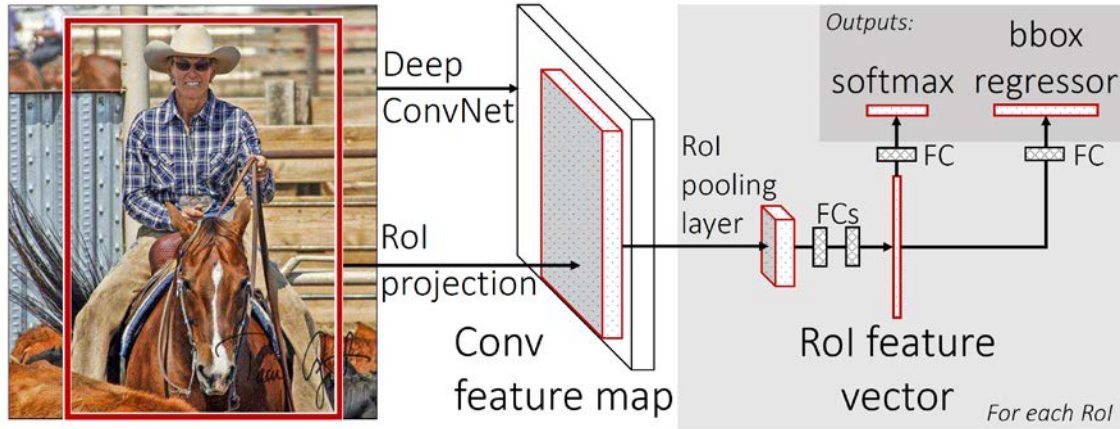


Figure 35. Architecture of Fast R-CNN. Source: Girshick (2015).

c. *Faster R-CNN*

In the R-CNN and Fast R-CNN algorithms just discussed, the selective search method was used to determine the region proposals. As noted in the discussion, the selective search method was inefficient and computationally expensive, which affected the overall performance of the algorithms. Shaoqing Ren et al. (2016) introduced the Region Proposal Networks (RPN) to replace the selective search method, with the aim to generate directly the region proposals within the image and bounding boxes around the detected objects. The Faster R-CNN, as illustrated in Figure 36, takes on the previous work of the Fast R-CNN and combines this with the RPN algorithm. Similar to Fast R-CNN, the algorithm produces convolutional feature maps based on the entire image. Next, the RPN generates the features vector connected to two fully-connected neural network layers, whereby one is for the box regression (coordinates of the bounding box, its height and width) and one is for box classification (probabilistic score to determine if an object is

within the box). The RPN is trained to generate high quality region proposals; hence, its usage in Faster R-CNN accelerates the computation process and improves the algorithm performance in object detection as compared to Fast R-CNN.

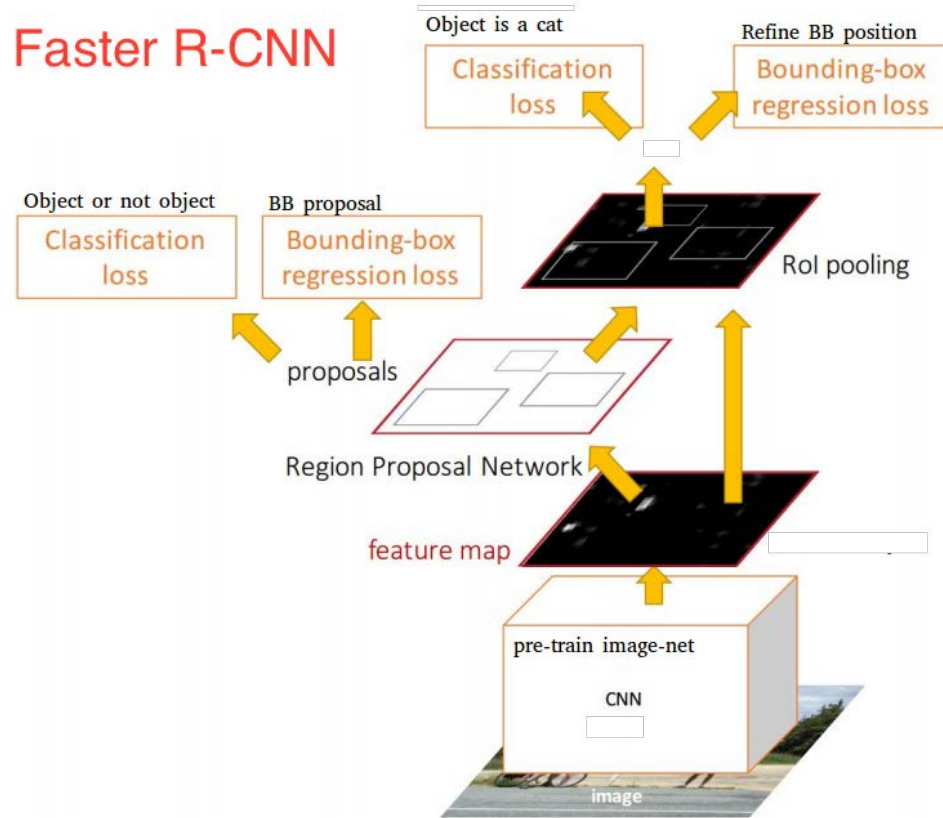


Figure 36. Architecture of Faster R-CNN. Source: Xu (2017).

d. *You Only Look Once (YOLO) Algorithm*

All of the previous CNN algorithms are designed to use regions proposal to localize objects within the image. By contrast, the You Only Look Once (YOLO) algorithm, developed by Joseph Redmon et al. (2016), predicts bounding boxes and object class probabilities using a single convolutional network. The architecture of the YOLO algorithm, as illustrated in Figure 37, shows the process flow of the algorithm. The algorithm takes in the whole input image and splits it into an $S \times S$ grid. Each grid cell predicts B bounding boxes with C class probabilities and an offset value for the bounding

box. The bounding boxes with probabilistic scores above the threshold value are processed, and object localization within the image is performed. In the final layer, the algorithm outputs a $S \times S \times (C + B \times 5)$ tensor, which represents the predictions for each grid cell (Redmon et al. 2016).

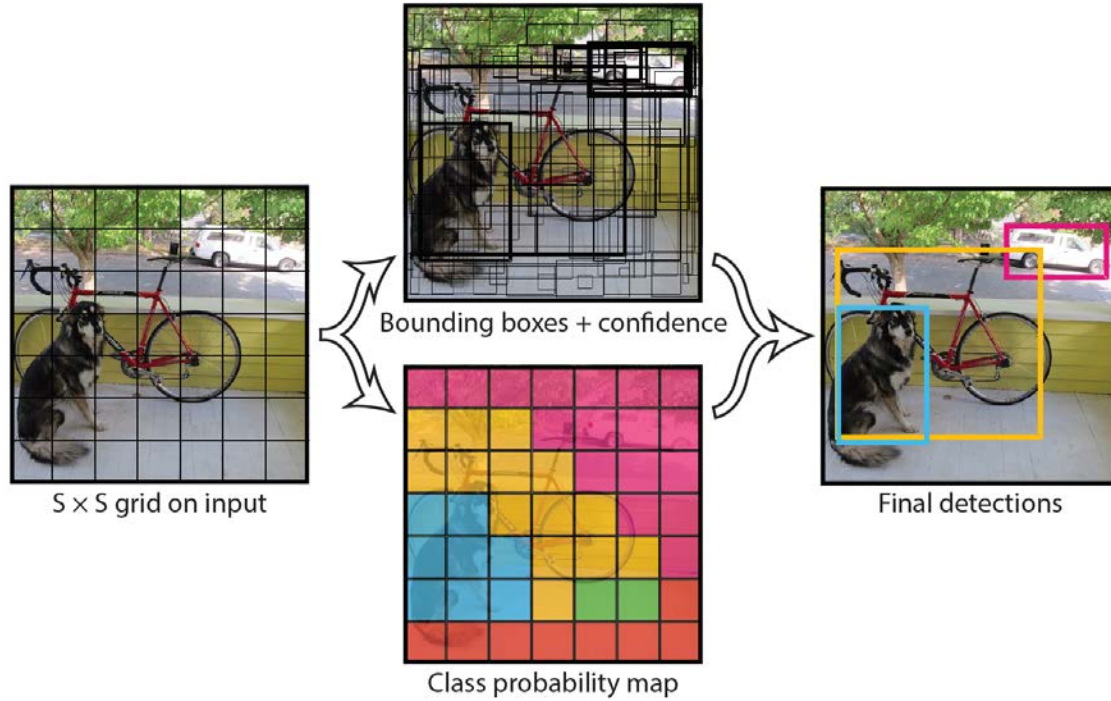


Figure 37. Architecture of YOLO Algorithm. Source: Redmon et al. (2016).

Furthermore, Redmon et al. (2016) highlighted that one potential drawback with the YOLO algorithm is the strong spatial constraint on bounding box prediction. Therefore, this limits the algorithm's ability to predict small objects appearing in groups (e.g., flocks of birds). In addition, although the YOLO algorithm is orders of magnitude faster, it might not be as accurate as the Faster R-CNN (Li et al. 2017). Similar to the R-CNN, the YOLO algorithm was fine-tuned and improved in subsequent ongoing research. As of this writing, the latest YOLOv3 had worked toward faster and more accurate object detection.

3. Performance

The recent excitement about deep machine learning has attracted different neural network related research. Subject matter experts in this area have been improving their algorithms for higher precision in object detection prediction and for achieving near real-time performance. Beside the R-CNN and YOLO algorithms, researchers have developed other object detection algorithms such as the single shot detector (SSD), region-based fully convolutional network (R-FCN), and the neural architecture search net (NASNet). Hence, the computer vision researchers use a standard dataset as a challenge to evaluate the performance of their algorithms. The commonly used metric in object detection challenges is mean Average Precision (mAP), providing the mean value of the average precision generated from all the class objects within the dataset challenge. From the performance results shown in Table 7, it is evident that the YOLO algorithm achieves a better frame per second (FPS) performance, which is critical for systems that need real-time object detection capabilities. Nevertheless, the better FPS rates do come with a cost to the mAP. Using the PASCAL VOC dataset, YOLO has a mAP score of 63.7 percent versus the Faster R-CNN, which has a mAP score of 78.8 percent. Hence, it will be important for designers to consider the system trade-off in deciding on the use of these generic algorithms for their applications.

Table 7. Performance of Algorithm Model. Adapted from Ouaknine (2017).

Algorithm Model	Mean Average Precision		Estimated FPS	Real Time Speed
	PASCAL VOC 2007	PASCAL VOC 2012		
Fast R-CNN	70.0%	68.4%	~ 0.5	No
Faster R-CNN	78.8%	75.9%	~ 5	No
YOLO	63.7%	57.9%	~21	Yes

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ADVANCED SYSTEM DESIGN

This chapter reviews the architecture, including the hardware and software configurations of the advanced system.

A. SYSTEM ARCHITECTURE

Based on the system considerations presented in Chapter III, the improvements to the baseline system are made. Figure 38 shows an overview of the advanced M100 system. As compared to Figure 1, it can be seen that the main refinement is on the inclusion of the DJI Manifold (processor element) to support the CV algorithm (application), with the video data stream from the DJI Zenmuse X3 (sensors payload).

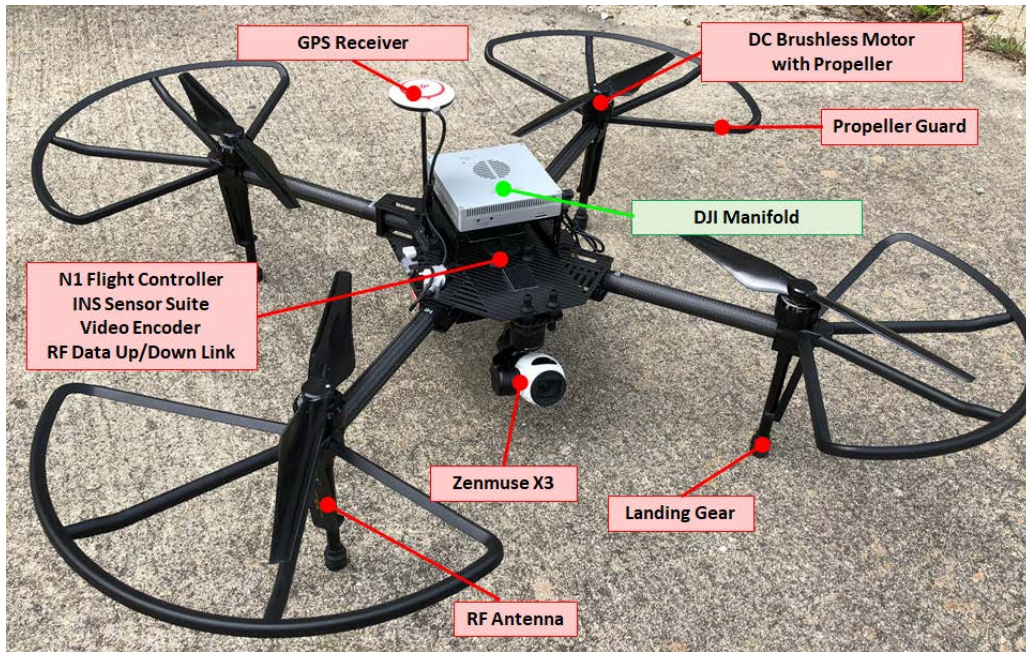


Figure 38. Overview of Advanced System

In terms of weight and dimensions, the proposed changes on the system have minimal effect. Hence, the flight performance of the new system setup will not be compromised. Based on the measurement of the system, the weight is 2.75 kilograms (Kg), and it has dimensions of 22 in (L) * 20 in (W) * 14.2 in (H).

B. HARDWARE CONFIGURATION

The hardware configuration of the M100 system consists of the base unit and the payload electronics. As discussed in Chapter II, the base unit employs only the fewest and most essential hardware items necessary for the M100 to operate in a flight environment. This includes modules such as N1 flight controllers, INS sensor suite, GPS receiver, ESC, DC brushless motors, propellers, battery pack, and gimbal camera. The payload electronic package is the experimental hardware installed as an additional feature to support autonomous system behavior research. For the new system's development, the main changes are the installation of propeller guards and the DJI manifold system. The propeller guards are additional safety considerations for future test and evaluation of semi/fully autonomous flight. The DJI manifold system replaces the Odroid-XU4 hardware and enables us to utilize the onboard Zenmuse X3 camera. Figure 39 illustrates the interconnection among the payload electronics package. The payload electronics package is defined as the modules required by the system to achieve autonomy, and it consists of N1 flight controller, Zenmuse X3 gimbal mount, and the DJI manifold processor.

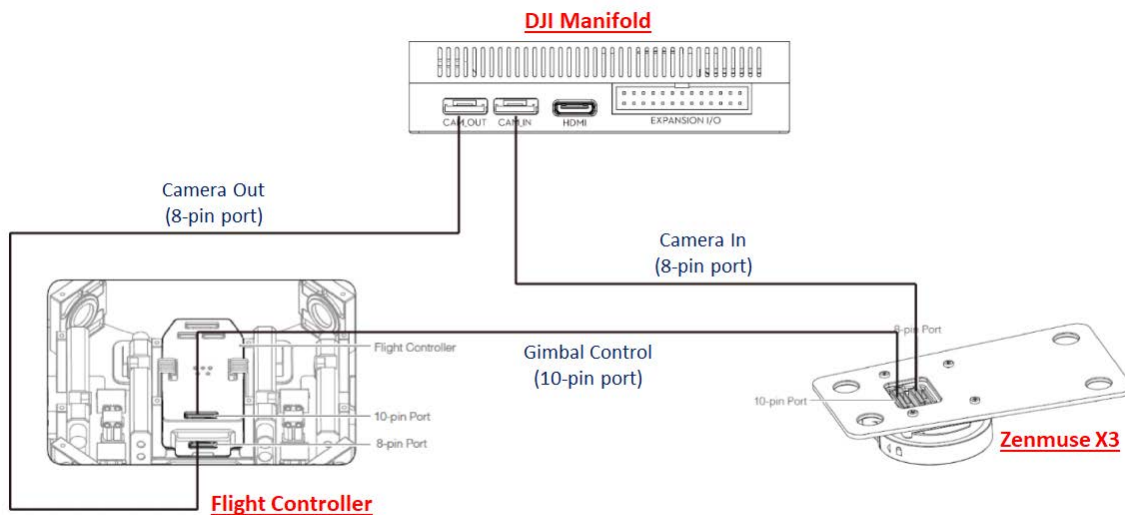


Figure 39. Interconnection for Flight Controller, Zenmuse X3 Gimbal Mount, and DJI Manifold

The entire imagery data flow commences with the payload sensor (Zenmuse X3), which is responsible for capturing the flight scene. The sensor will send this imagery data via a DJI proprietary interface on an eight-pin port into the DJI manifold CAM_IN port. Through this CAM_IN port, a software application will be executed to decode and convert the raw data into the required format for processing (RGB video) prior to publishing the video stream as an ROS node for either required CV algorithm processing or preparing the data for RF data downlink. The algorithm application can subscribe directly to the ROS node to process the imagery data.

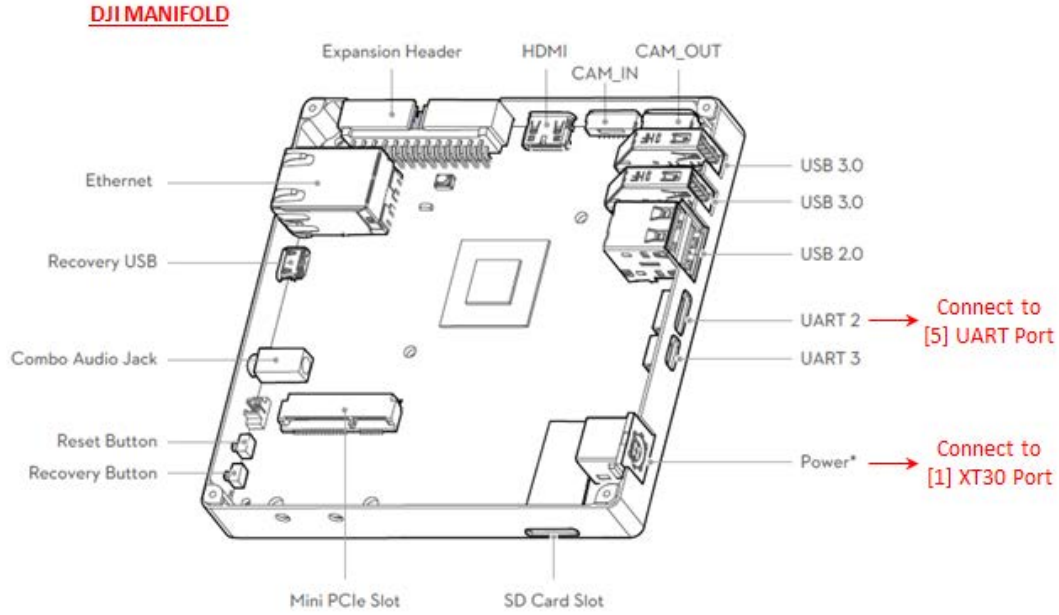
Once the CV algorithm detects and tracks the desired object, the M100 system can be commanded to perform specific flight maneuver actions. Examples of flight maneuver actions could be the following: (1) Evading the object if it is too close – SWARM UAV concept; (2) Moving toward the object – Homing of the weaponized UAV onto target; and (3) Monitoring and following the object – Surveillance. The DJI manifold, however, has a resource limitation for computationally intensive algorithms. As such, the DJI Manifold will be able to convert the video stream to the Real-Time Streaming Protocol (RTSP), which essentially is an internet protocol (IP) video. This provides opportunities for an external hardware module to be connected for further experimental testing with the system video stream.

For the RF downlink data to ground control station, the data will be forwarded to the N1 flight controllers via the CAM_OUT port on the DJI Manifold. This video data stream will be compressed within the N1 flight controller and sent by RF downlink to the ground control station for piloting or mission tasking purposes. The ground operator can control the viewing angle of the gimbal camera (Zenmuse X3) using his or her remote controller. Together with the control actions from ground control, the data are sent by RF uplink back to the M100 platform. This data is interpreted by the N1 controller to perform the required changes in motor speed or the gimbal motor. This establishes a man-in-the-loop function between human and machine. The RF controller at the ground control station remains as the single input/output source to the UAV system.

Figure 40 illustrates the signal and power connection required for the payload electronics package, namely the Zenmuse X3, DJI Manifold, and N1 flight controller. The

signal and power for the DJI manifold, a key element for CV applications, is described here:

- CAM_IN: Receives raw video data from the payload sensor. The interface protocol is undisclosed by DJI. However, the manifold and its associated library will be able to read in the raw video data. Its connection is through an 8-pin cable.
- CAM_OUT: Transmits the decoded video in RGB format to the N1 flight controller. Similar to CAM_IN, the interface protocol is undisclosed by DJI, but the manifold has been designed to perform the required functions. Its connection is through an 8-pin cable.
- UART 2: Transmits/receives flight data from the N1 flight controller. The DJI SDK must be set up in order to perform communication linkup between the devices. In order to receive data from the N1 flight controller, the interface protocol must be able to subscribe to the ROS topic. For the transmission of data, the manifold uses the published ROS node function. Its connection is through a 6-pin cable.
- Power: Provides the main power input to the DJI manifold. The output voltage level from the XT30 (source) is 20V ~ 26V, while the DJI manifold accepts an input voltage of 14V ~ 26V. Its connection is through a XT60 power cable.
- Ethernet: Is an expansion port for additional experimental hardware or data transfer. This port can be used to broadcast IP video from the Zenmuse X3 or to perform secure login via another Ubuntu device. Its connection is through a CAT5 or better Ethernet cable.



DJI M100 SYSTEM (Reserved Ports)

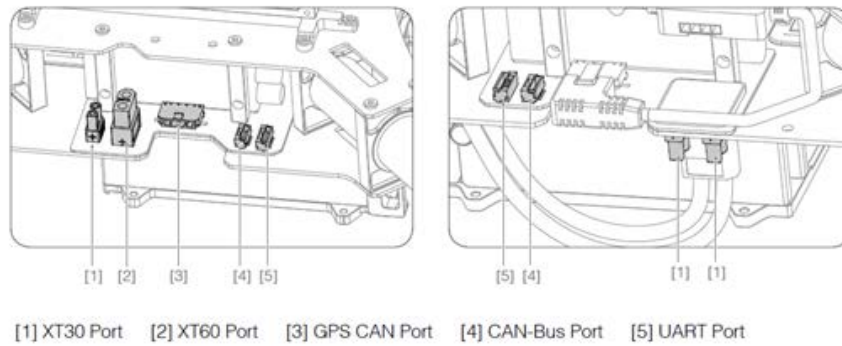


Figure 40. Schematic Overview of DJI Manifold Interface (top) and M100 Reserved Ports (bottom). Adapted from DJI (2017).

C. SOFTWARE CONFIGURATION

Similar to the existing setup, the software configuration for the refined M100 system is broken down into software modules residing within the base unit, the N1 flight controller, and the payload processor, the DJI Manifold. The software architecture of the M100 system is illustrated in Figure 41. In the earlier section on hardware configuration, it is discussed that the DJI Manifold is tasked to execute imagery data crunching and pre-processing of the raw data. Given that the Zenmuse X3 data uses a propriety interface,

decoding of this raw imagery data stream requires a specific camera driver library in the manifold. The decoded data will be sent to the video compression algorithm that resides within the DJI N1 flight controller for the RF video downlink to the ground control station. The N1 flight controller software, however, is protected from modification by the user. Appendix A presents the setup procedures for the software libraries and tools to support algorithm usage on the DJI Manifold.

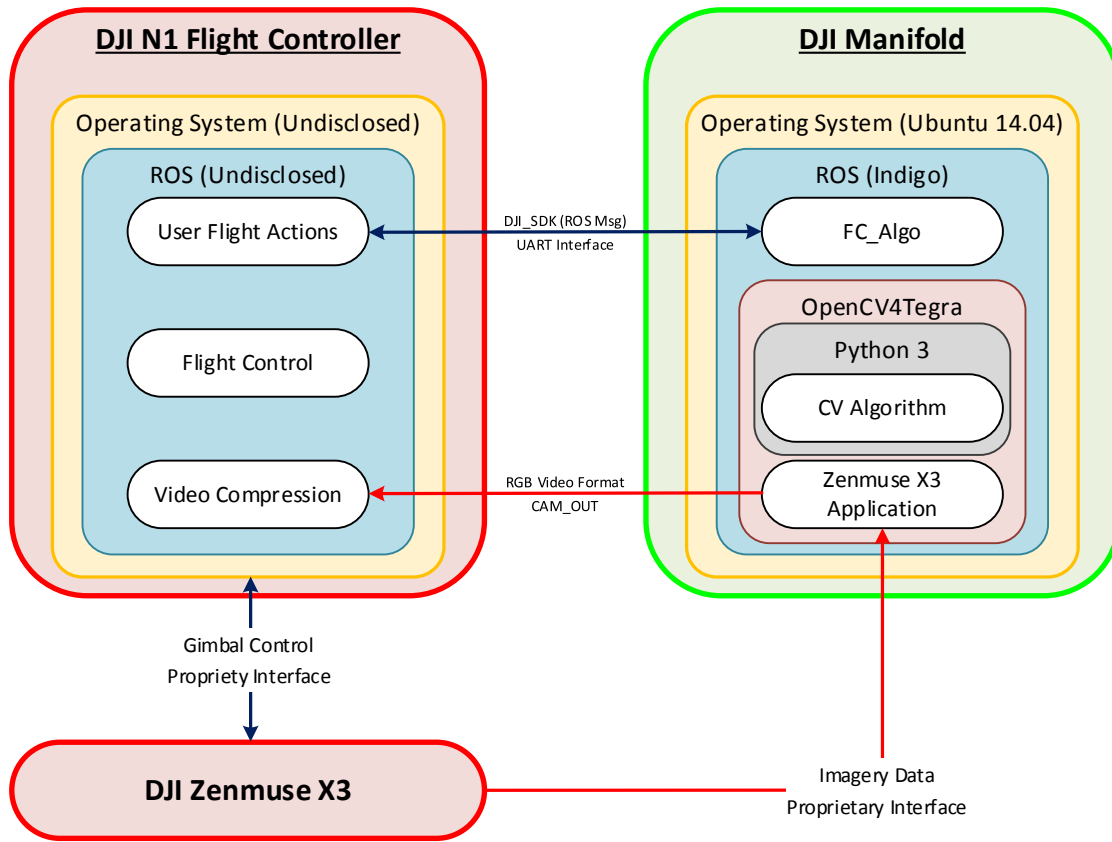


Figure 41. Software Architecture for Refined DJI Matrice 100

The CV algorithm is another beneficiary of the decoded imagery data, whereby different algorithms can be executed according to the system mission plan and its intended application. As an example of object detection and tracking, a CNN type of algorithm such as YOLO can be executed to perform detection on certain object categories. The output from the CV algorithm is combined with the results from other tracking applications and

sensor readings for the flight control application. This flight control process is taken care of by the FC_Algo residing on the DJI manifold system.

The FC_Algo is responsible for computing and generating the required flight control actions. For the M100 system, however, the N1 flight controller controls the actual execution of flight control in terms of changing the motor speed. The DJI N1 flight controller does not, however, allow for user modification of the software applications. For this reason, the M100's flight controller defined the specific ROS commands to control the flight dynamics and maneuvering of the system. The user-developed FC_Algo will communicate to the user flight action application via the UART interface pipeline.

The inputs of the user flight actions and the system sensor suite influences the flight control application. Since users are not allowed to modify the software layer on the N1 flight controller, DJI provides regular interval firmware updates to support additional stability for system performance. Hence, it is necessary to ensure the firmware version supports the correct DJI SDK package.

1. Software Integration

The DJI system software consists of the software package supporting the basic operating functions of the M100 system. The main focus of the software package is the DJI flight controller software, which was pre-loaded into the system. This software package, also known as the firmware, is maintained by DJI, and users are notified about updates through the DJI Go application. In order to achieve the full functional capabilities of the system, the firmware must handle the communication transactions from the aerial vehicles to the ground control station and the necessary debugging. A description of how to update the firmware can be found on the DJI website. Table 8 documents the summary of the DJI System software utilized to support the thesis research.

Table 8. Summary of DJI System Software

Description	Version
M100 Flight Controller	1.3.1.00+
Zenmuse X3	1.8.1.00
DJI Remote Control	1.7.80
DJI Go (IOS Application)	3.1.42
DJI Assistant (PC Desktop Application)	1.2.4

(1) DJI Software Development Kit

The DJI SDK is an open source software library provided by DJI for enabling direct communication with the M100 system and N1 flight controller over a UART interface. Through the SDK, a fully featured ROS wrapper compatible with ROS standards gives read/write access to aircraft telemetry, flight control dynamics, and other aircraft functions. When the system is connected to the Windows PC-based DJI Assistant, the DJI SDK provides a toolkit for the system simulator and situation visualization. Figure 42 illustrates the hierarchy of the DJI SDK software package. The SDK package consists of ten classes of functions to support various types of user-defined FC_Algo application development.

Assuming there is interest in developing specific flight control, the DJI SDK enables users to define low level flight dynamics and control using altitude, system velocity, and position commands. In addition, the system can be programmed to execute system take-off, system landing, and return to home point.³ With the availability of these basic flight control functions, users can develop their required algorithms for different missions and applications.

³ Home point refers to the programmed GPS coordinate location of the system prior to take-off.

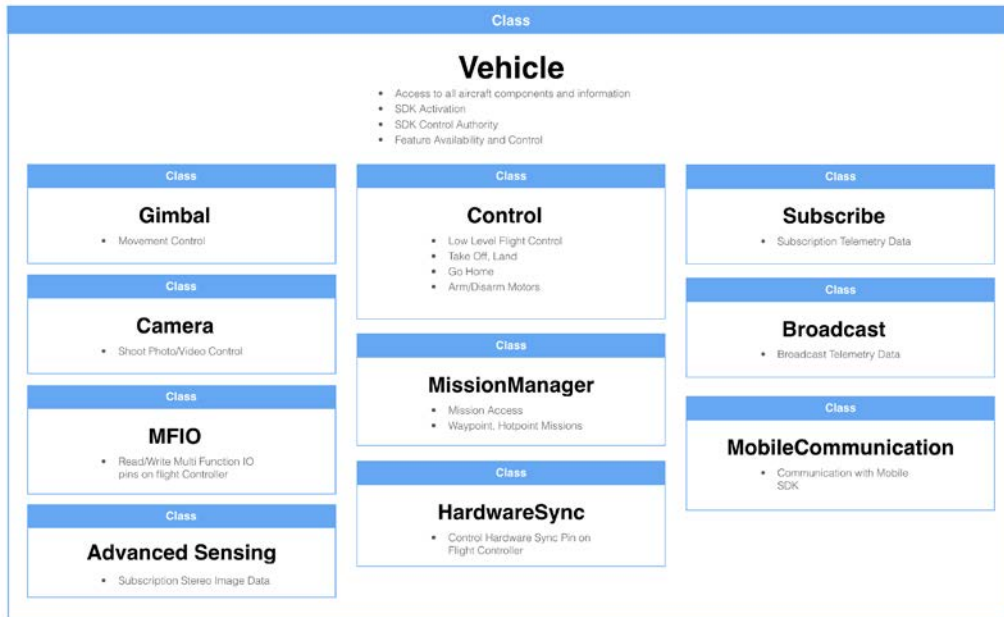


Figure 42. Hierarchy of DJI SDK Application. Source: DJI (2017).

Appendix B documents the procedure for the DJI SDK setup. In this research, the DJI SDK software package is installed for the Manifold hardware, and the SDK is activated (licensed) to utilize the flight control algorithm application. One can follow a similar setup procedure for any other ROS-enabled payload processor. In order to execute the DJI SDK core, the commands shown in Table 9 are utilized to launch the ROS server:

Table 9. Commands for Running the DJI SDK Server

<code>cd ~/DJI_SDK/catkin_ws/</code>	//Browse to Directory
<code>source devel/setup.bash</code>	//Load Setup Configuration
<code>roslaunch dji_sdk sdk_manifold.launch</code>	//Launch DJI Server

If the DJI SDK ROS server is executed correctly, a system status of “STATUS activateCallback, line 911: Activated successfully” is displayed on the system terminal. Figure 43 presents the screen capture of the successful execution of the DJI SDK ROS

server on the Manifold. In other words, the Manifold has completed the communication handshake with the DJI N1 flight controller and is ready for FC_Algo application.

```
/home/ubuntu/DJI_SDK/catkin_ws/src/Onboard-SDK-ROS-3.2/dji_sdk/launch/sdk_manifold
ubuntu@tegra-ubuntu:~/DJI_SDK/catkin_ws$ source devel/setup.bash
ubuntu@tegra-ubuntu:~/DJI_SDK/catkin_ws$ roslaunch dji_sdk sdk_manifold.launch
... logging to /home/ubuntu/.ros/log/d4726a52-8ed3-11e8-b7c1-60601fa92479/roslau
nch-tegra-ubuntu-13946.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://tegra-ubuntu:47626/

SUMMARY
=====

PARAMETERS
* /dji_sdk/app_bundle_id: Not required in t...
* /dji_sdk/app_id: 1055627
* /dji_sdk/app_version: 1
* /dji_sdk/baud_rate: 115200
* /dji_sdk/enc_key: a07e74ad43621a75b...
* /dji_sdk/groundstation_enable: 1
* /dji_sdk/serial_name: /dev/ttyTHS1
* /roscdistro: indigo
* /rosversion: 1.11.21

NODES
/
  dji_sdk (dji_sdk/dji_sdk_node)

auto-starting new master
process[master]: started with pid [13957]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to d4726a52-8ed3-11e8-b7c1-60601fa92479
process[rosout-1]: started with pid [13970]
started core service [/rosout]
process[dji_sdk-2]: started with pid [13974]
=====
app id   : 1055627
app key  : a07e74ad43621a75b65ade05b82b6bd4bc406aa8778b946ec90vbd10ffab7e7
=====
--- Connection Info ---
Serial port: /dev/ttyTHS1
Baudrate: 115200
-----
STATUS init,line 62: Open serial device /dev/ttyTHS1 with baudrate 115200...
[ INFO] [1532390109.998088311]: Succeed to create thread for readPoll
STATUS parseDroneVersionInfo,line 428: Device Serial No. = 041DDC0908
STATUS parseDroneVersionInfo,line 431: Hardware = M100
STATUS parseDroneVersionInfo,line 433: Firmware = 3.1.10.0
STATUS parseDroneVersionInfo,line 437: Version CRC = 0xA6453AAC
=====
Hardware : M100
Firmware : 0x03010A00
=====
Broadcast call back received
STATUS activateCallback,line 911: Activated successfully
█
```

Figure 43. Successful Execution of the DJI SDK ROS Server

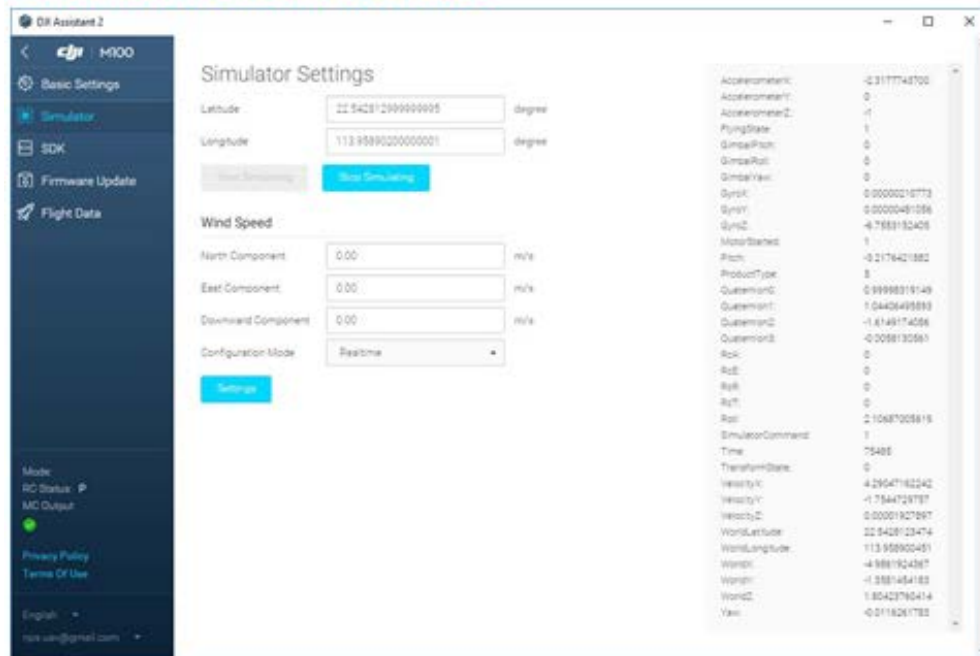
Alternatively, the DJI SDK demo client software can initialize simulation testing of the M100 system. In order to execute the DJI SDK demo client, the commands shown in Table 10 are utilized:

Table 10. Commands for Running the DJI Demo Client

<code>cd ~/DJI_SDK/catkin_ws/</code>	//Browse to Directory
<code>source devel/setup.bash</code>	//Load Setup Configuration
<code>roslaunch dji_sdk_demo dji_sdk_client.launch</code>	//Launch DJI Client

Once the demo client is launched, a variety of flight control functions is made available for simulation testing of the M100 system. Figure 44 presents the simulator environment for the DJI M100 system. By using the simulator, the user can test the FC_Algo application prior to the actual flight test, which can reduce the risk of algorithm error in flight.

DJI Assistant 2 Simulator Settings



Simulation Environment

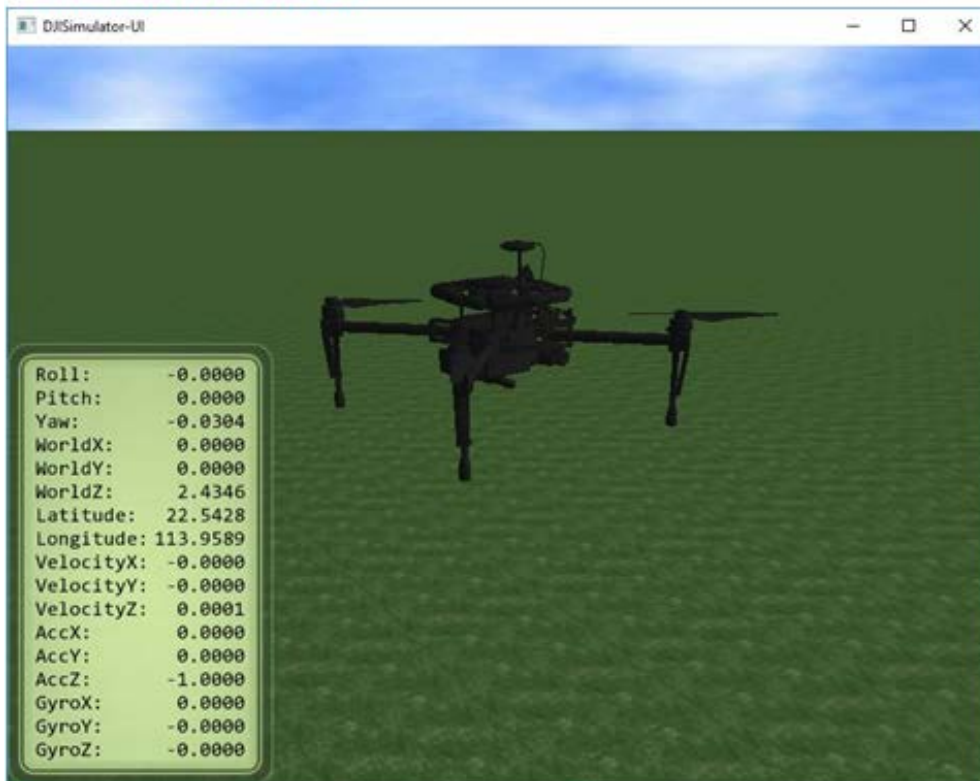


Figure 44. DJI M100 Simulator Environment

2. User Applications

The FC Algorithm is programmed using the DJI SDK ROS wrapper, which is enabled through the execution of the DJI SDK ROS server. Assuming there is a need for the flight control application to move the system to a specific GPS coordinate, a sampling code as illustrated in Figure 45 enables the objective. By initializing the ROS publishers, the ctrlPosYawPub publishes command for the x, y, z position, and the yaw angle of the system w.r.t the ground truth coordinate through the use of “/dji_sdk/flight_control_setpoint_ENUposition_yaw.” When used with the velocity ROS function of “/dji_sdk/flight_control_setpoint_generic,” the ctrlVelYawRatePub publishes commands for the X, Y, Z velocity, and the rate of change of the yaw angle of the vehicle w.r.t. the ground truth coordinate.

```
Toggle line numbers
36 // Publish the control signal
37 ctrlPosYawPub = nh.advertise<sensor_msgs::Joy>("/dji_sdk/flight_control_setpoint_ENUposition_yaw", 10);
38 ctrlVelYawRatePub = nh.advertise<sensor_msgs::Joy>("/dji_sdk/flight_control_setpoint_generic", 10);
```

Figure 45. Example of Flight Control Coding. Source: ROS WIKI (2017).

The DJI ROS wrapper, presented in Appendix C, provides a different system control and telemetry to be read or written by the external payload processor. The ROS architecture simplifies the development process and enables various missions to be developed.

For the Zenmuse X3 software application, the imagery data from the Zenmuse X3 is contained within a proprietary interface that requires specific software driver libraries to be accessed and decoded. Using the “dji_cam_transport,” the imagery data is decoded and converted to RGB video format. Subsequently, the user can choose to have RGB video data displayed on the graphic interface on the Manifold, stored to NV12 memory, or transferred to the N1 flight controller for video downlink to the ground control station.

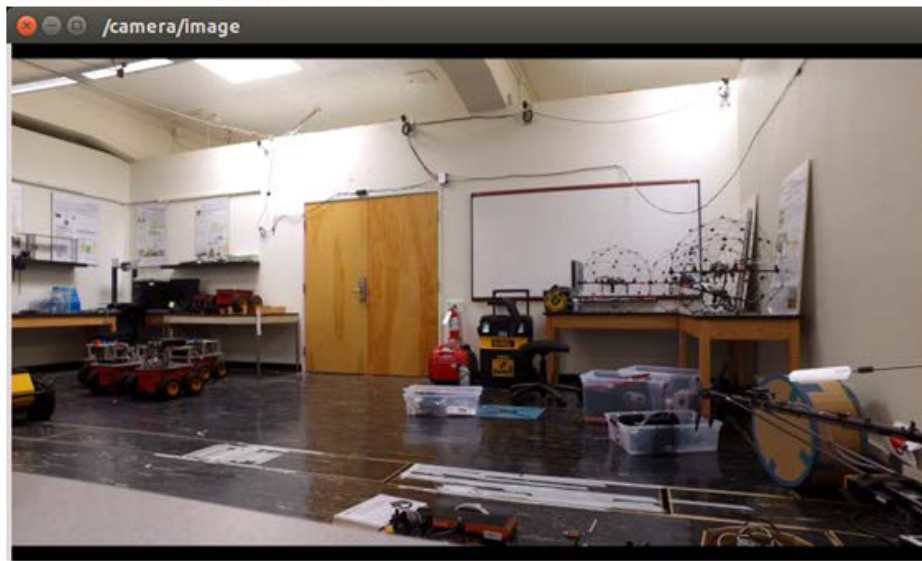
The system is also designed to enable users to debug the video stream by projecting the RGB video contents onto a ROS image viewer. This can be achieved through the use of a separate command terminal, using the command shown in Table 12:

Table 12. Command for Running the ROS Image Visualizer

<code>roslaunch image_view image:=/camera/image</code>	<code>//Launch Visualizer</code>
--	----------------------------------

Figure 47 presents a screen capture of the image viewer displaying the ROS image node in real time on the Manifold, followed by the video transmission displayed on the DJI Go application located at the ground control station.

ROS Visualizer of Image Node



Video Transmission Display on DJI GO Application

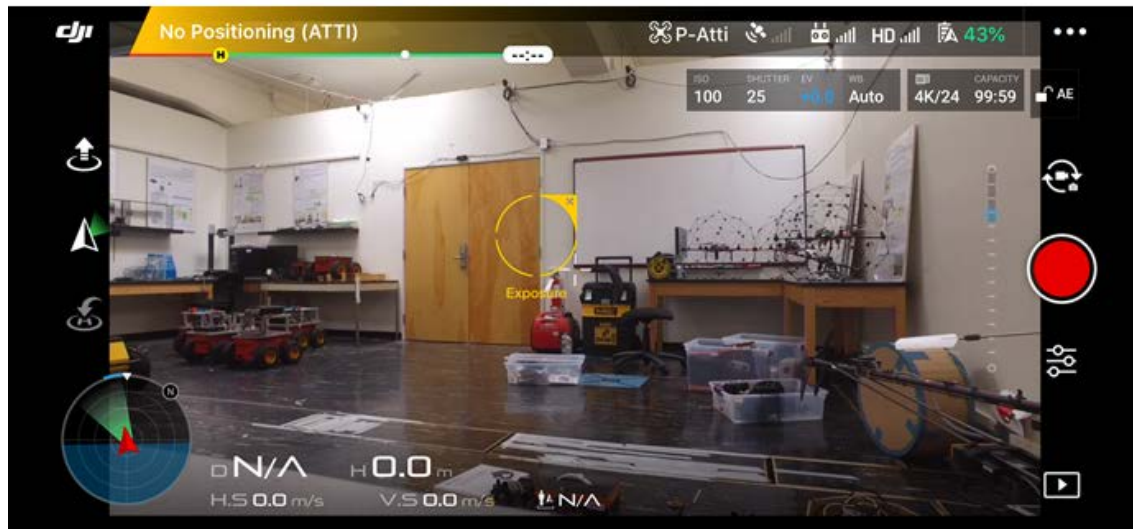


Figure 47. Visualizer of ROS Image Node (top) and Video Transmission Display on DJI Go Application (bottom)

3. State-of-the-Art Deep Learning-Based CV Algorithm

As discussed in Chapter III, it is evident that CV algorithm has a major role in supporting autonomous behavior for the unmanned system. Given that a wide variety of algorithms can be implemented to serve different applications, this thesis focuses on object

detection and tracking. Hence, the algorithm that is implemented on the system must be able to meet this criterion. Also, among the performance of different open-source CV algorithms, it is shown that YOLO is the better choice to achieve real-time embedded performance. The real-time performance will ensure timely updates on critical data for dynamic UAV flight control.

The YOLO algorithm is written in a custom deep learning architecture, Darknet, which enables acceleration of object detection capabilities in the CPU + GPU setting. The GPU core in the Nvidia embedded processor creates the framework favorable for the thesis research. Hence, it will be important to utilize the GPU and OpenCV software libraries support while compiling the YOLO algorithm. Appendix E presents the setup procedures for the building of the algorithm into the Manifold or similar GPU architecture-based processor system. Once the YOLO algorithm is built using the supporting software library, the algorithm is launched using a single line command, as illustrated in Figure 48. The command allows users to select the required dataset, YOLO configuration file, YOLO weights, and the type of input data (image or video).

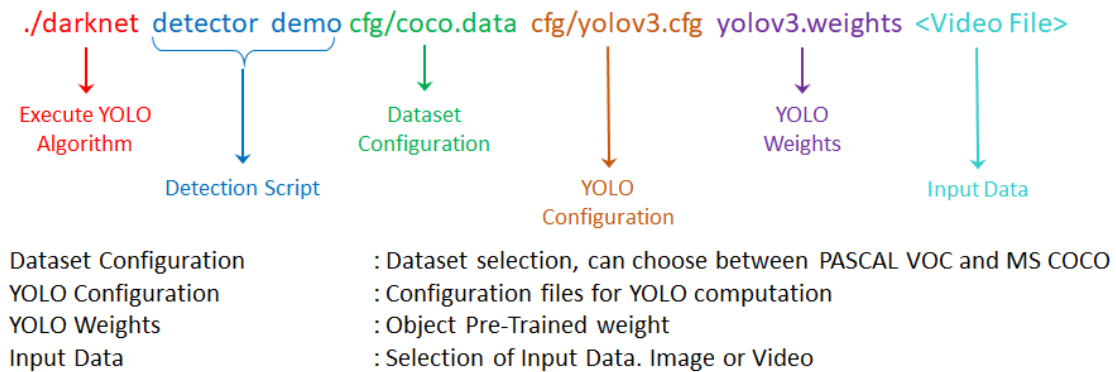


Figure 48. Decomposition of the YOLO (CV Algorithm) Command

For example, to test the performance of the YOLO algorithm, a single sample image can be loaded into the algorithm for object detection. Using the command shown in Table 13, the YOLO algorithm performs the object detection using the convolutional technique to classify the objects in the image.

Table 13. Command for Running the YOLO Algorithm

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights data/horse.jpg
```

Figure 49 presents the screen capture of a successful execution of the YOLO algorithm. As discussed in an earlier section, the convolutional network can be stacked to form different layers for object detection. It is observed that the “yolov3.cfg” configuration executes 106 process layers on a single image. In addition, the algorithm took 0.30275 seconds to detect a horse, dog, and person at respective confidence levels of 100, 99, and 100 percent.

```
nvidia@tegra-ubuntu: ~/darknet
78 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BFLOPs
79 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BFLOPs
80 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BFLOPs
81 conv 255 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 255 0.088 BFLOPs
82 yolo
83 route 79
84 conv 256 1 x 1 / 1 13 x 13 x 512 -> 13 x 13 x 256 0.044 BFLOPs
85 upsample 2x 13 x 13 x 256 -> 26 x 26 x 256
86 route 85 61
87 conv 256 1 x 1 / 1 26 x 26 x 768 -> 26 x 26 x 256 0.266 BFLOPs
88 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BFLOPs
89 conv 256 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 256 0.177 BFLOPs
90 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BFLOPs
91 conv 256 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 256 0.177 BFLOPs
92 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BFLOPs
93 conv 255 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 255 0.177 BFLOPs
94 yolo
95 route 91
96 conv 128 1 x 1 / 1 26 x 26 x 256 -> 26 x 26 x 128 0.044 BFLOPs
97 upsample 2x 26 x 26 x 128 -> 52 x 52 x 128
98 route 97 36
99 conv 128 1 x 1 / 1 52 x 52 x 384 -> 52 x 52 x 128 0.266 BFLOPs
100 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BFLOPs
101 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BFLOPs
102 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BFLOPs
103 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BFLOPs
104 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BFLOPs
105 conv 255 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 255 0.353 BFLOPs
106 yolo
Loading weights from yolov3.weights...Done!
data/person.jpg: Predicted in 0.302750 seconds.
horse: 100%
dog: 99%
person: 100%
```

Figure 49. Successful Execution of the YOLO Algorithm

Figure 50 illustrates the prediction results for object detection on the sample image that is loaded into the YOLO algorithm. The three objects are detected and bounded within their individual bounding box.

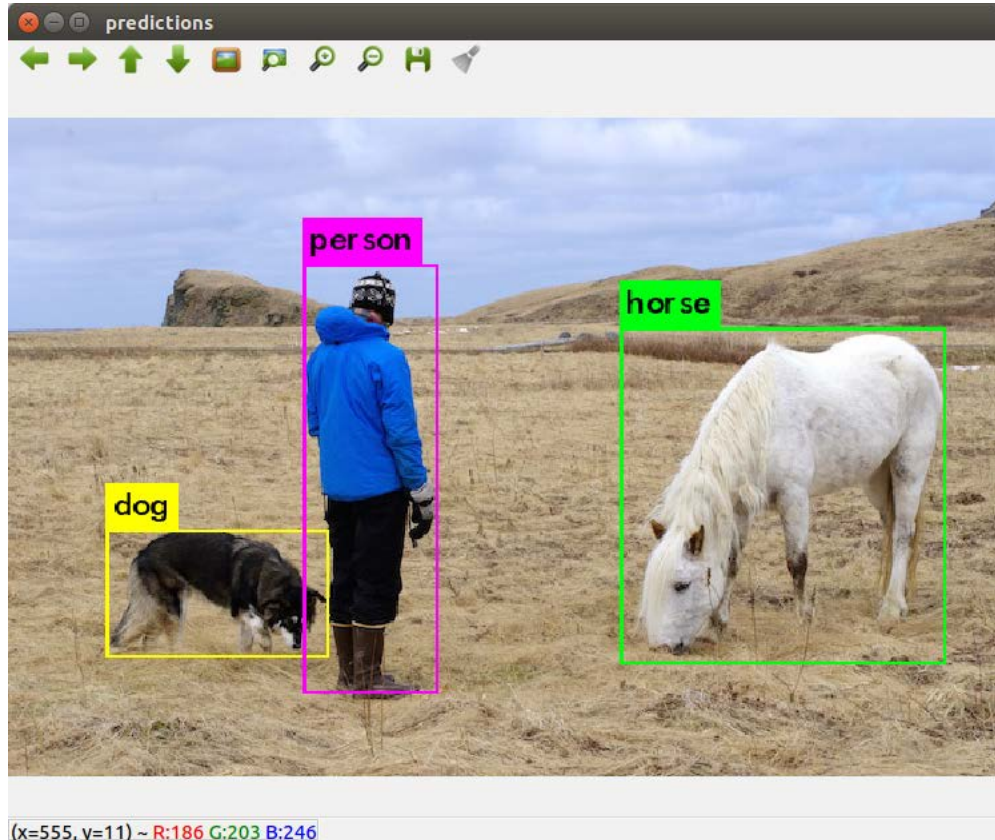


Figure 50. Predictions for Objects Detection on Sample Image

The dataset and pre-trained model used by the YOLO algorithm, however, are not comprehensive enough to provide object detection for all category classes. Hence, for specific object categories, the YOLO algorithm allows end users to perform the required training to achieve adequate object detection. For object detection training, it is necessary to determine the object of interest, which in this case is a quadcopter UAV. The complexity and precision of the object detection depends on the number of available image data sets that will be used to train the algorithm. As discussed by YOLO authors, the recommended

image data set for training is about 300–400 different images per category. Figure 51 illustrates a small subset of the image data for the quadcopter UAV.

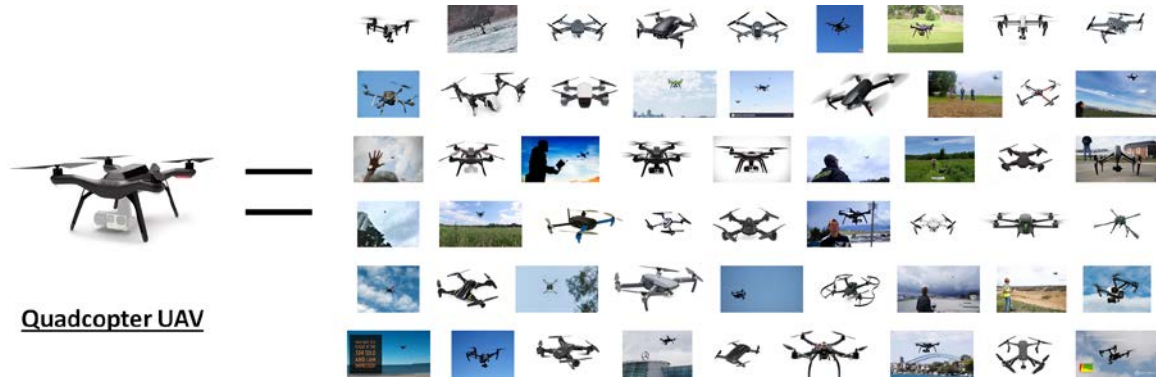


Figure 51. Image Data Set for the Quadcopter UAV

Next, each individual object is annotated to identify it in each individual image data set. The data annotation can be completed using software tools such as the BBox Label Tool. Figure 52 illustrates the data annotation of a quadcopter in a single image, whereby the object is bounded within a box of pixels from (307,545) to (817,238).



Figure 52. Data Annotation of Quadcopter in a Single Image

Hence, once the object within all the image data sets is annotated with the bounding box information, the data of the bounding box is stored in a data text file. This text file is subsequently converted to the YOLO required format as a preparation for YOLO algorithm training. The third step is to prepare the configuration files for the YOLO algorithm by defining the object names, the label to be shown, the convolutional weights, and the relevant computation setting. The final step is to use the algorithm to complete the training of the image data set in the data model, which can be used for testing the algorithm against detection of the desired object.

THIS PAGE INTENTIONALLY LEFT BLANK

V. T&E OF THE ADVANCED SYSTEM

This chapter discusses the results of T&E of the advanced system in a SAR mission. As introduced in the section I.A, the SAR mission can be a tedious and stressful task for human operators who need to locate potential survivors of a disaster as quickly as possible. Hence, the utilization of unmanned systems in the SAR mission reduces the workload for the human operator and enables prolonged mission time in finding the targets (i.e., the survivors). Following T&E of the advanced system, we collected and analyzed the test results from the system, and those results and their subsequent analysis are presented in this chapter.

A. EXPERIMENT SETUP

The focus of the experiment is to examine the relevant unmanned technologies and feasibility of the M100 system in supporting a SAR mission. The imagery data obtained from the aerial surveys during the mission can be processed by CV algorithms and applications to extract the critical information that can assist decision makers in allocating resources for the rescue operations. Depending on the complexity of the CV algorithm, different resource requirements are needed to process the imagery data. A combination of onboard and offline computation can provide the architecture to fulfill the requirements to effectively run the CV algorithm.

a. Mission Scenario

In the SAR mission, the unmanned system is capable to support human operators in numerous operational tasks. These tasks can be grouped into three key areas, which are defined as the mapping, search, and surveillance.

Prior to the search operation, it is essential to map out the area to gather critical information about the terrain or environment. The UAV system can provide the advantage for situational awareness mapping through its payload sensors. The M100's EO sensor is a form of remote sensing capability, which can enable the human operator to look beyond his or her visual limitations. Through the use of flight/mission planning tools, the flight

control algorithm on the M100 is programmed to perform the necessary waypoints flight to a specific location for imagery data collection. These imagery data are processed using the CV algorithm to generate 3D and 2D aerial maps or models for initial search and unmanned system navigation capabilities.

The purpose of a search operation is to achieve target identification at a close-in angle and assessment of the surrounding threat environments. Without this critical “ground” information, decision makers might not be in a good position to plan for further operations. Using the resultant outputs from the mapping operation, it is possible to develop an algorithm for obstacle avoidance to enable the M100 to execute flight in-between the buildings of an urban environment. Hence, the EO sensor can provide this optical flow to support this visual-to-flight-control algorithm. Currently, there are many ongoing research efforts to utilize EO sensors for UAV flight guidance in an urban environment. An example of the popular CV algorithm is visual SLAM, which provides UAV localization with and without the support of a GPS and mapping of the flight environment. A common issue with the implementation of the CV algorithm is the embedded processor’s computing power requirements. Therefore, a small, embedded processor unit is typically not suitable to run such a demanding algorithm.

In the surveillance operation, the system is required to supply the situational awareness about the designated area to ground control commanders. One of the strengths with the M100 UAV is its ability to hold its flight position (GPS coordinates and flight altitude). When combined with the RF data downlink, the system provides a continuous video stream for decision makers to monitor and plan the rescue activities. Nevertheless, the video quality can be affected by the parameters that defined the UAV flight mission. The flight altitude affects the GSD of the image, whereby the higher the altitude, the lower the GSD per pixels. For small objects (i.e., those objects with a small surface area), there is a need for higher GSD to achieve more pixels of objects. The video resolution affects the total ground area that can be monitored in a single snapshot and the data size of the video must be sent via downlink to the ground control. Hence, it is important to perform a trade-off study to achieve a balance between the desired video quality and system performance.

b. Operating Environment

The test and evaluation of the M100 system is conducted in Fort Ord's Impossible City. Figure 53 shows the aerial view (top) and elevated view (bottom) of Impossible City at Fort Ord, which provides an urban setting for the unmanned system to test different unmanned technologies in crafting the autonomous system's behavior. The system is tested during the period of January to June 2018. The environmental conditions were favorable and within the system specifications of the UAV for flight test execution.



Figure 53. Fort Ord's Impossible City. Source: Google Earth (top).

B. FIELD TRIALS RESULTS

In this section, the typical missions performed in SAR, namely mapping, search, and surveillance are simulated to demonstrate the capabilities of the advanced COTS UAV design and setup.

1. 3D Mapping

The first task of this field trial is to perform the aerial mapping of the desired operational area that might be defined by the ground control commanders. Using the number of pixels' representation for the object, as defined in Figure 19, the M100 is programmed to fly at an optimum flight altitude to (1) perform target detection from an aerial view (fly-pass mode) and (2) collect imagery data for modeling of the dense urban environment for unmanned system navigation.

In the mapping field experimentation, the UAV system is tasked to fly at an altitude of 50 m and execute imagery data collection over the area. The imagery data are stored in high resolution in the micro storage device, and compressed snapshots are sent from the system to the ground control station. The mission-planning tool used for this mapping task is the Pix4DCapture.

The Pix4DCapture is a third-party application developed by PIX4D to support a variety of DJI drones (such as the DJI Matrice 100 and Inspire 1). The strength of the PIX4D is its ability to do 2D aerial imagery and 3D mapping over the area of interest. In terms of its user friendliness, the application is easy to operate, and untrained users can learn the functions of the application within 1–2 hours of experimenting with it. On the Pix4DCapture user interface, the mission selection allows for mapping of the area in single or double grid manner upon entering the application.

The search area is established via the selection of the polygon or by dragging the search box over the template. Next, users are allowed to determine the desired flight altitude, angle of the camera, front and side overlap of the images captured, and the drone speed moving along the waypoints in the search area. Figure 54 shows the screen capture of the area waypoint mapping of 210 m x 210 m for Fort Ord's Impossible City. At the flight altitude of 50 m, angle of the camera at 90 degrees, front overlap of 80 percent and

side overlap of 70 percent for capturing the images, and the drone speed of fast, the estimated flight time for this image-mapping mission will be approximately 16 minutes 41 seconds. In addition, one other feature that is the strength of this application is its ability to provide the ground sample distance (GSD) measured as centimeters/pixels. This gives the operator a quick reference guide to determine the performance of the imaging when flying at a certain flight altitude.

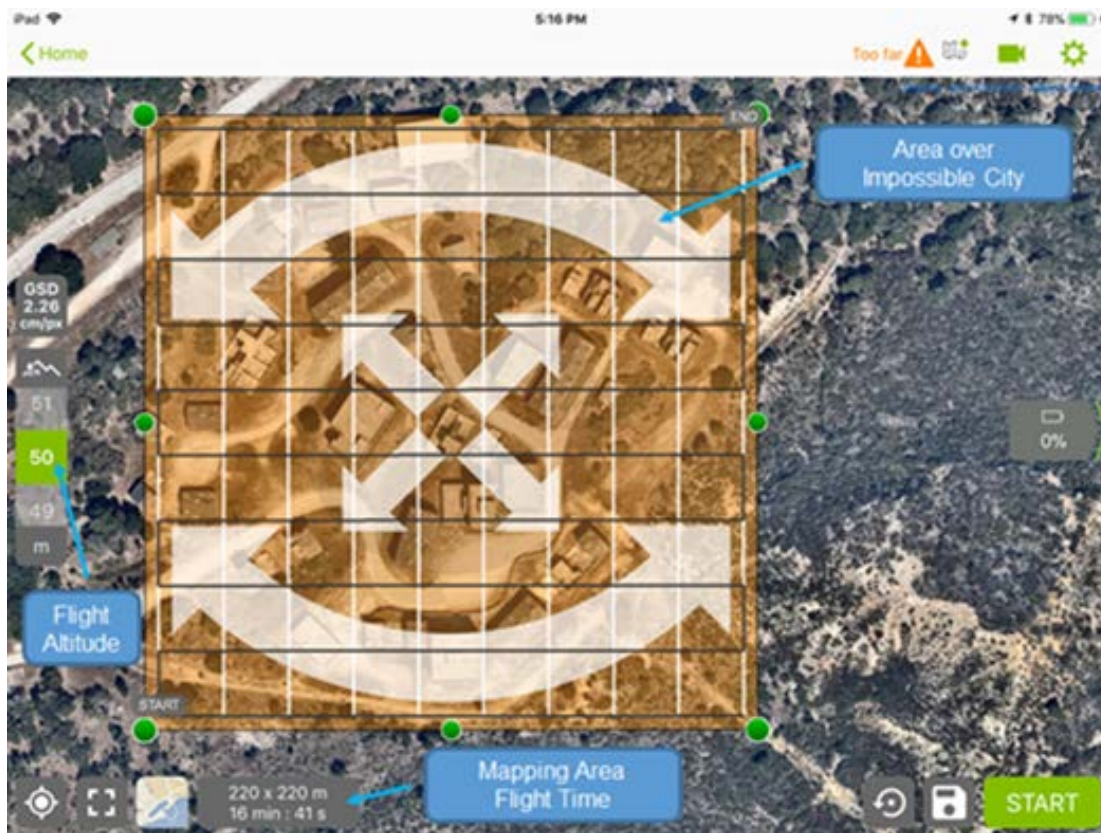


Figure 54. Illustration of Area Mapping in Pix4DCapture

Once the M100 system completed its imagery data collection, the imagery data are extracted from the system and processed on an offline computing system to generate the 3D mapping. The 3D mapping of the search area is an extension to the SLAM concept, where the image snapshots captured by the UAV during the aerial fly-over of the desired area are post-processed to generate a 3D point cloud for unmanned system navigation in the urban environment. Agisoft Photoscan software was selected as the primary solution to

fulfill the geo-referencing task for aligning the imagery data (with overlap) using the metadata for airborne GPS data and its proprietary point-cloud technology algorithm. Figure 55 shows an illustration of the 3D point cloud for Impossible City generated from the imagery data of the UAV.



Image Credit: Jeremy Metcalf

Figure 55. 3D Point Cloud Model of Impossible City

However, the amount of detail in the 3D point cloud is simply too much for the embedded processor within the unmanned system to perform meaningful machine learning for feasible navigation. Furthermore, the file size of a typical 3D point cloud model is in the order of 10–15 GB per 10000 m². For the mapping area of 210 m by 210 m, the expected file-size will be between 20 and 30 GB. Thus, the 3D point model is simplified into a 3D

mesh model by converting the points into vertex and reduced points, achieving significant file-size reduction. Figure 56 shows the screen capture of the 3D mesh model that had been further processed from the 3D point cloud model. The mesh model is the targeted end products, which can be used by the computer-vision algorithm to perform the required unmanned system navigation.

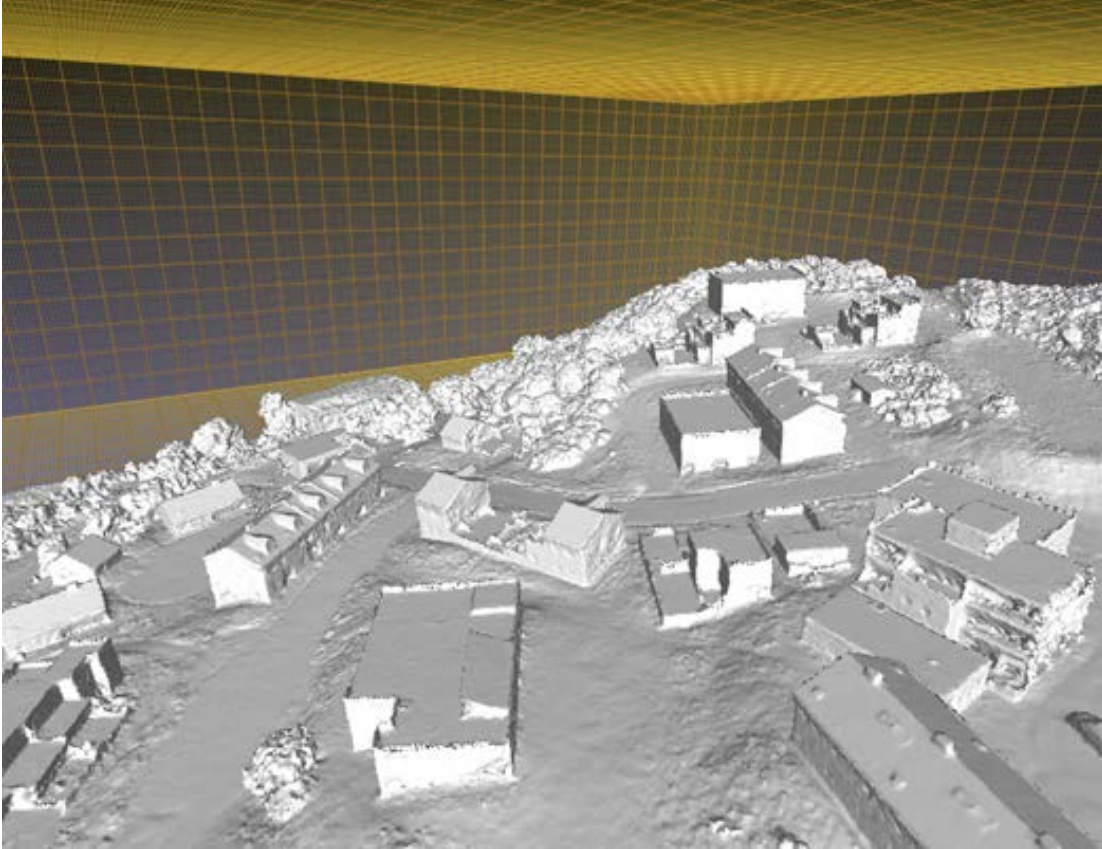


Figure 56. 3D Mesh Model of Impossible City

In general, the mesh model will enable the unmanned system to perform localization of its position in the model, determine the critical dimensions (length, width, and height) and location of the obstacles (i.e., building, tree), and perform the distance to go/avoid. All of these concepts are aimed at machine learning for the required vision-based navigation of the system.

2. 2D Aerial Imagery

The next product that can be extracted from the imagery data is the 2D aerial imagery. The 2D aerial imagery is used as the basis in the CV algorithm for target detection, and identification of the road/tracks from the 2D aerial mapping. Figures 57 and 58 show the 2D aerial imagery data in X-and Y-orientation generated from the image snapshots captured by the UAV. In order to preserve the clarity and precision of the GSD resolution per pixel, the images are stitched together to form a high-resolution map. When the map is zoomed in, the GSD resolution per pixel will be the same as the parameters that had been considered.



Figure 57. 2D Aerial Imagery of Impossible City (in X-Orientation)

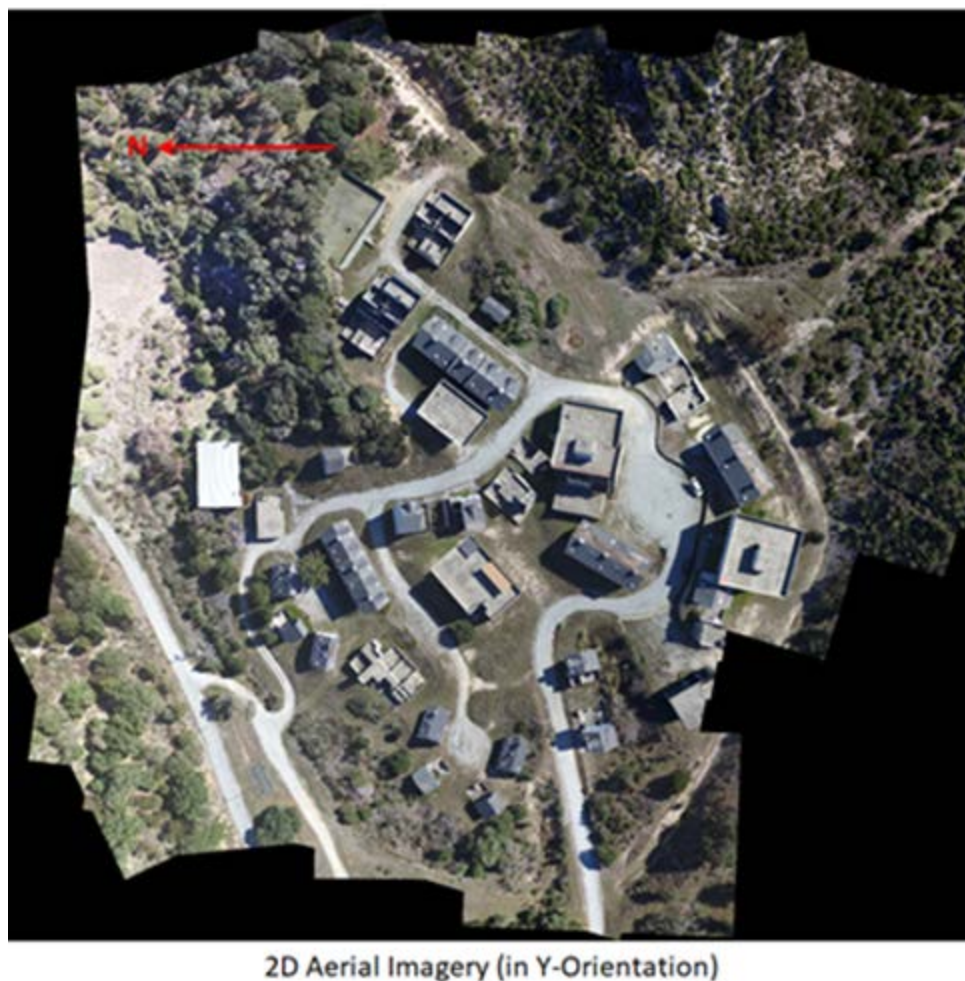


Figure 58. 2D Aerial Imagery of Impossible City (in Y-Orientation)

Using a target detection algorithm or man-in-the-loop operation technique, the targets within the 2D aerial imagery data can be detected (as shown in Figure 59). The imagery data provides a quick mean to detect the targets within the first fly-pass mission. If the targets are not detected in the images, areas that provide a high confidence level that the targets do not exist are isolated. This reduces the overall operation lead-time in the search and rescue mission.



Figure 59. Targets Detection on the 2D Aerial Imagery Map

Since the 2D aerial imagery exists in the X-and-Y orientation, the same process can be applied to the 2D aerial imagery data in a Y-axis orientation. The resultants will provide the opportunity to the ground control commanders and operators for alternative views of the targets and can provide the man-in-the-loop identification process for confirmation. For example, Figure 60 shows the screen capture of the targets found from the 2D aerial imagery in the X-and Y-axis orientation. The first row illustrates the detection of a friendly trooper; one can see that the Y-axis provides an alternative view to the soldier conditions. Similarly, the second row illustrates a civilian awaiting rescue support; when comparing the snapshots from X-axis versus Y-axis, one can see that the Y-axis image provides slightly more information about what the civilian victim is doing and perhaps the immediate threats in his surroundings.



Figure 60. Target Detection on the 2D Aerial Imagery for X- and Y-Axis Orientation

Furthermore, using the 2D aerial imagery, the algorithm for probability road map will generate the possible track/road visible from the 2D aerial imagery data. The identification of the road/track will enable the unmanned system to plan for the required movement path in the search area. Figure 61 shows the representation of the probability road map in image form for Impossible City. It is evident that the tracks in the image are shown in white, which provides a high contrast perspective of the possible movement track within Impossible City.



Figure 61. Representation of Probability Road Map for Impossible City

3. Simulation of a Search Mission

The second task of this field trial is to perform a target search via execution of a low-altitude flight along the identified tracks/road from the possibility road map to look for potential targets between buildings, along the side of a building, and outside building entrances/exits. The speed of the target search is dependent on the platform velocity and the field-of-view of the EO sensor. The imagery data is processed by the objects detection CV algorithm embedded in the onboard computer for real-time performance. Furthermore, there might be situation in which the UGV can be deployed to complement the search tasking by looking into areas that the UAV cannot access.

Since the purpose of the search operation is to perform target detection, a CV algorithm such as YOLO is used to search for objects of interest. As discussed in an earlier section, the CPU+GPU hardware architecture on the Nvidia processor enables a CNN-based algorithm to accelerate its image processing functions. The M100 with the payload hardware is utilized to test the YOLO algorithm. As for the flight-path planning to conduct the search, mission-planning tools can use the generated 2D aerial imagery to create a waypoints flight path for the M100 system. In the case of this thesis, the application that is used to define the target search flight is DJI Ground Station Pro.

The DJI GS Pro developed by DJI allows users to set the desired flight location for the UAV flight operation. Based on the 3D point cloud and 2D aerial imagery map result, the UAV is programmed to fly along the movement path into the desired area to look for the target. Figure 62 shows the screen capture of the target search mission waypoints for the UAV to look for the target in the pre-defined mission set.

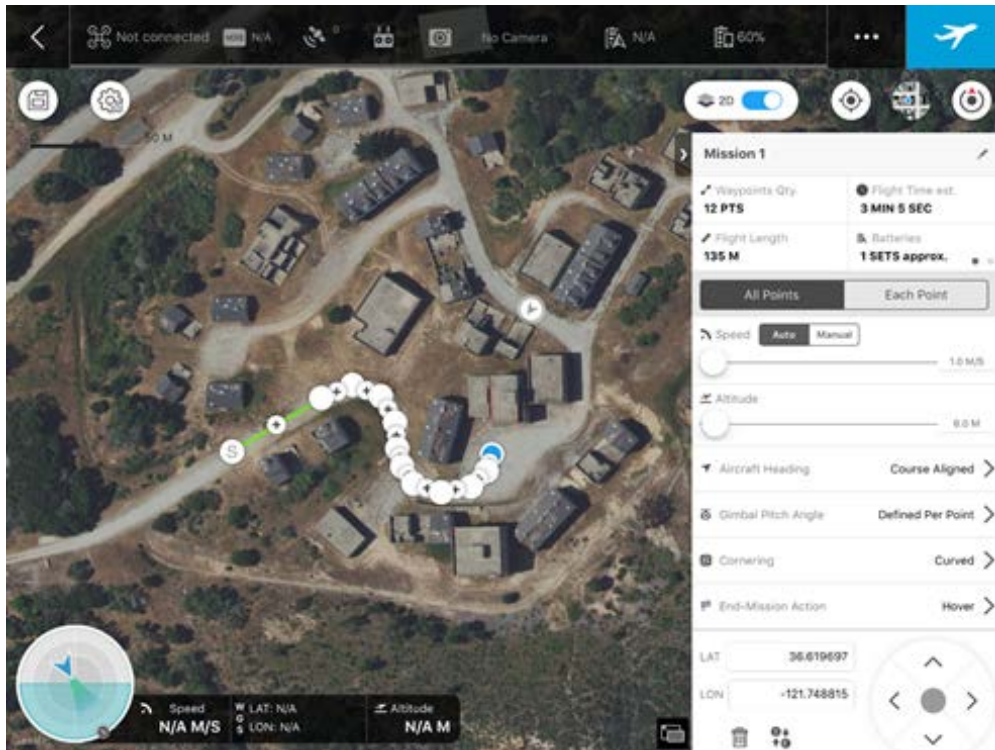


Figure 62. Screen Capture of the Target Search Mission Waypoints

In the field trials, it is discovered that the mission-planning tool lacks in ability to read in the 3D point cloud model terrain information. Due to the terrain of Impossible City, the flight altitude of the UAV should not be constant throughout the flight. An additional modeling tool is used to extract the terrain information from the 3D model. Given that the 3D model is not survey grade, though, the elevation information lacks in precision. Thus, a tolerance of 5 m is used as a safety margin in the flight trial. To cater to the terrain changes, if the desired flight altitude is 10 m off the ground, the UAV performs continuous adjustment in its flight altitude as the terrain under its flight path changes. Hence, in the case of the flight mission, if the starting altitude is 10 m and the ending altitude is defined as 20 m, the flight altitude is increased by 10 m gradually as the terrain elevation changes.

The DJI UAV system is also designed to operate with two sets of remote controllers for independent control over the flight dynamics and the gimbal camera on the UAV. Due to flight safety considerations, the master controller will always have unique control over the flight dynamics and shared control over the gimbal control. On the other hand, the slave controller does not have control over the flight dynamics of the UAV, but just the control of the gimbal camera.

During the target detection operation, the gimbal camera is pointed forward at the heading angle of the UAV forward flight. As discussed in an earlier section on object resolution, the change in distance of the objects to the EO sensor will change the number of pixels' representation for the objects, as seen in Figure 19. Hence, the YOLO algorithm will require a specific pixel threshold for successful target detection. Figure 63 shows the viewing scenario from the UAV during the Target Search mission. It is evident that the per pixel resolution of the objects decreases as the distance from the camera to the object increases. Thus, a complex problem results if the resolution is required to be predicted.



Figure 63. Viewing Scene of the UAV during Target Search

When the imagery data is decoded and processed by the YOLO algorithm, the algorithm utilizes the CNN architecture framework to process the data in each individual frame. In the field trial, two different object categories (person and car) are used to represent the target. Figure 64 presents the snapshot of the processed imagery data for target detection with the YOLO algorithm. The four snapshots represent various scenarios that the UAV might encounter during its mission. The top row images represent detection of a single individual object category (i.e., person and car) within a single image. The bottom left image represents the detection of two different object categories separated by a specific distance, and the bottom right image represents the detection of two different object categories overlapping each other. It is evident that the YOLO algorithm works well in the four scenarios and is able to detect the object as captured by the EO sensor on the M100 system. It is also observed that the YOLO has little probability (confidence) in object classification when the object appears to be small (far from the system). The probability in object classification increases as the object gets nearer to the system (pixel resolution increases).



Figure 64. Snapshot of the Target Detection in YOLO Algorithm

4. Simulation of a Surveillance Mission

The third and final task of this field trial is to provide a means for the ground control commander to monitor the activities occurring over the desired SAR mission area. The UAV is programmed to hover in position over the designated area at a defined flight altitude. The RF data downlink will be responsible to send the data and video telemetry from the UAV to the ground control station.

In the surveillance field experiment, the UAV is programmed to hover at different flight altitudes to perform an assessment of its effect on the viewable ground area. By using the same mission planning tools (DJI GS Pro), the M100 system is programmed to hover at the defined flight altitude. The imagery data are compressed and sent at full HD video resolution of 1920 x 1280 pixels, and the recorded videos are stored at the 4K video resolution of 4096 x 2160 pixels. The compression of the video bitrate helps to reduce the latency in video transmission, thus enabling the real-time updates of events to the ground control station. Figure 65 shows the illustration of the video resolution of the Zenmuse X3 when operated in 4K or HD mode.



Figure 65. Representation of Video Resolution Based on Generic Pictorial.
Source: DJI (2017).

Similar to the mapping operation, the surveillance coverage (ground distance) can be estimated from the flight altitude using the ground sample distance. Figure 66 shows the viewable ground area versus the flight altitude of the UAV. Assuming that the ground control commander will need to view an area of 125m-by-125m (total surface area =15625m²), the minimum flight altitude of the UAV will be approximately 200 m with full HD real-time video stream. On the other hand, for still image snapshots, the UAV can be operated at a flight altitude of around 82.5 m to achieve the same ground coverage.

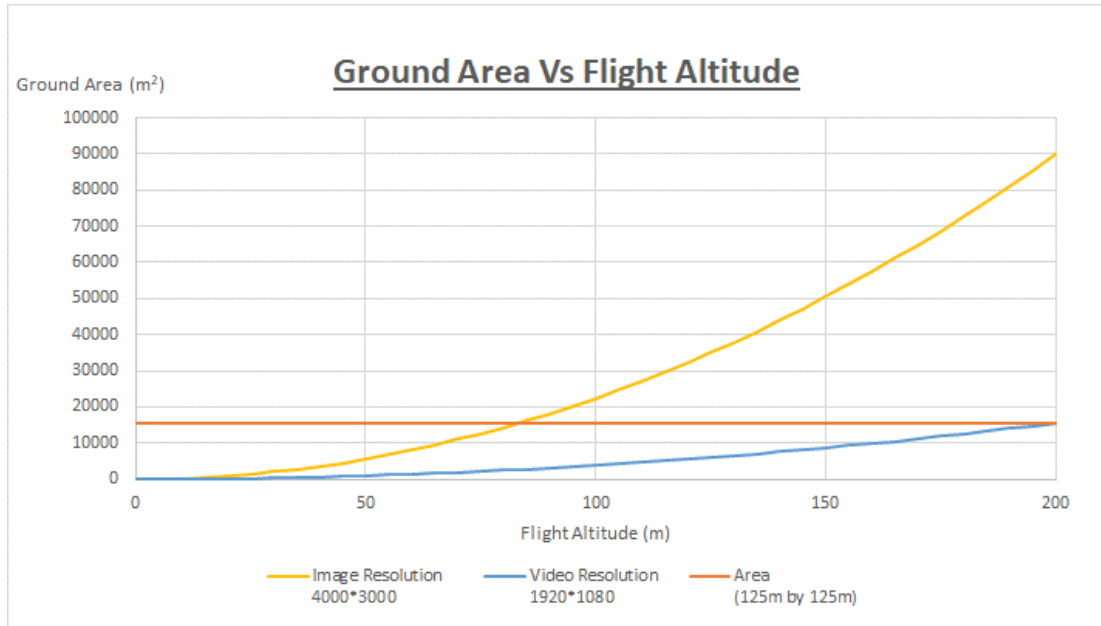


Figure 66. Plot for Ground Area versus Flight Altitude

As part of the field trial, the effect of the viewable ground area against the flight altitude is studied. The UAV is programmed to hover at different flight altitudes (50 m, 100 m, 150 m, 200 m). Figure 67 shows various image snapshots over Impossible City at different flight altitudes. It is evident that the viewable area increases in a slightly exponential relationship as the flight altitude increases. Also, as expected, the GSD resolution decreases due to the increase in the flight altitude. Therefore, it will be necessary to perform trade-off studies between the GSD resolution and the viewable ground area prior to the decision on the flight altitude.



Flight Altitude = 50m



Flight Altitude = 100m



Flight Altitude = 150m



Flight Altitude = 200m

Figure 67. Snapshot of Impossible City at Different Flight Altitudes

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

This thesis explored areas for advancement in COTS UAV capability to provide vision-based SA/ISR data through the use of an EO sensor and image processing algorithm. Prior to addressing the system improvements, the problems with the baseline platform were systematically identified and analyzed. Through the analysis, the three identified problems (i.e., Sensitivity of Classifier Parameter, Overlap in Bounding Box, and Rolling Shutter Effects) were attributed to the performance of the CV algorithm and EO sensor.

The rolling shutter effects degraded the quality of the captured image and affected the overall performance of the Purdue/NPS CV algorithm. Since the rolling shutter capturing mechanism is an inherent property of the existing EO sensor (Logitech C920 webcam), it became necessary to develop and integrate new payload electronics on the UAV to support the advanced vision-based capability. Given that the M100 baseline platform supports a DJI Zenmuse X3 gimbal camera (global shutter EO sensor), it is necessary to deploy the relevant electronics to decode and process the imagery data from the new EO sensor. Therefore, system considerations for the EO sensor and computing hardware technologies were conducted to understand the technologies' expected performance and limitations.

Besides the identified problems with the Purdue/NPS CV algorithm, the algorithm required the EO sensor to be stationary for capturing of the objects' scene. Hence, the algorithm also limits the capability of the M100 system. As such, the current advancement in the CV algorithm is reviewed. The convolutional-based algorithm is found to have the best computational effectiveness in object detection. In a comparison of the architecture and performance of various CNN algorithms, the YOLO algorithm is selected due to its ability to perform object detection in a real-time manner.

The new advanced system is tested and evaluated for its capability in a simulated SAR mission. In order to establish a realistic mission scenario, the SAR mission was broken down into three operating phases, such as mapping, target search, and surveillance.

In the mapping operation, the system was programmed to capture imagery data over the operational area (Impossible City, Fort Ord). The imagery data were processed by CV-based software to generate a 3D mapping model, 2D aerial imagery, and probability road map, which can be utilized to support autonomous navigation in unmanned system. Furthermore, an additional CV algorithm can be executed to perform a first-pass target search based on the initial imagery data. In the search operation, the system is tasked to fly along the tracks in the built-up urban environment. Based on the collected imagery data, YOLO (CV algorithm) is able to perform object detection (human and vehicle) in four different scenarios. The successful detection indicated that the newly added payload electronics (Zenmuse X3 and Manifold) resolved the rolling shutter effects in the captured data. Finally, in the surveillance operation, the system was programmed to provide imagery data over a specific designated area. The imagery data were then fed to the ground control station as live video.

In conclusion, this thesis demonstrated advancements in COTS UAV capability through the use of an EO sensor and CV algorithm to fulfill mission needs. With the improvements pursued in this thesis, the M100 platform (COTS UAV) can be quickly adapted for military missions. Furthermore, the use of unmanned technology increases situational awareness and reduces the workload for its users. Hence, the operation of unmanned systems can likely provide users with an advantage in modern warfare.

B. RECOMMENDATIONS FOR FUTURE WORK

As this research demonstrated, the added payload electronics resolved the issue of rolling shutter effects on degraded imagery data. Along with the generic software architecture designed for the computing element, the new payload electronics will enable a wide variety of algorithms or software applications to be executed. With this in mind, future research areas can involve:

- Training of the YOLO algorithm—Research into machine learning has attracted interest for numerous civilian and military applications. The YOLO algorithm is designed for highly efficient processing of object detection using embedded processor electronics. Therefore, the ability to

define the object model, training of the CV algorithm, and detection of a specific object facilitates customized system development.

- Optimization of optical flow for EO sensors—New payload electronics convert the imagery data from the memory buffer to the ROS image node. However, ROS is not optimized for image processing, and it induces minor latency in the optical flow chain. Hence, one of the potential means for optimization is to utilize the OpenCV software library to handle all imagery data processing requirements.
- Flight control algorithm—Research and development of the flight control algorithm will be relevant for customized missions and applications. Since the M100 platform can utilize ROS commands for user-defined flight applications, the DJI ROS SDK has been setup in the new payload electronics. Thus, this algorithm can provide the ability to control the M100 system to users' requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DJI MANIFOLD SETUP

This appendix presents the setup procedures for the DJI Manifold. To reduce the installation effort for future development, the setup codes have been written as script files for quick installation.

A. CODE FOR CUDA INSTALLATION

```
1. #####
2. #   INSTALL CUDA 6.5 on Nvidia TK1   #
3. #####
4.
5. # 1. Download Nvidia CUDA 6.5 Repository
6.
7. Wget http://developer.download.nvidia.com/embedded/L4T/r21_Release_v3.0/cuda-repo-l4t-
   r21.3-6-5-prod_6.5-42_armhf.deb
8.
9. # 2. INSTALL THE Repository
10.
11. sudo dpkg -i cuda-repo-l4t-r21.3-6-5-prod_6.5-42_armhf.deb
12. sudo apt-get update
13. sudo apt-get install cuda-toolkit-6-5
14.
15. # 3. SET THE AUTHORITY OF USER
16.
17. sudo usermod - a - G video Ubuntu
18.
19. # 4. SET THE ENVIRONMENT VARIABLES OF SYSTEM AND TAKE THEM INTO EFFECTS
20.
21. echo ' ' >> ~/.bashrc
22. echo '# Cuda dependencies' >> ~/.bashrc
23. echo 'export PATH=/usr/local/cuda-6.5/bin:$PATH' >> ~/.bashrc
24. echo 'export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib:$LD_LIBRARY_PATH' >> ~/.bashrc
25.
26. # 5. RUN CHECK
27.
28. nvcc - V
```

B. CODE FOR OPENCV INSTALLATION

```
1. #####
2. # INSTALL OPENCV ON UBUNTU 14.04 LTS #
3. #####
4.
5. # VERSION TO BE INSTALLED
6.
7. OPENCV_VERSION = '3.0.0'
8.
9. # 1. KEEP UBUNTU OR DEBIAN UP TO DATE
10.
11. sudo apt-get -y update
12. sudo apt-get -y upgrade
13. sudo apt-get -y dist-upgrade
```

```

14. sudo apt-get -y autoremove
15.
16. # 2. INSTALL THE DEPENDENCIES
17.
18. # Build tools:
19.     sudo apt-get install -y build-essential cmake gif
20.
21. # GUI( if you want to use GTK instead of Qt, replace 'qt5-default'
22.     with 'libgtkglext1-dev' and remove '-DWITH_QT=ON' option in CMake):
23. sudo apt-get install -y qt5-default libvtk6-dev qtbase5-dev
24.
25. # Media I / O:
26. sudo apt-get install -y zlib1g-dev libjpeg-dev libwebp-dev libpng-dev libtiff5-
    dev libjasper-dev libopenexr-dev libgdal-dev
27.
28. # Video I / O:
29. sudo apt-get install -y libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-
    dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev yasm libopencore-amrnb-
    dev libopencore-amrwb-dev libv4l-dev libxine2-dev
30.
31. # Parallelism and linear algebra libraries:
32. sudo apt-get install -y libtbb-dev libeigen3-dev
33.
34. # Python:
35. sudo apt-get install -y python-dev python-tk python-numpy python3-dev python3-
    tk python3-numpy
36.
37. # Java:
38. sudo apt-get install -y ant default-jdk
39.
40. # Documentation:
41. sudo apt-get install -y doxygen
42.
43.
44. # 3. INSTALL THE LIBRARY
45.
46. sudo apt-get install-y unzip wget
47. wget https://github.com/opencv/opencv/archive/${OPENCV_VERSION}.zip
48. unzip ${OPENCV_VERSION}.zip
49. rm ${OPENCV_VERSION}.zip
50. mv opencv-${OPENCV_VERSION} OpenCV
51. cd OpenCV
52. mkdir build
53. cd build
54. cmake - DCMAKE_BUILD_TYPE = Release - DCUDA_GENERATION = Kepler - DWITH_OPENMP = ON..
55. make - j2
56. sudo make install
57. sudo ldconfig

```

C. CODE FOR ROS INDIGO INSTALLATION

```

1. #####
2. # INSTALL ROS INDIGO ON Ubuntu 14.04 #
3. #####
4.
5. # 1. Setup ROS sources list
6.
7. sudo sh - c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -
    sc) main" > /etc/apt/sources.list.d/ros-latest.list'

```



```
8.
9. # 2. SETUP your Keys
10.
11. sudo apt - key adv--keyserver hkp: //ha.pool.sks-keyservers.net:80 --recv-
    key 421C365BD9FF1F717815A3895523BAEEB01FA116
12.
13. # 3. INSTALLATION
14.
15. sudo apt-get update
16. sudo apt-get install ros-indigo-desktop
17. # sudo apt-get install ros-indigo-ros-base
18.
19. # 4. INITIALIZE ROSDEP
20.
21. sudo rosdep init rosdep update
22.
23. # 5. ENVIRONMENT SETUP
24.
25. echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
26. source ~/.bashrc
27.
28. # 6. GETTING ROSINSTALL
29.
30. sudo apt-get install python-rosinstall
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DJI ONBOARD SDK

This appendix presents the setup procedures for the SDK software package that is used for the development of the flight controller algorithm. The setup procedures are extracted from DJI Onboard SDK documentation.⁴ We recommend that the SDK packages be processed and executed from a common folder (i.e., DJI_SDK) within the Manifold.

A. INSTALLING THE SDK CORE

1. Clone the DJI OSDK from Github.⁵ In this research, we used Onboard-SDK 3.2 as the version for the DJI SDK.
2. Open the terminal and locate the SDK folder.
3. Execute the following commands to build the SDK:

```
mkdir  
cd build  
cmake ..  
make dji-sdk-core
```

4. Perform installation of the sdk-core library to the Manifold system,

```
sudo make install dji-sdk-core
```

B. INSTALLING THE SDK ROS NODES

5. Create a catkin workspace using the following commands:

```
mkdir catkin_ws  
cd catkin_ws  
mkdir src  
cd src  
catkin_init_workspace
```

⁴ DJI Onboard SDK information is located at <https://developer.dji.com/onboard-sdk/documentation/introduction/homepage.html>.

⁵ The GitHub repository for DJI OSDK is found at <https://github.com/dji-sdk/Onboard-SDK/tree/3.2>

6. Clone the DJI OSDK-ROS from GitHub.⁶ In this research, we used Onboard-SDK-ROS 3.2 as the version for DJI SDK.
7. Execute the following commands to build the SDK ROS package and SDK Demo ROS package:

```
cd ..  
catkin_make
```

C. CONFIGURATION OF THE DJI SDK

8. Source the catkin workspace setup.bash file using the following command:

```
source devel/setup.bash
```

9. Edit the launch files and enter the App ID, Key, Baudrate, and Port Name in the designated place using the following command:

```
roscd dji_sdk sdk_manifold.launch
```

10. Use the following information for the launch files.

Description	Values
App Name	Manifold_OpenCV
App ID	1055627
App Key	a070e74ad43621a75b65ade05b82b6bd4bc406aa8778b946ec90cbd10ffab7e7
Baudrate	115200
Port Name	/dev/ttyTHS1 (UART2)

D. EXECUTION OF THE SDK PROGRAM

11. Start up the SDK ROS core package using the following command:

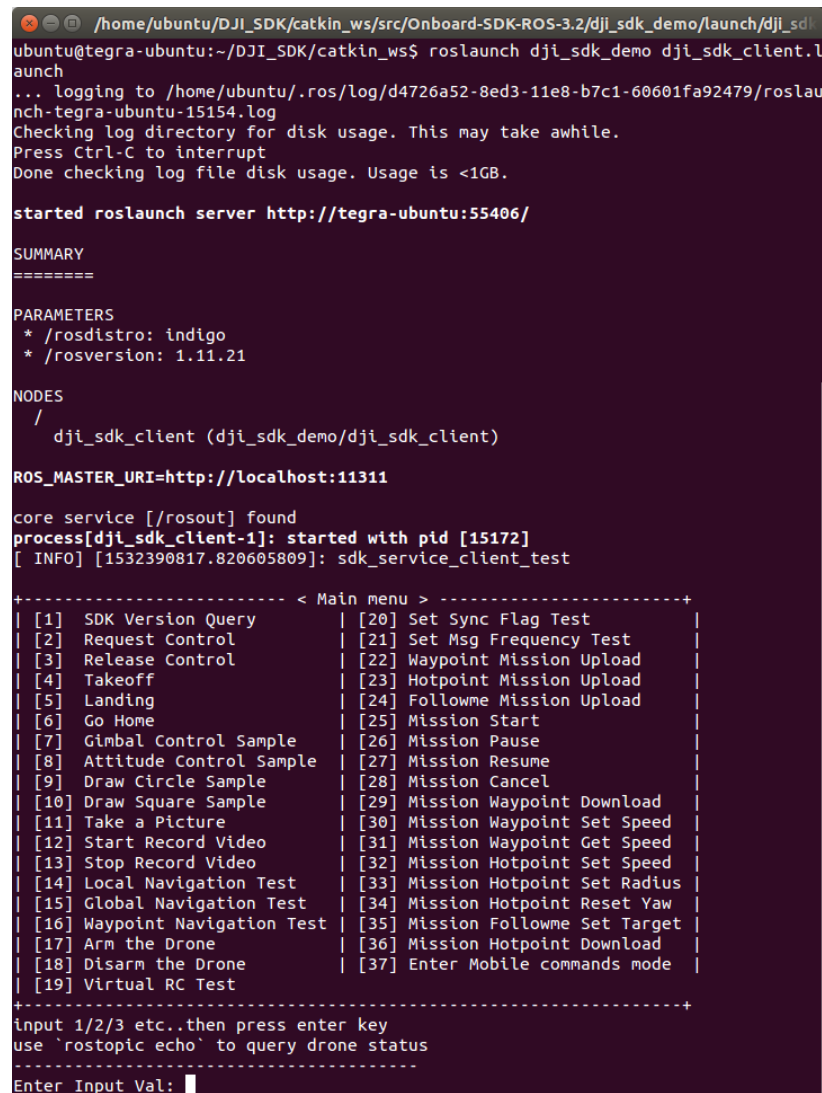
```
roslaunch dji_sdk sdk.launch
```

⁶ GitHub repository for DJI OSDK-ROS is <https://github.com/dji-sdk/Onboard-SDK-ROS/tree/3.2>

12. Open up another terminal (Ctrl+T), and cd into the catkin workspace location.
13. Execute the following commands to start the demo client software:

```
source devel/setup.bash
roslaunch dji_sdk_demo dji_sdk_client.launch
```

14. Choose the demo application for the UAV to execute, as illustrated in Figure 68.



```
/home/ubuntu/DJI_SDK/catkin_ws/src/Onboard-SDK-ROS-3.2/dji_sdk_demo/launch/dji_sdk_client.launch
ubuntu@tegra-ubuntu:~/DJI_SDK/catkin_ws$ roslaunch dji_sdk_demo dji_sdk_client.launch
... logging to /home/ubuntu/.ros/log/d4726a52-8ed3-11e8-b7c1-60601fa92479/roslaunch-tegra-ubuntu-15154.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://tegra-ubuntu:55406/

SUMMARY
=====

PARAMETERS
* /roscdistro: indigo
* /rosversion: 1.11.21

NODES
/
  dji_sdk_client (dji_sdk_demo/dji_sdk_client)

ROSMasterURI=http://localhost:11311

core service [/rosout] found
process[dji_sdk_client-1]: started with pid [15172]
[ INFO] [1532390817.820605809]: sdk_service_client_test

+-----+-----+-----+-----+-----+-----+
| [1] SDK Version Query | [20] Set Sync Flag Test |
| [2] Request Control | [21] Set Msg Frequency Test |
| [3] Release Control | [22] Waypoint Mission Upload |
| [4] Takeoff | [23] Hotpoint Mission Upload |
| [5] Landing | [24] Followme Mission Upload |
| [6] Go Home | [25] Mission Start |
| [7] Gimbal Control Sample | [26] Mission Pause |
| [8] Attitude Control Sample | [27] Mission Resume |
| [9] Draw Circle Sample | [28] Mission Cancel |
| [10] Draw Square Sample | [29] Mission Waypoint Download |
| [11] Take a Picture | [30] Mission Waypoint Set Speed |
| [12] Start Record Video | [31] Mission Waypoint Get Speed |
| [13] Stop Record Video | [32] Mission Hotpoint Set Speed |
| [14] Local Navigation Test | [33] Mission Hotpoint Set Radius |
| [15] Global Navigation Test | [34] Mission Hotpoint Reset Yaw |
| [16] Waypoint Navigation Test | [35] Mission Followme Set Target |
| [17] Arm the Drone | [36] Mission Hotpoint Download |
| [18] Disarm the Drone | [37] Enter Mobile commands mode |
| [19] Virtual RC Test |
+-----+-----+-----+-----+-----+
input 1/2/3 etc..then press enter key
use `rostopic echo` to query drone status
-----
Enter Input Val: 
```

Figure 68. Successful Execution of the DJI SDK Demo Client

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. DJI SDK ROS WRAPPER

This appendix presents the DJI SDK software package that can be utilized for the development of a user-specified flight controller algorithm. All information of the DJI SDK ROS wrappers and their functional descriptions are copied directly from the wiki site of ROS.org.⁷

A. OVERVIEW

This package provides a ROS interface for the DJI onboard SDK and enables the users to take full control of supported platforms (DJI M100, M600, M210, or drones equipped with A3/N3 flight controllers) using ROS messages and services.

B. SUBSCRIBED TOPICS

1. Flight Control Topics

The user sends flight control setpoints to the drone by publishing one of the following topics, which are subscribed by the `dji_sdk` node. All the flight control topics have message type `sensor_msgs/Joy`. Among the flight control topics, the `dji_sdk/flight_control_setpoint_generic` is the most general one and requires the user to provide the control flag which dictates how the inputs are interpreted by the flight controller, while the rest are wrappers for the convenience of users and don't need the flag. All supported flags are listed in `dji_sdk.h`.

`/dji_sdk/flight_control_setpoint_generic` (`sensor_msgs/Joy`)

- General setpoint where `axes[0]` to `axes[3]` stores set-point data for the 2 horizontal channels, the vertical channel, and the yaw channel, respectively. The meaning of the set-point data will be interpreted based on the control flag which is stored in `axes[4]`.

`/dji_sdk/flight_control_setpoint_ENUposition_yaw` (`sensor_msgs/Joy`)

- Command the X, Y position offset, Z position (height) and yaw angle in ENU ground frame.

`/dji_sdk/flight_control_setpoint_ENUvelocity_yawrate` (`sensor_msgs/Joy`)

- Command the X, Y, Z velocity in ENU ground frame, and yaw rate.

⁷ The Wiki site of the DJI_SDK is http://wiki.ros.org/dji_sdk.

/dji_sdk/flight_control_setpoint_rollpitch_yawrate_zposition
(sensor_msgs/Joy)

- Command the roll pitch angle, height, and yaw rate.

2. Gimbal Control

The subscriber that takes the input for Gimbal arm controls.

/dji_sdk/gimbal_angle_cmd (dji_sdk/Gimbal)

- Gimbal control command: Controls the Gimbal roll pitch and yaw angles (unit: 0.1 deg). mode: 0 - incremental control, the angle reference is the current Gimbal location. 1 - absolute control, the angle reference is related to configuration in DJI Go App.

/dji_sdk/gimbal_speed_cmd (geometry_msgs/Vector3Stamped)

- Gimbal speed command: Controls the Gimbal rate of change for roll pitch and yaw angles (unit: 0.1 deg/sec).

C. PUBLISHED TOPICS

/dji_sdk/attitude ([geometry_msgs/QuaternionStamped](#))

- Vehicle attitude represented as quaternion for the rotation from FLU body frame to ENU ground frame, published at 100 Hz.

/dji_sdk/battery_state ([sensor_msgs/BatteryState](#))

- Report the current battery voltage at 10 Hz.

/dji_sdk/flight_status ([std_msgs/UInt8](#))

- Simple status of the vehicle published at 50 Hz, detailed status is listed in dji_sdk.h. Note that status for M100 and A3/N3 are different.

/dji_sdk/from_mobile_data ([std_msgs/UInt8\[\]](#))

- Data received from mobile device to the vehicle.

/dji_sdk/gimbal_angle ([geometry_msgs/Vector3Stamped](#))

- Current gimbal joint angles, published at 50 Hz. If no gimbal present, default publishes all zeros.

/dji_sdk/gps_health ([std_msgs/UInt8](#))

- GPS signal health is between 0 and 5, 5 is the best condition. Use gps_position for control only if gps_health >= 3. Published at 50 Hz.

/dji_sdk/gps_position ([sensor_msgs/NavSatFix](#))

- Fused global position of the vehicle in latitude, longitude, and altitude(m). Position in WGS 84 reference ellipsoid, published at 50 Hz. If no gps present, default publishes longitude and latitude equal zeros.

/dji_sdk/imu ([sensor_msgs/Imu](#))

- IMU data including raw gyro reading in FLU body frame, raw accelerometer reading in FLU body frame, and attitude estimation, published at **100 Hz for M100, and 400 Hz for other platforms**. Note that raw accelerometer reading will give a Z direction 9.8 m/s^2 when the drone is put on a level ground statically.

/dji_sdk/rc ([sensor_msgs/Joy](#))

- Reading of the 6 channels of the remote controller, published at 50 Hz.

sensor_msgs/Joy	description (LB2, M100)	description (SBUS)	Range (LB2)	Range (SBUS)	Range (M100)
axes[0]	Roll Channel	Channel A	-1 to +1	-1 to +1	-1 to +1
axes[1]	Pitch Channel	Channel E	-1 to +1	-1 to +1	-1 to +1
axes[2]	Yaw Channel	Channel R	-1 to +1	-1 to +1	-1 to +1
axes[3]	Throttle Channel	Channel T	-1 to +1	-1 to +1	-1 to +1
axes[4]	Mode switch	Channel U	-10000, 0, 10000	-10000, 0, 10000	-8000, 0, 8000
axes[5]	Landing gear (H) switch	Channel Gear	-5000, -10000	-10000, 10000	-4545, -10000
Known bug	For SBUS controllers, the gear output depends on the channel mapping. Please refer to DJI Assistant 2 Remote controller settings.				

/dji_sdk/velocity ([geometry_msgs/Vector3Stamped](#))

- Velocity in ENU ground frame, published at 50 Hz. The velocity is valid only when `gps_health >= 3`.

/dji_sdk/height_above_takeoff ([std_msgs/Float32](#))

- Height above takeoff location. It is only valid after drone is armed, when the flight controller has a reference altitude set.

/dji_sdk/local_position ([geometry_msgs/PointStamped](#))

- Local position in Cartesian ENU frame, of which the origin is set by the user by calling the /dji_sdk/set_local_pos_ref service. Note that the local position is calculated from GPS position, so **good GPS health is needed** for the local position to be useful.

D. SERVICES

/dji_sdk/activation ([dji_sdk/Activation](#))

- The service to activate the drone with app ID and key pair. The activation arguments should be specified in launch files.
- Usage:

Response		
bool result	true--succeed	false--invalid action

/dji_sdk/camera_action ([dji_sdk/CameraAction](#))

- Take photo or video via service, return true if successful.
- Usage:

Request			
UInt8 Camera_action	0--Shoot Photo	1--Start video taking	2--Stop video taking
Response			
bool result	true--succeed	false--invalid action	

/dji_sdk/drone_arm_control ([dji_sdk/DroneArmControl](#))

- Enable or disable vehicle's arm motors.
- Usage:

Request		
uint8 arm	1--enable vehicle arm motor	else: disable arm motor
Response		
bool result	true--succeed	false--invalid action

/dji_sdk/drone_task_control ([dji_sdk/DroneTaskControl](#))

- Execute takeoff, landing or go home.
- Usage:

Request			
uint8 task	4--takeoff	6--landing	1--gohome
Response			
bool result	true--succeed	false--failed	

/dji_sdk/mission_hotpoint_action ([dji_sdk/MissionHpAction](#))

- Service that start/stop/pause/resume the hotspot mission.
- Usage:

Request				
uint8 action	0--start	1--stop	2--pause	3--resume
Response				
bool result	true--succeed	false--failed		

/dji_sdk/mission_hotpoint_getInfo ([dji_sdk/MissionHpGetInfo](#))

- Return the hotspot tasks info. Use `rosmmsg show dji_sdk/MissionHotpointTask` for more detail.
- Usage:

Response
MissionHotpointTask hotspot_task

/dji_sdk/mission_hotpoint_resetYaw ([dji_sdk/MissionHpResetYaw](#))

- Resets the Yaw position of the vehicle
- Usage:

Response		
bool result	true--succeed	false--failed

/dji_sdk/mission_hotpoint_updateRadius (dji_sdk/[MissionHpUpdateRadius](#))

- Update the radius of the hot point mission
- Usage:

Request		
float32 radius		
Response		
bool result	true--succeed	false--failed

/dji_sdk/mission_hotpoint_updateYawRate (dji_sdk/[MissionHpUpdateYawRate](#))

- Update the rate of change for Yaw and the direction of the change.
- Usage:

Request		
float32 yaw_rate		
uint8 direction		
Response		
bool result	true--succeed	false--failed

/dji_sdk/mission_hotpoint_upload (dji_sdk/[MissionHpUpload](#))

- Upload a set of hotpoint tasks to the vehicle. Use rosmmsg show dji_sdk/MissionHotpointTask for more detail
- Usage:

Request		
MissionHotpointTask hotpoint_task		
Response		
bool result	true--succeed	false--failed

/dji_sdk/mission_waypoint_action ([dji_sdk/MissionWpAction](#))

- Start/stop/pause/resume waypoint action.
- Usage:

Request				
uint8 action	0--start	1--stop	2--pause	3--resume
Response				
bool result	true--succeed	false--failed		

/dji_sdk/mission_waypoint_getInfo ([dji_sdk/MissionWpGetInfo](#))

- Get the current waypoint tasks. Use `rosmmsg show dji_sdk/MissionWaypointTask` for more detail.
- Usage:

Response
<code>MissionWaypointTask</code> waypoint_task

/dji_sdk/mission_waypoint_getSpeed ([dji_sdk/MissionWpGetSpeed](#))

- Return the waypoint velocity.
- Usage:

Response
float32 speed

/dji_sdk/mission_waypoint_setSpeed ([dji_sdk/MissionWpSetSpeed](#))

- Set the waypoint velocity.
- Usage:

Request		
float32 speed		
Response		
bool result	true--succeed	false--failed

/dji_sdk/mission_waypoint_upload ([dji_sdk/MissionWpUpload](#))

- Upload a new waypoint task, return true if successful. Use rosmmsg show dji_sdk/MissionWaypointTask for more detail.
- Usage:

Request		
MissionWaypointTask waypoint_task		
Response		
bool result	true--succeed	false--failed

/dji_sdk/sdk_control_authority ([dji_sdk/SDKControlAuthority](#))

- Request/release the control authority.
- Usage:

Request		
uint8 control_enable	1--request control	0--release control
Response		
bool result	true--succeed	false--failed

/dji_sdk/send_data_to_mobile ([dji_sdk/SendMobileData](#))

- Send data to the mobile side. The length of the data is upper-limited to 100.
- Usage:

Request		
uint8[] data	length(data) <= 100	
Response		
bool result	true--succeed	false--failed

/dji_sdk/query_drone_version ([dji_sdk/QueryDroneVersion](#))

- Query drone firmware version. Available version list can be found in dji_sdk.h

/dji_sdk/set_local_pos_ref ([dji_sdk/SetLocalPosRef](#))

- Set the origin of the local position to be the current GPS coordinate. Fail if GPS health is low (≤ 3).

E. PARAMETERS

~/dji_sdk/app_id (int)

- You must register as a developer with DJI and create an onboard SDK application ID and Key pair. Please go to <https://developer.dji.com/register/> to complete registration.

~/dji_sdk/ baud_rate (int, default: 921600)

- Baud rate should be set to match that is displayed in DJI Assistant 2 SDK settings.

~/dji_sdk/enc_key (string, default: Enter your enc key here)

- You must register as a developer with DJI and create an onboard SDK application ID and Key pair. Please go to <https://developer.dji.com/register/> to complete registration.

~/dji_sdk/serial_name (string, default: /dev/ttyUSB0)

- The serial port name that the USB is connected with. Candidates can be /dev/ttyUSBx, /dev/ttyTHSx, etc.

~/dji_sdk/use_broadcast (, default: false)

- Choose to use subscription (supported only on A3/N3) or broadcast method (supported by both M100 and A3/N3) for accessing telemetry data.

F. DETAILS ON FLIGHT CONTROL SETPOINT

All the above flight control topics take setpoint values of the X, Y, Z, and Yaw channels in axes[0]-axes[3]. In addition, the /dji_sdk/flight_control_setpoint_generic topic requires a control **flag** as axes[4] of the input. The control flag is an UInt8 variable that dictates how the inputs are interpreted by the flight controller. It is the bitwise OR of 5 separate flags defined as enums in dji_sdk.h, including Horizontal, Vertical, Yaw, Coordinate Frame, and the Breaking Mode.

Horizontal	description	reference	limit
0x00	Command roll and pitch angle	Ground/Body	0.611 rad (35 degree)

0x40	Command horizontal velocities	Ground/Body	30 m/s
0x80	Command position offsets	Ground/Body	N/A
0xC0	Command angular rates	Ground/Body	$5/6\pi$ rad/s
Vertical	description	reference	limit
0x00	Command the vertical speed	Ground	-5 to 5 m/s
0x10	Command altitude	Ground	0 to 120 m
0x20	Command thrust	Body	0% to 100%

Yaw	description	reference	limit
0x00	Command yaw angle	Ground	$-\pi$ to π
0x08	Command yaw rate	Ground	$5/6\pi$ rad/s
Coordinate	description		
0x00	Horizontal command is ground_ENU frame		
0x02	Horizontal command is body_FLU frame		

Active Break	description
0x00	No active break
0x01	Actively break to hold position after stop sending setpoint

APPENDIX D. DJI MANIFOLD CAMERA

This appendix presents the setup and coding of the `dji_cam_transport` that is utilized by the Manifold to decode and convert the image data for the CV algorithm application. The procedures for setup and execution of the `dji_cam_transport` are as follows:

A. BUILDING THE DJI_CAM_TRANSPORT

1. Create a catkin workspace to store the coding using the following command:

```
mkdir zenmuse_ws  
cd zenmuse_ws  
mkdir src  
catkin_init_workspace
```

2. Clone the `dji_cam_transport` from Github using the following command.

```
git clone https://github.com/guanfuchen/dji_cam_transport
```

3. Ensure that the following software package is installed in the Manifold system: `roscpp`, `rospy`, `std_msgs`, `image_transport`, `cv_bridge`
4. Execute the following commands to build the `dji_cam_transport`:

```
cd ~/zenmuse_ws  
catkin_make
```

B. RUNNING THE DJI_CAM_TRANSPORT

5. In order to execute the software application, ROS sever must be launched with the following command.

```
roscore
```

6. Next, provide root access rights for the `dji_cam_transport` using the following command:

```
sudo su
cd zenmuse_ws
source devel/setup.bash
```

7. Execute the following commands to run the dji_cam_transport:

```
roslaunch dji_cam_transport [-dgt]
```

8. The usage of `-dgt` depends on the required ROS image interface where `-d` is display video stream, `-g` is store video into NV12 memory, and `-t` is transfer video data to N1 flight controller. The `d` and `g` attributes cannot be used at the same time.
9. If data is placed into the NV12 memory data stream, the following commands will enable the image to be displayed on the ROS image viewer.

```
roslaunch image_view image:=/camera/image
```

C. CODES FOR DJI_CAM_TRANSPORT

The code is adapted from DJI's Manifold_Cam⁸ and Guan's DJI_Cam_Transport⁹ GitHub source code repository. The coding for the dji_cam_transport is as follows:

```
1. #include < stdio.h >
2. #include < setjmp.h >
3. #include < stdlib.h >
4. #include < malloc.h >
5. #include < string.h >
6. #include < unistd.h >
7. #include < signal.h >
8. #include < assert.h >
9. #include < sys / types.h >
10. #include < sys / stat.h >
11. #include < unistd.h >
12. #include < getopt.h >
13. #include < pthread.h >
14. #include < string.h >
15. #include < stdio.h >
16. #include < stdlib.h >
17. #include < unistd.h >
```

⁸ The GitHub repository for Manifold_Cam is available at <https://github.com/dji-sdk/Manifold-Cam>.

⁹ The GitHub repository for DJI_Cam_Transport is available at https://github.com/guanfuchen/dji_cam_transport.

```

18.  #include < errno.h >
19.  #include < ctype.h >
20.  #include "djicam.h"
21.  #include < opencv / cv.h >
22.  #include < opencv2 / core / core.hpp >
23.  #include < opencv2 / highgui / highgui.hpp >
24.  #include < opencv2 / imgproc / imgproc.hpp >
25.  #include < ros / ros.h >
26.  #include < cv_bridge / cv_bridge.h >
27.  #include < image_transport / image_transport.h >
28.
29.  using namespace cv;
30.
31.  #define FRAME_SIZE(1280 * 720 * 3 / 2) /*format NV12*/
32.  # define BLOCK_MODE 1
33.
34.  static unsigned char buffer[FRAME_SIZE] = {0};
35.  static unsigned int nframe = 0;
36.  static int mode = 0;
37.  //ros::NodeHandle nh;
38.  image_transport::Publisher pub;
39.  sensor_msgs::ImagePtr msg;
40.  //ros::Rate loop_rate(5);
41.
42.  IplImage * rgb = cvCreateImage(cvSize(1280, 720), IPL_DEPTH_8U, 3);
43.  IplImage * src = cvCreateImage(cvSize(1280, 1080), IPL_DEPTH_8U, 1);
44.
45.  static void print_usage(const char * prog)
46.  {
47.      printf("Usage: sudo %s [-dgt]\n", prog);
48.      puts("  -d --display      display vedio stream\n"
49.        "  -g --getbuffer    get NV12 format buffer\n"
50.        "  -t --transfer     transfer vedio datas to RC\n"
51.        "  Note: -d and -g cannot be set at the same time\n");
52.  }
53.
54.  static void parse_opts(int argc, char * * argv)
55.  {
56.      int c;
57.      static
58.      const struct option lopts[] =
59.      {
60.          {"display," 0, 0, 'd'},
61.          {"getbuffer," 0, 0, 'g'},
62.          {"transfer," 0, 0, 't'},
63.          {NULL, 0, 0, 0},
64.      };
65.
66.      while ((c = getopt_long(argc, argv, "dgt", lopts, NULL)) != -1)
67.      {
68.          switch (c)
69.          {
70.              case 'd':
71.                  mode |= DISPLAY_MODE;
72.                  break;
73.              case 'g':
74.                  mode |= GETBUFFER_MODE;
75.                  break;
76.              case 't':
77.                  mode |= TRANSFER_MODE;

```

```

78.             break;
79.         default:
80.             print_usage(argv[0]);
81.             exit(0);
82.     }
83. }
84. }
85.
86. static void * get_images_loop(void * data)
87. {
88.     int ret;
89.     while (!manifold_cam_exit()) /*Ctrl+c to break out*/
90.     {
91.         if (mode & GETBUFFER_MODE)
92.         {
93.             #if BLOCK_MODE ret = manifold_cam_read(buffer, & nframe, CAM_BLOCK);
94.             /*blocking read*/
95.             if (ret < 0)
96.             {
97.                 printf("manifold_cam_read error \n");
98.                 break;
99.             } else {
100.                 cvSetData(src, buffer, src -> widthStep);
101.                 cvCvtColor(src, rgb, CV_YUV2BGR_NV12);
102.                 msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8," rgb).toImageMsg();
103.                 for (int i = 0; i < 3; ++i) {
104.                     pub.publish(msg);
105.                     ros::Duration(0.1).sleep();
106.                 }
107.             }
108.         #else
109.             ret = manifold_cam_read(buffer, & nframe, CAM_NON_BLOCK); /*non_blocking read*/
110.             if (ret < 0) {
111.                 printf("manifold_cam_read error \n");
112.                 break;
113.             }
114.         #endif
115.     }
116.     usleep(1000);
117. }
118. printf("get_images_loop thread exit! \n");
119. }
120.
121. int main(int argc, char * * argv)
122. {
123.     ros::init(argc, argv, "image_publisher");
124.     ros::NodeHandle nh;
125.     image_transport::ImageTransport it(nh);
126.     pub = it.advertise("camera/image," 1);
127.     int ret;
128.     pthread_attr_t attr;
129.     struct sched_param schedparam;
130.     pthread_t read_thread;
131.
132.     if (0 != geteuid())
133.     {
134.         printf("Please run ./test as root!\n");
135.         print_usage(argv[0]);
136.         return -1;

```

```

137.     }
138.
139.     parse_opts(argc, argv); /*get parameters*/
140.     if (0 == mode || 3 == mode || 7 == mode) /*invalid mode*/
141.     {
142.         print_usage(argv[0]);
143.         return -1;
144.     }
145.
146.     ret = manifold_cam_init(mode);
147.     if (-1 == ret)
148.     {
149.         printf("manifold init error \n");
150.         return -1;
151.     }
152.
153.     /*
154.      * if the cpu usage is high, the scheduling policy of the read thread
155.      * is recommended setting to FIFO, and also, the priority of the thread
156.      * should be high enough.
157.      */
158.
159.     pthread_attr_init( & attr);
160.     pthread_attr_setinheritsched( & attr, PTHREAD_EXPLICIT_SCHED);
161.     pthread_attr_setschedpolicy((pthread_attr_t *) & attr, SCHED_FIFO);
162.     schedparam.sched_priority = 90;
163.     pthread_attr_setschedparam( & attr, & schedparam);
164.     pthread_attr_setscope( & attr, PTHREAD_SCOPE_SYSTEM);
165.
166.     if (pthread_create( & read_thread, & attr, get_images_loop, NULL) != 0)
167.     {
168.         perror("usbRead_thread create");
169.         assert(0);
170.     }
171.
172.     if (pthread_attr_destroy( & attr) != 0)
173.     {
174.         perror("pthread_attr_destroy error");
175.     }
176.     pthread_join(read_thread, NULL); /*wait for read_thread exit*/
177.
178.     while (!manifold_cam_exit())
179.     {
180.         sleep(1);
181.     }
182.     return 0;
183. }

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. YOLO

This appendix presents the setup procedures for YOLOv3, which provided the object detection and tracking capability to the UAV system. The setup procedures are extracted from the YOLO website.¹⁰ The procedures for the setup of the YOLOv3 are as follows:

A. INSTALLING THE BASE SYSTEM

1. Clone the Darknet(YOLOv3) from Github.¹¹
2. Open the terminal and locate the Darknet folder.
3. Execute the following commands to build the Darknet:

```
git clone https://github.com/pjreddie/darknet.git
cd darknet
make
```

B. COMPILING WITH CUDA AND OPENCV

4. Although Darknet is relatively fast on a CPU, the speed of the algorithm is enhanced through the use of GPU processing on the Nvidia processor. Follow the guide to install CUDA on the Nvidia chipset.
5. Change the first line of the Makefile in the base directory to:

```
GPU = 1
```

6. Similarly, the OpenCV compilation enabled support for various image/video formats. Follow the guide to install OpenCV on the Nvidia chipset.
7. Change the second line of the Makefile in the base directory to:

```
OPENCV = 1
```

¹⁰ Information on YOLO is found at <https://pjreddie.com/darknet/yolo/>.

¹¹ The GitHub repository for Darknet is available at <https://github.com/pjreddie/darknet.git>.

8. Recompile the YOLO algorithm using the following command:

```
make
```

C. SETUP OF PRE-TRAINED MODEL INTO ALGORITHM

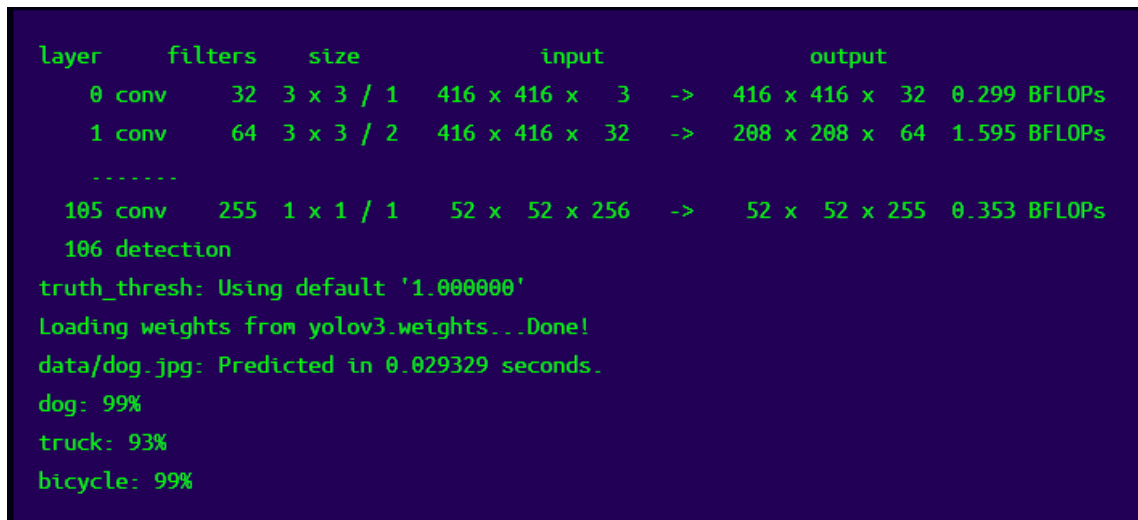
9. For quick algorithm testing, download the following pre-trained model (weights) into the Darknet folder:

```
cd ~/Darknet
wget https://pjreddie.com/media/files/yolov3.weights
wget https://pjreddie.com/media/files/yolov3-tiny.weights
wget https://pjreddie.com/media/files/yolov2.weights
wget https://pjreddie.com/media/files/yolov2-tiny.weights
```

10. To perform object detection on a sample image, use the following commands:

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

11. If the compilation of the algorithm was done correctly, the screen capture in the following figure will be displayed from the payload hardware:



```
layer    filters  size      input           output
   0 conv    32  3 x 3 / 1  416 x 416 x  3  ->  416 x 416 x  32  0.299 BFLOPs
   1 conv    64  3 x 3 / 2  416 x 416 x 32  ->  208 x 208 x  64  1.595 BFLOPs
-----
 105 conv   255  1 x 1 / 1   52 x  52 x 256  ->   52 x  52 x 255  0.353 BFLOPs
 106 detection
truth_thresh: Using default '1.000000'
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 0.029329 seconds.
dog: 99%
truck: 93%
bicycle: 99%
```

Figure 69. Output Data from the Algorithm

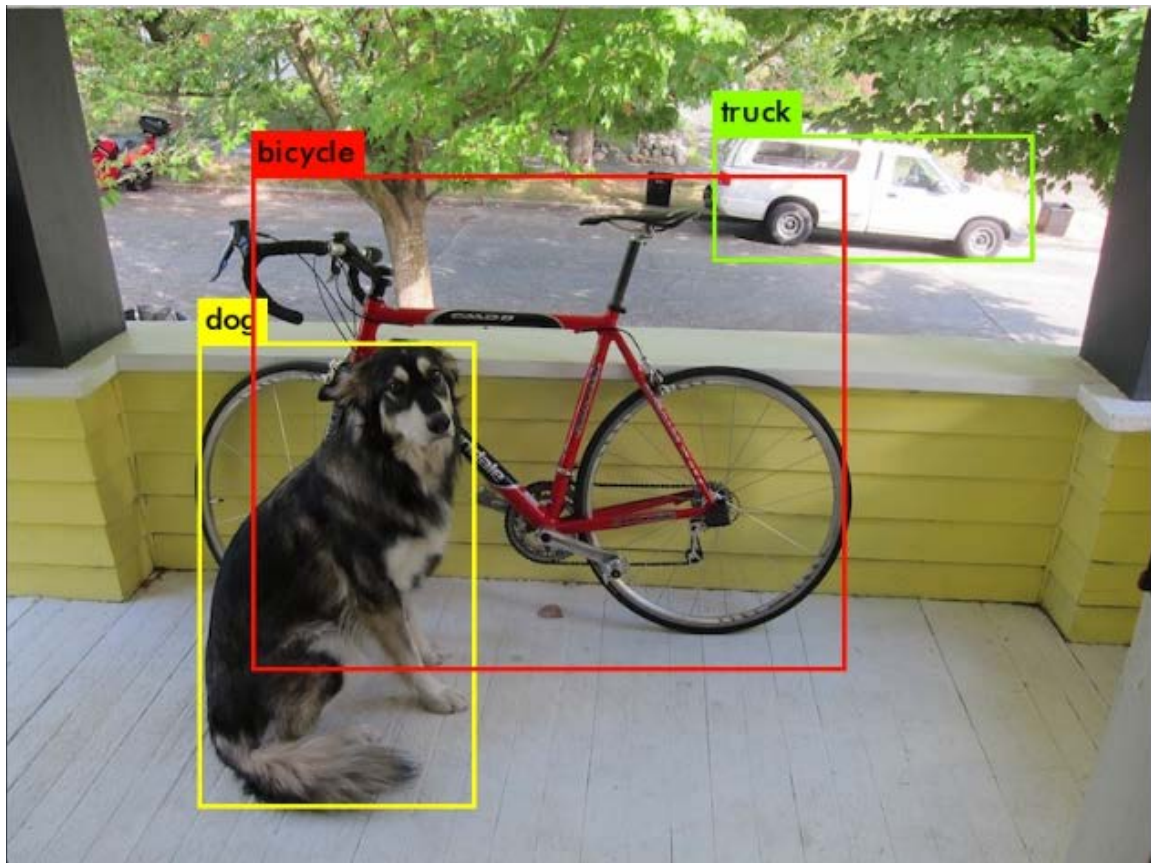


Figure 70. Prediction Data of the Sample Image. Source: Redmon et al. (2016).

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Adler, David. 2016. "Rolling Shutter versus Global Shutter." B&H. Accessed May 25, 2018. <https://www.bhphotovideo.com/explora/video/tips-and-solutions/rolling-shutter-versus-global-shutter>.
- Ang, Wee Kiong. 2017. "Assessment of an Onboard EO Sensor to Enable Detect and Sense Capability for UAVs Operating in a Cluttered Environment." Master's thesis, Naval Postgraduate School. <https://calhoun.nps.edu/handle/10945/56165>
- Beery, Paul T. 2017. "DoD Architecture Framework." Class notes for SE4150: System Architecting and Design, Naval Postgraduate School, Monterey, CA.
- Berkeley Design Technology, Inc. 2014. "NVIDIA's Jetson TK1 Evaluation Board Gets Embedded Vision." April 29, 2014. <https://www.bdti.com/InsideDSP/2014/04/29/NVIDIA>.
- Brox, Thomas, and Jitendra Malik. 2011. "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, no. 3 (2011): 500–513. <https://doi.org/10.1109/TPAMI.2010.143>.
- Datalight. 2018. "What is eMMC?." Accessed June 15, 2018. <https://www.datalight.com/solutions/technologies/emmc/what-is-emmc>.
- Dertat, Arden. 2017. "Applied Deep Learning - Part 4: Convolutional Neural Networks." Towards Data Science. November 8, 2017. <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>.
- DJI. n.d. "DJI Manifold." Accessed June 15, 2018. <https://www.dji.com/manifold>.
- . 2017. "Matrice 100 UAV Specifications." Accessed April 20, 2018. <https://www.dji.com/matrice100>.
- eLinux. 2016. "Jetson TK1 Power." May 16, 2016. https://elinux.org/Jetson/Jetson_TK1_Power.
- Fergus, Rob. 2015. "Neural Networks." Materials for Machine Learning Summer School 2015, Max Planck Institute for Intelligent Systems, Tübingen, Germany. http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf.
- Gandhi, Rohith. 2017. "Support Vector Machine—Introduction to Machine Learning Algorithms." Towards Data Science. June 7, 2018. <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>.

- . 2017. “Understanding Object Detection Algorithms.” Towards Data Science. July 9, 2018. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- Geitgey, Adam. 2016. “Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks.” Medium. June 13, 2016. <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721>.
- Girshick, Ross. 2015. “Fast R-CNN.” In *Proceedings of the 2015 IEEE International Conference on Computer Vision*: 1440–1448. <http://doi.org/10.1109/ICCV.2015.169>.
- Girshick, Ross, Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Sergio Guadarrame, and Trevor Darrell. 2014. “Caffe: Convolutional Architecture for Fast Feature Embedding.” In *Proceedings of the 22nd ACM International Conference on Multimedia*: 675 – 678. <https://arxiv.org/abs/1408.5093>.
- Hardkernel. 2014. “Odroid-XU4.” Accessed April 20, 2018. https://www.hardkernel.com/main/products/prdt_info.php.
- Harney, Robert C. 2013. “Combat Systems Engineering - Volume 1: Sensor Signals and Functions.” Class Textbook for SE3112: Combat System Engineering, Naval Postgraduate School, Monterey, CA.
- Jonen. 2007. “Jamtlands Flyg EC120B Colibri.” Image Media. <https://commons.wikimedia.org/w/index.php?curid=6039822>.
- Karn, Ujjwal. 2016. “An Intuitive Explanation of Convolutional Neural Networks.” August 11, 2016. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>.
- Krizhevsky, Alex. 2010. “Convolutional Deep Belief Networks on CIFAR-10.” Unpublished manuscript.
- Lappenküper, Dominik. 2018. *Global Shutter, Rolling Shutter - Functionality and Characteristics of Two Exposure Methods (Shutter Variants)*. White Paper. Ahrensburg, Germany. https://www.baslerweb.com/fp-1528105088/media/en/downloads/documents/white_papers/BAS1401_White_Paper_Rolling_Shutter.pdf.
- Li, Fei-Fei, Justin Johnson, and Serena Yeung. 2017. “Neural Networks Part 1: Setting up the Architecture.” Class Materials for CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, Palo Alto, CA. <http://cs231n.github.io/neural-networks-1/>.

- Li, Jing, Dong Hye Ye, Timothy Chung, Mathias Kolsch, Juan Wachs, and Charles Bouman. 2016. "Multi-Target Detection and Tracking from a Single Camera in Unmanned Aerial Vehicles (UAVs)." *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*: 4992–97. New York: Institute of Electrical Electronics Engineers. doi:10.1109/IROS.2016.7759733.
- Logitech. 2016. "HD Pro Webcam C920." Accessed April 20, 2018. http://support.logitech.com/en_us/product/hd-pro-webcam-c920/specs.
- Muellar, Matthias, Neil Smith, and Bernard Ghanem. 2016. "A Benchmark and Simulator for UAV Tracking." In *Proceedings of 14th European Conference on Computer Vision*: 445–461. http://doi.org/10.1007/978-3-319-46448-0_27.
- Mueller, Markus S., and Boris Jutzi. 2018. "UAS Navigation with SqueezePoseNet - Accuracy Boosting for Pose Regression by Data Augmentation." In *Drones 2018 Volume 2*. <http://dx.doi.org/10.3390/drones2010007>.
- Norris, Jim. 2014. "Android Processors: The Past, Present and Future of Smartphone Chip Design." Greendot. February 13, 2014. <https://www.greenbot.com/article/2095485/android-processors-the-past-present-and-future-of-smartphone-chip-design.html>.
- Nvidia. 2014. *Nvidia Tegra K1 - A New Era in Mobile Computing*. Technical Notes. Nvidia Corporation, Santa Clara, California. http://www.nvidia.com/content/PDF/tegra_white_papers/Tegra_K1_whitepaper_v1.0.pdf.
- Office of the Secretary of Defense. 2005. *Unmanned Aircraft Systems Roadmap, 2005 – 2030*. Washington, DC: U.S. Department of Defense. <http://www.dtic.mil/dtic/tr/fulltext/u2/a445081.pdf>.
- . 2013. *Unmanned Systems Integrated Roadmap, 2013–2038*. Washington, DC: U.S. Department of Defense. <http://www.dtic.mil/dtic/tr/fulltext/u2/a592015.pdf>.
- Oliver, Nuria, Barbara Rosario, and Alex Pentland. 1999. "A Bayesian Computer Vision System for Modeling Human Interactions." In *Proceedings of International Conference on Computer Vision Systems 1999*: 255–272. http://doi.org/10.1007/3-540-49256-9_16.
- Ouaknine, Arthur. 2018. "Review of Deep Learning Algorithm for Object Detection." Medium. February 5, 2018. <https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>.
- PIX4D. 2017. "Support - Starting a Project." Accessed December 6, 2017. <https://support.pix4d.com/hc/en-us/articles/202557409#gsc.tab=0>.

- Qimaging. 2014. *Rolling vs Global Shutter*. Technical Notes. Qimaging, British Columbia, Canada. <https://www.qimaging.com/ccdorscmos/pdfs/RollingvsGlobalShutter.pdf>.
- RED. 2018. “Global and Rolling Shutter.” Accessed May 15, 2018. <http://www.red.com/learn/red-101/global-rolling-shutter>.
- Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. “You Only Look Once: Unified, Real-Time Object Detection.” In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*: 779 – 788. <https://doi.org/10.1109/CVPR.2016.91>.
- Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” In *Proceeding of 2016 IEEE Transactions on Pattern Analysis and Machine Intelligence*: 1137–1149. <http://doi.org/10.1109/TPAMI.2016.2577031>.
- ROS Wiki. 2017. “Running the Flight Control Sample Code.” Accessed July 15, 2018. http://wiki.ros.org/dji_sdk/Tutorials/Running%20the%20flight%20control%20sample%20code.
- Samsung. 2014. *Mobile Processor Exynos 5 Octa (5422)*. Technical Notes. Samsung Electronics, South Korea. <https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/>.
- Schiesser, Tim. 2012. “Guide to Smartphone Hardware (3/7): Memory and Storage.” Neowin. March 12, 2012. <https://www.neowin.net/news/guide-to-smartphone-hardware-37-memory-and-storage>.
- Steele, Steve. 2016. “Integrating CPU and GPU. The ARM Methodology.” ARM. Accessed June 15, 2018. https://www.arm.com/files/event/Morning_3_Integrating_CPU_and_GPU_The_ARM_Methodology.pdf.
- Tirosh, Udi. 2012. “Everything You Wanted to Know about Rolling Shutter.” DIY-Photography. September 24, 2012. <https://www.diyphotography.net/everything-you-wanted-to-know-about-rolling-shutter/>.
- Tortonesi, M., A. Morelli, M. Govoni, J. Michaelis, C. Stefanelli, and S. Russell. 2017. “Leveraging Internet of Things within the Military Network Environment — Challenges and Solutions.” In *Proceedings of 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*: 111–116. <https://doi.org/10.1109/WF-IoT.2016.7845503>.
- Triggs, Robert. 2014. “SoC Showdown: Tegra K1 vs Exynos 7 Octa vs Snapdragon 805.” Android Authority. October 21, 2014. <https://www.androidauthority.com/tegra-k1-exynos-5433-snap-805-541582/>.

- Uijlings, Jasper RR, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. 2013. "Selective Search for Object Recognition." *International Journal of Computer Vision* 104, no. 2 (September): 154–171. <https://doi.org/10.1007/s11263-013-0620-5>.
- Weisstein, Eric W. 2018. "Convolution - Wolfram MathWorld." Mathworld. Accessed July 9, 2018. <http://mathworld.wolfram.com/Convolution.html>.
- Wilson, J. R. 2016. "EO/IR Sensors Boost Situational Awareness." *Military Aerospace*. January 19, 2016. <http://www.militaryaerospace.com/articles/print/volume-27/issue-1/special-report/eo-ir-sensors-boost-situational-awareness.html>.
- Xu, Joyce. 2017. "Deep Learning for Object Detection: A Comprehensive Review." *Towards Data Science*. September 11, 2017. <https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9>.
- Zhu, Pengfei, Longyin Wen, Xiao Biao, Haibin Ling, and Qinghua Hu. 2018. "Vision Meets Drones: A Challenge." *Cornell University Library*. Accessed July 12, 2018. <https://arxiv.org/abs/1804.07437v2>.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California