Optical Satellite/Component Tracking and Classification via Synthetic CNN Image Processing for Hardware-in-the-Loop testing and validation of Space Applications using free flying drone platforms

Marco A. Peterson

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Aerospace Engineering

Jonathan T. Black, Chair

Scott L. England

Pat H. Artis

Ryan K. Williams

March 22, 2022

Blacksburg, Virginia

Keywords: Computer Vision, Machine Learning, Neural networking, drones, mobile manipulation, optical simulation, synthetic data, virtual reality, Object tracking

Copyright 2022, Marco A. Peterson

Optical Satellite/Component Tracking and Classification via Synthetic CNN Image Processing for Hardware-in-the-Loop testing and validation of Space Applications using free flying drone platforms

Marco A. Peterson

ABSTRACT

The proliferation of reusable space vehicles has fundamentally changed how we inject assets into orbit and beyond, increasing the reliability and frequency of launches. Leading to the rapid development and adoption of new technologies into the Aerospace sector, such as computer vision (CV), machine learning (ML), and distributed networking. All these technologies are necessary to enable genuinely autonomous decision-making for space-borne platforms as our spacecraft travel further into the solar system, and our missions sets become more ambitious, requiring true "human out of the loop" solutions for a wide range of engineering and operational problem sets. Deployment of systems proficient at classifying, tracking, capturing, and ultimately manipulating orbital assets and components for maintenance and assembly in the persistent dynamic environment of space and on the surface of other celestial bodies, tasks commonly referred to as *On-Orbit Servicing* and *In Space Assembly*, have a unique automation potential. Given the inherent dangers of manned space flight/extravehicular activity (EVAs) methods currently employed to perform spacecraft construction and maintenance tasking, coupled with the current limitation of long-duration human flight outside of low earth orbit, space robotics armed with generalized sensing and control machine learning architectures is a tremendous enabling technology. However, the large amounts of sensor data required to adequately train neural networks for these space domain tasks are

either limited or non-existent, requiring alternate means of data collection/generation. Additionally, the wide-scale tools and methodologies required for hardware-in-the-loop simulation, testing, and validation of these new technologies outside of multimillion-dollar facilities are largely in their developmental stages. This dissertation proposes a novel approach for simulating space-based computer vision sensing and robotic control using both physical and virtual reality testing environments. This methodology is designed to both be affordable and expandable, enabling hardware-in-the-loop simulation and validation of space systems at large scale across multiple institutions. While the focus of the specific computer vision models in this paper are narrowly focused on solving imagery problems found on orbit, this work can be expanded to solve any problem set that requires robust onboard computer vision, robotic manipulation, and free flight capabilities.

Optical Satellite/Component Tracking and Classification via Synthetic CNN Image Processing for Hardware-in-the-Loop testing and validation of Space Applications using free flying drone platforms

Marco A. Peterson

GENERAL AUDIENCE ABSTRACT

The lack of real-world imagery of space assets and planetary surfaces required to train neural networks to autonomously identify, classify, and perform decision-making in these environments is either limited, none existent, or prohibitively expensive to obtain. Leveraging the power of the unreal engine, motion capture, and theatre projections technologies combined with robotics, computer vision, and machine learning to provide a means to recreate these worlds for the purpose of optical machine learning testing and validation for space and other celestial applications. This dissertation also incorporates domain randomization methods to increase neural network performance for the above mentioned applications, and apply them to hardware-in-the-loop (HIL) simulation for space environment visual testing and vision based control systems.

Dedicated to Virginia Tech.

Acknowledgments

Over the last five years, I have received a great deal of support from the students and staff at Virginia Tech. Firstly, I'd like to thank my advisor, Dr. Black, for accepting a computer science student into the aerospace curriculum. After a tumultuous first semester and a rapid education on new terminologies not found in the field of computer science, such as Young's modulus and relative motion, I am thankful for the opportunity to apply my skills to a new field I have always wanted to explore and contribute to. Second, I would like to acknowledge and thank the countless professors that have guided me through this journey. Dr. Williams, who formally introduced me to the world of robotics, Dr. England for showing me the minutiae of space travel and detailing all the ways it endeavors to cripple your spacecraft; and Dr. Artis for showing me the ins and outs of flight dynamics as well as describing the rigors of the test pilot school. I would also like to thank the Virginia Tech ICTAS (Institute for Critical Technology and Applied Science) department for accepting me into their institution as an ICTAS scholar and allowing me to freeze my academic progress during a 12-month deployment to Iraq. To my fellow graduate students, thank you for mentorship, leadership, and friendship over the last 5 years. Satyaki, you offered a crash course in mechanical engineering that was sorely needed during our first semester. Minzhen, You are an incredibly talented designer and engineer who got many of our projects out of a jam in the 11th hour. Thank you for all your help and expertise over the last two years. I would also like to acknowledge and thank Gustavo Gargioni. Your projects always shot for the Moon, and you executed your ideas with an enthusiasm that inspired not only me but the lab as a whole. Finally, I would like to thank my Company Command team. As an Army Reserve commander and aviator pursuing a Ph.D., my time is allocated to both

military and academic deadlines, with obligations often overlapping. The professionalism and expertise of the men and women of Delta Company 5-159th GSAB have never failed to deliver mission success, even during the dynamic and often constraining environment presented by the COVID-19 pandemic, enabling me to allocate time to academic deadlines when needed. 1SG Jamison, your unwavering dedication to the soldiers of Delta company and invaluable knowledge concerning all things aviation maintenance has proven instrumental in building and expanding the capabilities of the formation. I could not have asked for a better 1SG as a Commander. CW4 loving, as the company Executive Officer (XO), you have fulfilled every tasker thrown your way, many of which were due two hours after receiving the email. You are worth every dot on your chest and earned my unconditional trust serving as command authority when academic obligations pulled me away. SFC Mashaw, we served together on deployment, and you proved yourself as competent and capable back then. I was excited to learn you will be joining my team permanently in garrison. As one of the few full-time NCOs, your job has not been easy, and I appreciate your sted fast dedication. To all the platoon leaders, platoon sergeants, and NCOs of Delta Company, Thank you for making my transition to command seamless. You have been able to do more with less over the last 12 months. Without your efforts, this Ph.D. publication would have been a far less complete product.

Contents

List of Figures xii

List of Tables xviii

List of Abbreviations xix

1 Introduction 1

1.1 Motivation 1

1.2 Research Questions 5

1.3 Contributions 6

1.3.1 Synthetic Data sets 6

1.3.2 Unreal Engine Synthetic Orbital Environments 6

1.3.3 Domain Randomized Space Application Results 7

1.3.4 HIL Proof of Concept Results 7

1.3.5 SpaceDrones Robotic Platform 7

1.4 Summery of Chapters 8

2 Review of Current Literature 9

2.1 Simulation and Validation 9

2.1.1 Physical Simulators . . . 10
2.1.2 Surrogate Data Generation . . . 14
2.2 Today's State of the art CNNs . . . 17
2.3 Domain Randomization . . . 18
2.4 Computer Vision in the Space Industry . . . 20
2.5 Ground Truth Tracking . . . 23

3 Neural Network Modeling **25**

3.0.1 Activation Functions . . . 27
3.0.2 Pooling . . . 29
3.0.3 Fully Connected Layers . . . 30
3.0.4 Forward Propagation . . . 31
3.0.5 Back Propagation . . . 32

4 Materials and Methods **36**

4.1 CNN Architecture Selection . . . 36
4.2 Synthetic Reality creation and collection . . . 37
4.2.1 Synthetic Domain Randomization . . . 39
4.3 Porting CAD Models & Hardware into Synthetic Environments . . . 40
4.4 Methods of Labeling . . . 41
4.5 Division of Data sets . . . 42

4.6 Evaluation Metrics . . . 43
4.7 Flight Hardware . . . 44
4.8 SpaceDrones 2.0 Architecture . . . 47
4.8.1 Target Localization . . . 50
4.9 Drone Design . . . 52
4.9.1 Drone Body . . . 52
4.9.2 Drone Power Distribution . . . 53
4.9.3 Drone Reference Frames . . . 54
4.9.4 Drone Mounted Robotic Arm . . . 55
4.9.5 Drone Limitations . . . 58
4.9.6 Sensor Limitations . . . 59
4.10 Virginia Tech CUBE . . . 59

5 Results **62**

5.1 Synthetic Data vs Real World Results . . . 62
5.1.1 Domain Randomization Results . . . 66
5.1.2 Database Augmentations . . . 70
5.1.3 Computer Vision Results Summery . . . 72
5.1.4 CUBE Computer Vision Results . . . 73
5.1.5 Robotic Arm Capture Sequence . . . 75

6 Discussion **78**

6.1 Future Work 79

6.1.1 Orbital Dynamics Control Simulation 79

6.1.2 Deep Space Simulation Environments 81

6.1.3 Autonomous Path Planning 82

6.2 Domain Adaptation 83

6.2.1 Defective Manufacturing Computer Vision capability 84

6.2.2 Space Compute Hardware Needed 85

6.2.3 Effects of Ionizing Radiation on Neural Networks 87

7 Conclusions **89**

Bibliography **91**

Appendices **99**

Appendix A Neural Network Mathematical Diagram **100**

A.1 Weighted Sum Cost 100

A.2 Summed Cost 100

Appendix B System UML Diagram **103**

List of Figures

1.1 SpaceDrones optical visual simulation concept of operation. Synthetic CV [Top Left], Robotic CV and arm testing [Bottom Left], VT CUBE CV Synthetic HIL testing [Top Middle], VT CUBE drone flight [Bottom Middle], SpiderFab In-space assembly concept [6] [Far Right] 3

1.2 Synthetic Render of the James Webb Telescope (JWST) 4

2.1 Optical simulator and testbed for spacecraft star tracker development [14], Johns Hopkins University Applied Physics Laboratory 11

2.2 TRON Testbed located Stanfords The Space Rendezvous Laboratory (SLAB) [46] Facility 12

2.3 DLR's Docking simulation using Photonic Mixer Device Senors [33] 13

2.4 Optical Navigation via crater detection for lunar landers [56] 14

2.5 Pose Estimation from Photorealistic Rendering [59] 15

2.6 SpaceX Dragon Synthetic image rendering setup in Blender [17] 16

2.7 SurRender visual simulation demonstrating photometric accuracy [35] 17

2.8 Intersection over Union (IoU) bounding box threshold examples. 18

2.9 Synthetic Domain Randomization applied to self driving cars published by Nvidia [55] 19

2.10 Synthetic Data Domain Randomization (Unreal Engine 4) 20

2.11 Major Conference and Journal papers (2016 - 2020) detailing vision based solutions for space-based assembly, servicing, and manufacturing [45] 21

2.12 Major Conference and Journal papers (2016 - 2020) illustrating domain randomized computer vision solutions [45] 22

2.13 Astrobee (1) shown on a level granite surface (2), which is used to simulate a 2D microgravity environment. The prototype has trackers mounted to its port and starboard sides (3), and a single lighthouse (4) mounted overhead [15] 24

3.1 ReLu vs Leaky ReLu nonlinear activation functions 28

3.2 Example of Pooling methods 30

3.3 Example of a convolutional neural network (CNN) architecture. This particular architecture is a CNN known as ResNet [31] 31

3.4 Simple single neuron back propagation diagram. 34

4.1 Simplified Neural Network architecture diagram. 37

4.2 Example Orbital Images Generated using Unreal Engine 4 38

4.3 Example Domain Randomization Images Generated using Unreal Engine 4 Video illustration 40

4.4 Project work flow when creating 3D objects for both synthetic environments and 3D printing 41

4.5 Ground truth vs in frame Labeling Problem 42

4.6 Division of imagery data into specific data sets 43

4.7 Intersection over Union evaluation metric 45

4.8 Pixhawk to Raspberry PI 3 GPIO Pin out [27] 46

4.9 Pixhawk to Xavier NX GPIO Pin out 47

4.10 SpaceDrones Opti-Track tracking and control architecture 48

4.11 SpaceDrones VR tracking and control architecture 49

4.12 RGB Camera and LiDAR Point cloud integration senor data side-by-side . . 51

4.13 Porous structure LiDAR Depth Problem 52

4.14 5 meter LiDAR scan localization result when integrating Computer vision bounding box data with LiDAR point cloud data. 53

4.15 Drone frame with onboard tracking, computer vision, machine learning computation, LiDAR sencing, and VR tracking 54

4.16 Drone Camera frame offset 55

4.17 Drone Arm link measurements (mm) 56

4.18 Drone Arm Base Frame relative to other relevant world frames. Raw Links [Left, with arm model animated [Right] 57

4.19 Depiction of the integrated Drone, Camera, Robotic Arm System 59

4.20 RGB Computer Vision during sensor saturation events 60

4.21 onboard drone Computer Vision testing inside a physical/virtual simulation projection at Virginia Tech's CUBE facility [39] 61

5.1 Synthetic orbital data set self evaluation results showing relatively high mean average precision over 250 training epochs. Establishing a baseline performance metric. 63

5.2 Real World Truss Data Set self evaluation mean average precision (mAP) . 63

5.3 Real World Truss Data Set self evaluation F1 curve (left), confusion matrix (right) 64

5.4 Mean average precision result when training with synthetic imagery and testing against real world imagery 65

5.5 Non-Standard trusses used to evaluate neural network performance before and after introducing Domain Randomization 67

5.6 Domain Randomized Synthetic Data Set self evaluation F1 curve (left), confusion matrix (right) 68

5.7 mean average precision (mAP) result when training with Domain Randomized synthetic imagery and testing against real world imagery 68

5.8 Domain Randomized vs None Domain Randomized results 69

5.9 Mean average precision result when training 90% domain randomized synthetic imagery and 10% real world imagery, CNN evaluated against a real world data set 70

5.10 Synthetic Low light / supplemental lighting training images 71

5.11 Low light / supplemental lighting testing images 71

5.12 Overall summery of all major neural network performance evaluations across data sets. Synthetic Imagery can achieve results with in 11% mAP(0.50) of real world base lines, and with in 4% mAP(0.50:0.95) of the synthetic baseline when incorporating Domain Randomization and Augmenting with 10% real world data. 72

5.13 Drone camera POV (point of view) running computer vision on objects of interest against a synthetic orbital background, Video illustration 74

5.14 Virginia Tech CUBE Optical Space Simulation 75

5.15 Chronological illustration of drone pick sequence, Video illustration 76

6.1 Virginia Tech Omni-directional drone (Omni Copter) 80

6.2 Relative motion simulation using micro drones 80

6.3 Simulated Mars environment with virtual reality drone bodies performing autonomous tasking [Top]. Mars environment projection in the Virginia Tech Cube with real-world drone motioned being mirrored to a virtual drone vehicle [Bottom]. Video illustration 82

6.4 Vehicle Path Planning by University of Washington's Autonomous Control Laboratory [62], Video illustration 83

6.5 Initial Detection results additive Manufacturing imperfections using neural networking 85

6.6 Full vs Half Boom extension using Computer Vision to determine deployment failures on real world space flight hardware 86

6.7 Space grade radiation hardened computational hardware vs. traditional commercial-off-the-shelf computational hardware 87

7.1 Research Comparison with current literature. Tron Lab [25], DLR RPO [21], DLR ATON [56], United States Navy (USN) Pose Estimation, [17], Airbuse SurRender [35], Jet Propulsion Laboratory (JPL) Formation Control Testbed (FCT) [49], Domain Randomization (Tobin) [54], Domain Randomization (Nvidia)[55] 90

A.1 Weighted Sum mathematical Diagram 101

A.2 Summation Cost over L Layers 102

B.1 Drone Localization and Capture Software Diagram 103

List of Tables

5.1 Overall summery table of all major neural network (YOLOv5) performance evaluations across data sets. 73

5.2 Table of CUBE computer vision Results 75

List of Abbreviations

API Application Programming Interface

CV Computer Vision

IK Inverse Kinematics

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

ML Machine Learning

OTS Off the shelf

RGB Red-Green-Blue

TLR Technology Readiness Levels

VR Virtual Reality

Machine learning is the study of computer algorithms designed to incrementally improve automatically at a specific or generalized task through experience. This is usually done with the introduction of vast amounts of data into the algorithm. This discipline is seen as a branch of artificial intelligence.

Computer vision is an interdisciplinary scientific field designed to gain high level understanding of the world using image and video sensor data in an effort to replicate human visual understanding.

Virtual reality is a simulated experience created to closely replicate real world visual, audio, and locomotion sensory inputs.

Inverse Kinematics is the mathematical process of calculating the joint parameters (usually robotic arms or animation) needed to place the end of a kinematic chain (End effector) at a particular point in space, usually a robotic claw or other device designed to manipulate/interact with an environment or object.

APIs are a set of software functions or procedures specifically designed to enable the transfer of data between two or more software systems.

An electronic device that measures and outputs a body's orientation, specific force, angular rate, and sometimes position, using a combination of accelerometers, gyroscopes, and magnetometers.

Off-the-shelf means any product, system, or service that can be purchased on the open market or used with a proprietary license.

The mentioning of the RGB color standard usually means the presence of color or specifies the order of color parameters for a given senor or system.

Light Detection and Ranging is a point cloud sensor most often used to determine range or build a detailed map of a particular environment.

A methodology developed by NASA, Technology readiness levels are a method for estimating the maturity of technologies during the acquisition phase of a project or program.

Chapter 1

Introduction

1.1 Motivation

Manufacturing and maintaining economic, scientific, and defense interests in earth orbit and beyond will prove vital as both private and public ventures seek to discover and harness space's vast resources and unique properties. This emerging domain has several particular problem sets uniquely suited for autonomous systems due to the inherent dangers of space travel to manned crews. However, systems engineering outside the familiar confines of Earth has two significant hurdles. First, and perhaps the most obvious, is recreating the conditions in space such as microgravity, vacuum, and radiation is either impossible, or methods developed to do so are costly. The second is the validation of both hardware and software for space-based systems in these environmental conditions. NASA and many other space agencies have engineered physical facilities to both recreate orbital conditions and validate systems and procedures, such as NASA air bearing Facilities [49], Neutral Buoyancy Laboratory [32], and Space Environments Complex [42].

These facilities, summarized here [18], provide a proven "hardware-in-the-loop" (HIL) environment for system testing or training required for most NASA missions. However, these facilities are expensive to both build and maintain, requiring millions of dollars of capital investment. These are world-class facilities performing groundbreaking work, but as space assets become cheaper to deploy, including autonomous robotic capabilities, many organiza-

tions have looked at alternative means to both reliably recreate space conditions and create HIL testbeds to validate such systems. In terms of space/orbital simulation, many facilities including those listed above solve or attempt to solve one or more of the issues listed below.

- Magnetic Fields and Fluctuation
- Temperature
- Radiation Effects
- Time delay
- Air Density / Vacuum
- Relative Motion
- Optical Visual Sensing Guidance & Navigation

This is not an exhaustive list, but it demonstrates some of the most pressing problem sets when developing the means and methods to execute verification and validation (V and V) for space-borne systems. The simulations tasks mentioned above are active fields of research, where significant breakthroughs in any single method will enable substantial benefits to a broader range of research and improvements to current orbital capabilities. However, this dissertation will narrow down to the following four problem sets related to optical visual sensing for robotic manipulation: 1) Demonstrate the feasibility of simulated optical visual environments to supplement or completely replace real world orbital or deep space imagery for the purposes of training neural networks to perform automated tasks. 2) Evaluate the impact of domain randomization implementation data sets on neural network performance

on space-borne problem sets. 3) Deploy such a system onboard a free flying system performing onboard computer vision (CV) to provide a proof of concept for CV based real world robotic manipulation. 4) Lastly deploy this system within a theater projection space cable of projecting high fidelity environments to provide hardware-in-the-loop optical-visual flight capability to further expand V and V capabilities.

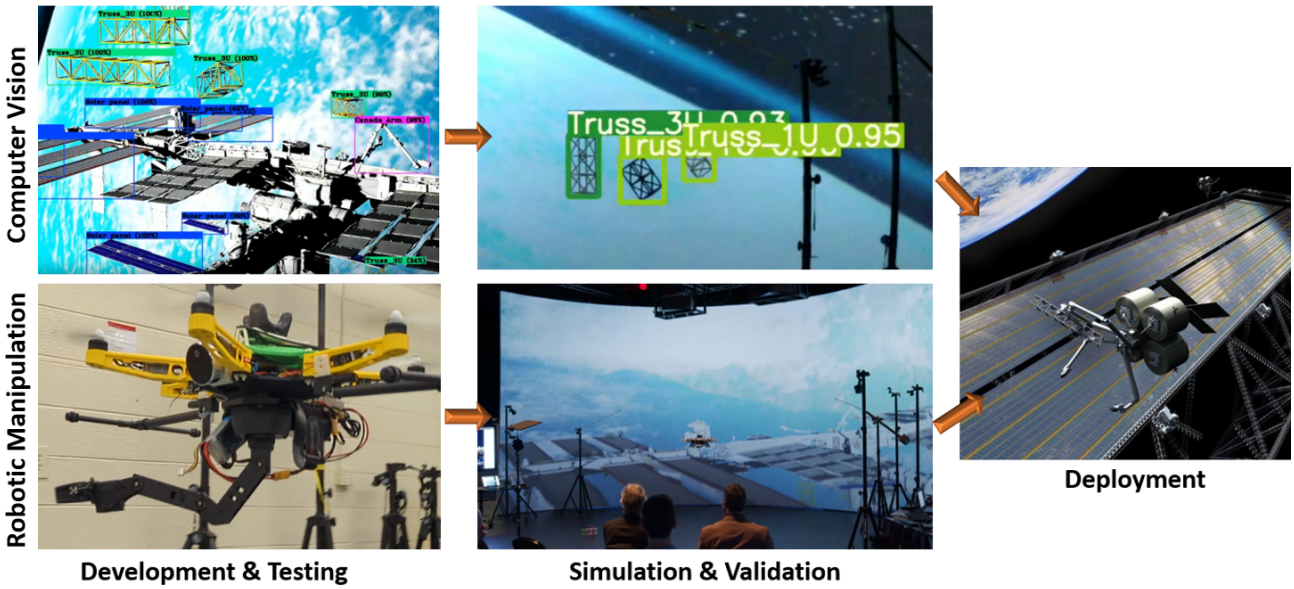

Figure 1.1: SpaceDrones optical visual simulation concept of operation. Synthetic CV [Top Left], Robotic CV and arm testing [Bottom Left], VT CUBE CV Synthetic HIL testing [Top Middle], VT CUBE drone flight [Bottom Middle], SpiderFab In-space assembly concept [6] [Far Right]

Perhaps the most pressing real-world example of this problem would be that of the newly launched James Webb Telescope (JWST), synthetically rendered in Figure 1.2. Unlike its predecessor, the Hubble Telescope, the James Webb's parking orbit at lagrange point two (L2) is substantially further away than Hubble's low earth orbit, thus making repair operations a far more challenging task, rendering a manned mission to the JWTS much less likely. In order to serve the JWTS and other deep space assets like it, the autonomous and semi-autonomous capabilities already described will need to be solved. With an estimated fuel reserve of about 20 years, James Webb's fuel to maintain such an orbit is not unlimited.

This constraint establishes a real-world timeline to develop, test, and deploy these systems to save a 10 Billion dollar asset, and other deep space assets.

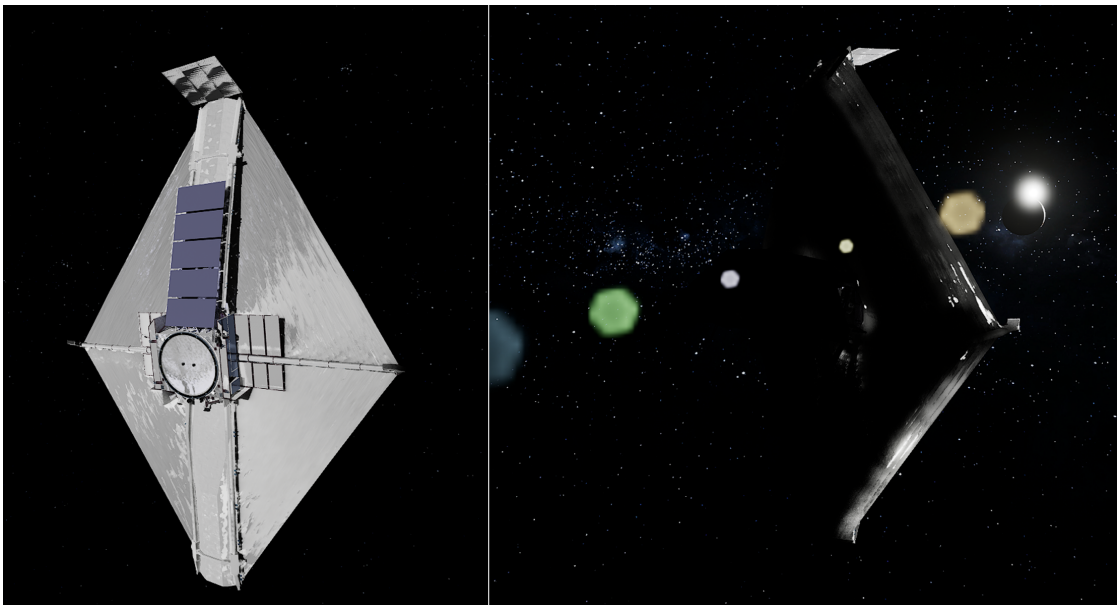

Figure 1.2: Synthetic Render of the James Webb Telescope (JWST)

To achieve a hardware-in-the-loop testing simulation for a maneuvering system platform and associated sensors, building and expanded upon a robotics architecture referred to as Space-Drones. The first iteration of the SpaceDrones Platform was constructed out of necessity to develop a free flying sensor platform in 2018 [27]. At its heart, this architecture consists of one or more maneuvering robotic platforms controlled by a central ground station via centralized networking or decentralized decision-making between robotic nodes. This framework enables sensor, position, and other applicable data transfer between systems. The robotic and simulation platform has since provided valuable flight and sensor data for a wide range of organizations and engineering teams on the Virginia Tech Campus [24] [40] [41]. To make the system cheaper and easier to deploy at other academic institutions across Virginia, the SpaceDrones architecture was updated to include Steam VR's [5] motion tracking API. Sacrificing sub-millimeter accuracy for a price reduction of several orders of magnitude, a

capability developed and deployed in 2019 [24]. This project endeavors to add a machine learning/computer vision capability to the SpaceDrones platform as a proof of concept for space domain applications and as a real-world flying 3D transnational and single rotational robotics motion solution capable of providing useful testbed capabilities; as well as a complete end to end solution that can be deployed to solve a wide range of applications requiring an autonomous drone technology with onboard computer vision capabilities.

Like all machine learning applications, space domain computer vision tasks will be dependent on vast amounts of imagery data. Given the inherent lack of space domain imagery data needed to perform functional robotic tasks in orbit and deep space, this project has to create these data sets from scratch. As of today, the only realistic means of generating the thousands of images needed is to use synthetic world creation tools such as Unreal Engine or Unity. Unlike many standardized data sets used in the industry to evaluate neural network performance in a relatively clean testing regime, a space-based neural network will need to handle significant variations in lighting, rotational orientation, or both. Data set parameters such as resolution, average image distribution, label distribution, and label noise will need to be considered and evaluated when creating brand new data sets. Lastly, I intend to leverage domain randomization and data augmentation to improve CNN (convolutional neural network) model performance to solve the high lighting, background, and pixels variations found in space-based imagery. To my knowledge, this will be the first piece of literature to introduce the research questions posed below.

1.2 Research Questions

The primary questions to provide a operational proof of concept for the problem sets listed above are illustrated below.

1. Can synthetic data be used for reliable real world object tracking?

2. Can Domain Randomization and data augmentation be used to increase cross domain computer vision performance?

3. Can synthetically trained computer vision models be used for hardware-in-the-loop testing and localization for real world applications?

1.3 Contributions

1.3.1 Synthetic Data sets

This project has compiled over 15,000 uniquely labeled images, highlighting space-borne objects of interest within synthetic and non-synthetic environments useful for orbital autonomous tasking such as solar panels, robotic arms, Spacecraft, and additive manufacturing construction parts. These meticulously curated images span a wide range of synthetic and real-world data sets, including domain randomized, non-domain randomized, data augmented, nighttime-daytime, as well as augmented hardware-in-the-loop data sets derived from the Virginia Tech Cube [39]. All of these data sets are open source and can be found at [43].

1.3.2 Unreal Engine Synthetic Orbital Environments

Second, in an effort to limit the barriers to entry to recreate this particular project and experiment. I have also provided all the unreal engine assets, settings, and plug-ins used; All of which can be found here [44] in a single zip file. All of the Unreal Engine parameters pertaining to the synthetic generation of domain randomized and none domain randomized

data sets were compiled and made publicly available provide a springboard for any future space domain machine learning and/or Artificial intelligence applications/experimentation detailed in this dissertation.

1.3.3 Domain Randomized Space Application Results

Thirdly, this project details the results and advantages of domain randomization for space based environments and use cases regarding sensor-based situational awareness using synthetically trained computer vision models for monocular computer vision object identification, classification, and tracking.

1.3.4 HIL Proof of Concept Results

Fourth, This dissertation combines all of the previously mentioned contributions into one seamless hardware-in-the-loop simulation and validation method for space based neural network and robotics testing, as well as future controls and path planning integration that will be layers on top of the foundation of this body of work.

1.3.5 SpaceDrones Robotic Platform

Lastly, This work demonstrates a complete end-to-end drone hardware-in-the-loop capability that enables algorithm and control testing for space domain applications. However, this architecture has a wide range of possible use cases. Any economic, academic, or public service application requiring an autonomous drone solution capable of onboard computer vision/machine learning and mobile manipulation will benefit from the methods detailed in this dissertation.

1.4 Summery of Chapters

Chapter 2 provides a detailed overview of current literature in the fields of space-borne simulation, validation, and verification; focusing on creating testing regimes, methodologies, and procedures designed to test space systems within the constraints of Earth. Given the inherent environmental differences, this chapter continues with a review of state-of-the-art literature on machine learning-based computer vision, domain randomization, and current computer vision applications in the space industry. Chapter 3 follows with a high-level overview of the architecture and mathematics governing convolutional neural networks so readers can better understand the results presented. Chapter 4 illustrates the materials and methodologies used in this dissertation. Methods include CNN architecture selection, synthetic data creation, workflows, evaluation metrics, drone design, object localization, and the use of Virginia Tech's CUBE facility. Chapter 5 details integrated computer vision and domain randomization results as well as object localization results using computer vision and LiDAR sensor integration. Both of these capabilities are then used to perform autonomous object capture tasking via a free-flying drone platform. Chapter 6 discusses future work and capabilities actively being developed for the project at the time of the publication. Finally, Chapter 7 concludes this dissertation.

Chapter 2

Review of Current Literature

This chapter illustrates current work in the fields of space-borne simulation using both physical mock-ups and virtual reality (otherwise known as Surrogate data generation) environments to create useful testbeds and methods for space application testing. It also details today's state-of-the-art convolutional neural networks (CNNs) being deployed for computer vision applications as well as current attempts to improve neural network performance via domain randomization. This chapter concludes with a brief overview of all previously mentioned topics and how they have been trending over the last 5 years when applied to the space industry.

2.1 Simulation and Validation

Traditional terrestrial optical-based sensing used to perform practical tasks such as classifying objects in a frame or extracting features has typically been accomplished using easily accessible environments where image data can be collected at relatively low cost and with relatively little noise. Examples of these applications include self-driving and manufacturing.

Conversely, obtaining usable image data and associated noise parameters for similar vision-based applications within the space domain is often a costly and time-consuming endeavor. Few missions have been launched for the expressed purpose of collecting direct imaging of spacecraft, space stations, or other space based assets needed to test and validate vision-

based tasks such as satellite position/orientation estimation or maneuver and manipulating tasks associated with future industry milestones like in-space assembly, on-orbit servicing, active space debris removal, and many more. A lack of adequate software/hardware-in-the-loop validation capability can cause unforeseen anomalies causing system degradation, like the optical blooming around the Tango spacecraft during the PRISMA mission [20], or in the worst case can result in mission failure as demonstrated by the DART mission's [12] collision between two spacecraft as a result of a software mishap. In an attempt to fill this gap in capability, several researchers have devised several alternative methods to which can be classified into two categories detailed in the subsections below.

2.1.1 Physical Simulators

Physical simulation methods use physical objects (often scale replicas) such as a satellite, spacecraft, or celestial bodies, and supporting light and background stages to recreate desired optical parameters. Simulation devices such as John Hopkins University's Celestial Object Simulator [14] in Figure 2.1 have proven crucial for the validation and development of vision-based star tracking systems, providing a testbed for new navigation sensors, designs, and tracking algorithms [50]. Incorporating a 4ft diameter acrylic hemispherical dome, designed to emulate the 100 brightest stars in the northern hemisphere using light-emitting diodes, with a control dynamic range of 16:1 and maximum optomechanical coupling error of 10%. Coupled with a rotating 2-axis gimbal, this test stand is capable of simulating and validating attitude motion controls.

Stanford's Space Rendezvous Lab's (SLAB) TRON (Testbed for Rendezvous and Optical Navigation) [46], employs a different approach to physical simulation, but it is a method that is becoming an industry standard. Using ten wall-mounted high-fidelity LED panels

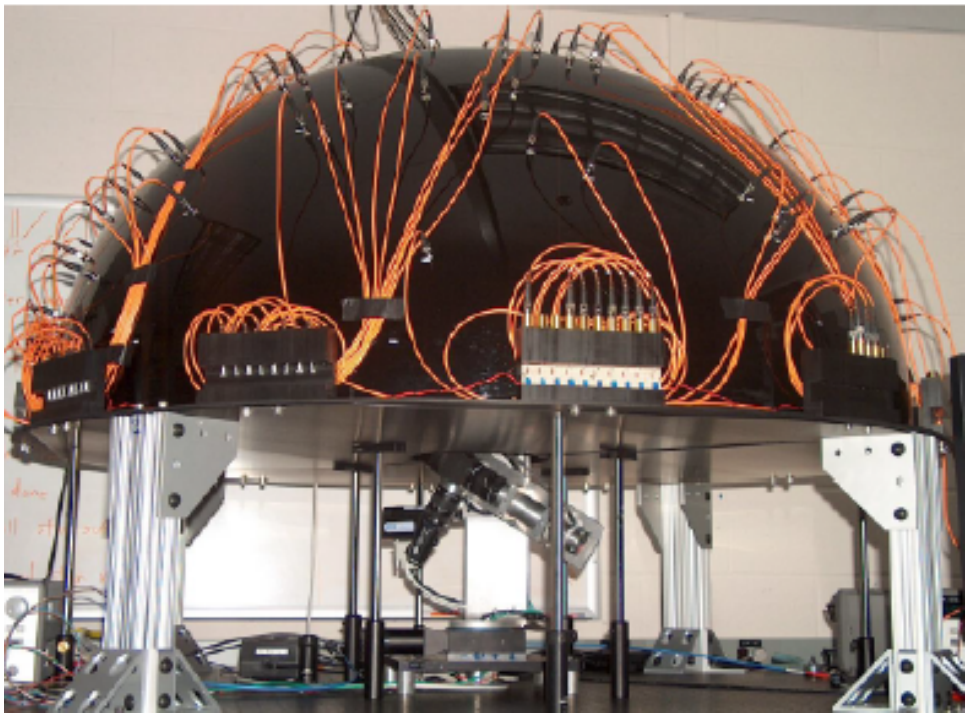

Figure 2.1: Optical simulator and testbed for spacecraft star tracker development [14], Johns Hopkins University Applied Physics Laboratory

to reproduce diffused Earth albedo and sunlight sources, 1:25 scale 3D spacecraft mock-ups, and a seven degree-of-freedom (7DOF) robotic arm to maneuver camera capture elements within the environment relative to the target spacecraft illustrated in Figure 2.2. This system can simulate space-borne imagery by accurately positioning and matching geometric and radiometric light sources for the purposes of testing and validating visual-based navigation algorithms and controls. The system also integrates a variable magnification optical simulator consisting of two physical lenses to facilitate both narrow and wide optical fields of view (FOV) operations. However, this method uses post-processing techniques to overlay objects of interest such as mock-up satellite bodies onto a space background, a synthetic image generation process sometimes referred to as a "photoshop" approach. This can be effective, but it can be time-consuming, limited to static imagery, and eliminates post-processing alterations such as domain randomization.

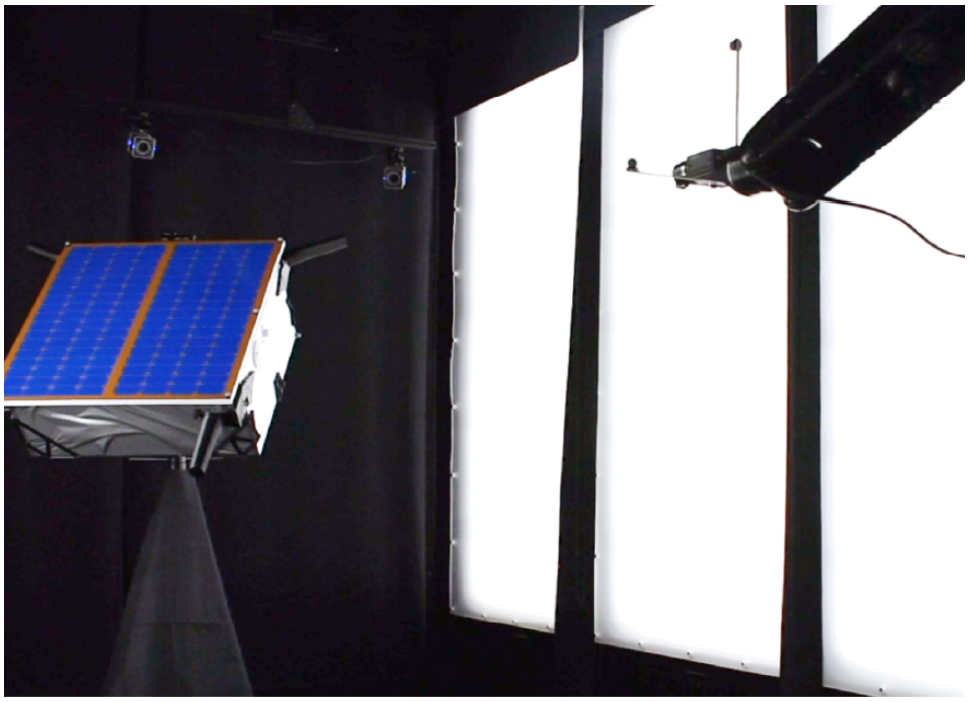

Figure 2.2: TRON Testbed located Stanfords The Space Rendezvous Laboratory (SLAB) [46] Facility

The German Aerospace Center (DLR) deploys a similar physical simulation approach using robotic manipulators but doesn't generate high fidelity lighting, instead focuses on final stage spacecraft rendezvous and docking. The sensor of choice here is known as a Photonic Mixer Devices (PMDs) [33] seen in Figure 2.3. PMD smart pixels measure the intensity and distance of a target pixel using modulated light, without the need for active scanning. Using the phase delay of each pixel in a given frame, PMDs can effectively determine image depth information from a 2D picture, allowing for closed-loop control. This capability is extremely useful when determining spacecraft range, especially for uncooperative or tumbling targets, and may become the industry standard sensor for space-based applications in the future. This facility takes validation a step further by implementing and testing WAN-simulation of User Datagram Protocol (UDP) and Tunnel Setup Protocol (TCP) data links, complete with packet loss and signal delay simulation between the ground station and any hardware

and/or software currently using the test stand for computer vision-based decision making and navigation.

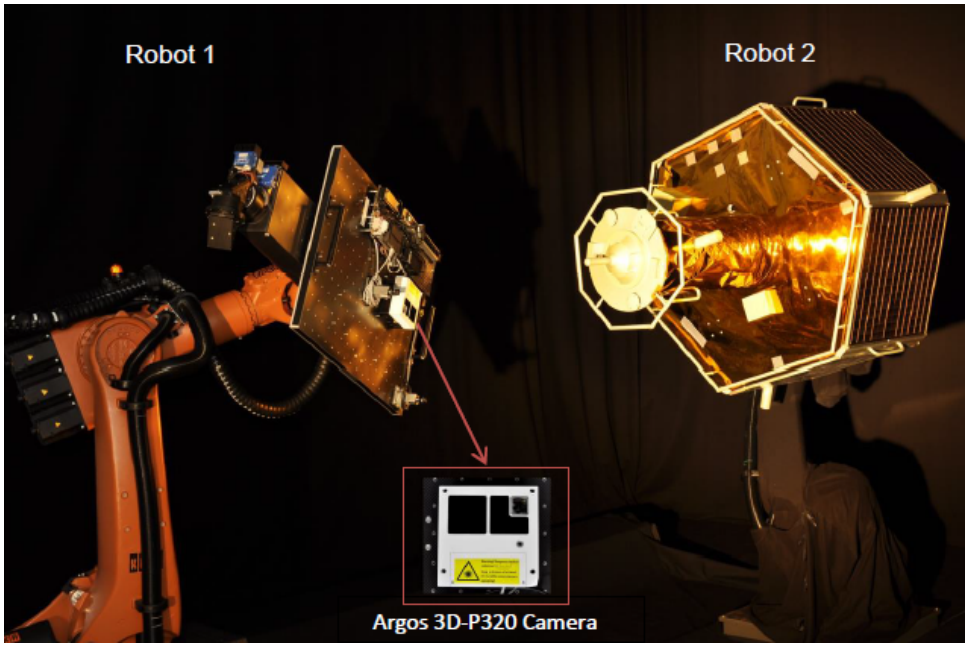

Figure 2.3: DLR's Docking simulation using Photonic Mixer Device Senors [33]

Simulation methods outside of highly controlled lab environments have also been deployed in the field. The German aerospace center has utilized overhead drone flights and meticulously placed real-world mockups of 80 lunar surface craters ranging in diameters of 5m to 0.5m over an area of 45,000 square meters, depicted in Figure 2.4, to develop Crater Navigation (CNav) algorithms for human out of the loop autonomous terrain-based optical navigation (ATON) for lunar landers [56]. Using segmentation image processing to back out known crater patterns and crater shadow direction, this testbed was able to successfully reorient lunar location after a complete loss of navigation, also known as the "lost in space" scenario. Starting at an altitude of 50m and steadily descending to 20m in a straight-line path, the drone used a forward-facing monocular monochromatic camera with a resolution of 1024p x 1024 px at 20 Hz. The camera was paired with a laser altimeter with a matching pro sight angle, which provided ground truth telemetry. This framework ultimately achieved a

position error of less than 2m, velocity error of less than 0.4 m/s, and attitude error of fewer than 2 degrees.

Figure 2.4: Optical Navigation via crater detection for lunar landers [56]

All of the physical methods for space domain optical simulation detailed above are necessary for real world hardware/software validation, but they can be further augmented by introducing synthetic visualization and domain randomization to create more robust, representative, and exhaustive simulation frameworks, many of which are detailed below.

2.1.2 Surrogate Data Generation

Synthetic visualization is often used to generate massive amounts of hyper-realistic imagery using rasterization engines such as the Unreal Engine [26] or Blender [19] (example shown in Figure 2.5), which have proven to be a powerful means of solving both limitations in gathering specific space domain imagery and recreating variations found in space. Recent industry capabilities such as Nvidia's photon ray-tracing simulation technology have increased synthetic data viability by more accurately visualizing lighting effects. This is enabling machine learning architectures to begin testing and deployment for space applications because we now have a means of generating the massive amount of image data required to train a neural

network to identify any desired object of interest with acceptable performance.

Researchers have begun to leverage these powerful world-building tools, creating hyper realistic Earth-Sun environments. Organizations such as the Surrey Space center have demonstrated that purely synthetic space domain imagery can be used to train a convolutional neural network to perform spacecraft Pose, achieving an orientation error of less than five degrees and localization error of 0.17m using ResNet [31] neural network architecture, but the synthetic data has yet to be evaluated on-orbit with real-world spacecraft.

Figure 2.5: Pose Estimation from Photorealistic Rendering [59]

Similarly, the United States Naval Academy conducted a similar study to perform spacecraft attitude initialization shawn in Figure 2.6. Using Blender to render a Dragon X spacecraft model at varying angles, labeling the resultant images defined by Euler angle rotations at intervals of 10 degrees relative to the camera frame. Finally, a series of nine fully lit (no nighttime images) high-resolution images from the GOES-16 satellite were randomly sampled and placed in the background. With a grand total of 27,000 labeled images, the resulting

labeled data set was then passed into Matlab's Deep Learning toolkit prior to being passed into AlexNet [34] CNN Framework using transfer learning. This methodology yielded a 95% spacecraft angle accuracy rate.

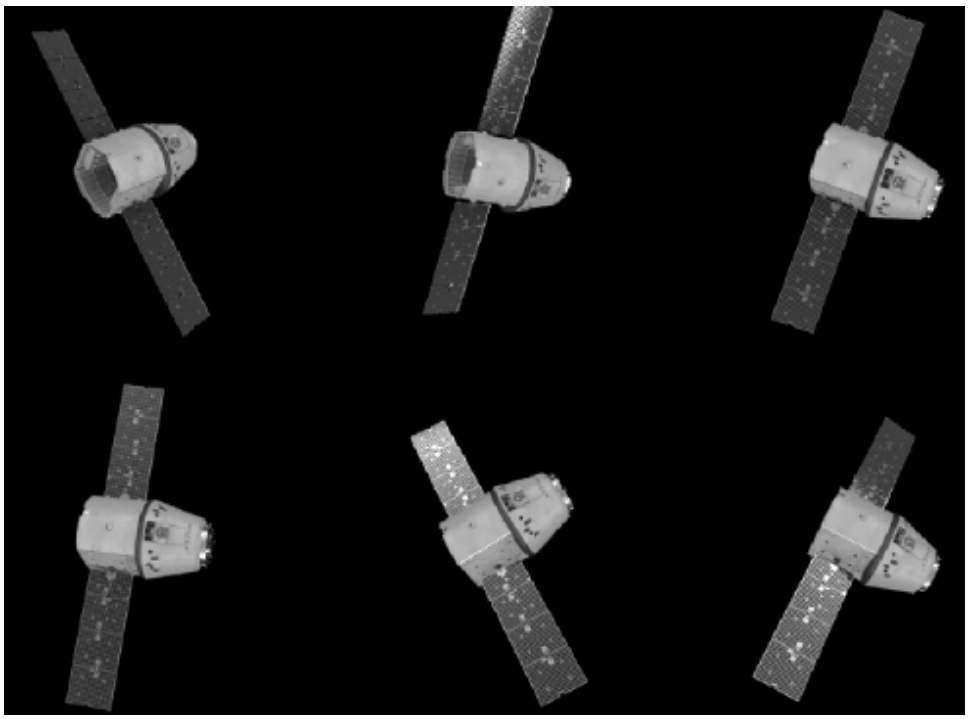

Figure 2.6: SpaceX Dragon Synthetic image rendering setup in Blender [17]

Finally, the SurRender [35] image rendering software illustrated in Figure 2.7, developed by Airbus, is designed to provide detailed realistic lighting for celestial surface imagery (e.g., Mars, asteroids, and orbital debris) to test and improve vision-based navigation (VBN) algorithms. It tackles some synthetic imagery problems such as secondary illumination, realistic light propagation, and ray tracing light transport restoration technology. Focusing on pure photometric accuracy that has gone through a formal qualification process with the European space agency (ESA). It is also capable of producing synthetic LiDAR data for Monocular SLAM testing, making the SuRender Software perhaps the most robust purpose-built API to solve the problem of obtaining accurate synthetic space-based imagery. However, this software requires a license from Airbus for use.

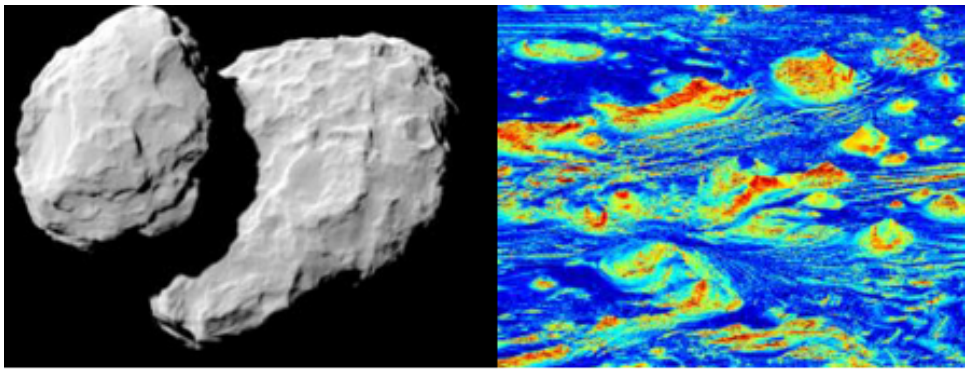

Figure 2.7: SurRender visual simulation demonstrating photometric accuracy [35]

While visual simulation is a key part of enabling neural network training for cross-domain applications, and consequently neural network performance. The other half of neural network performance is dictated by the architecture used, which is derived from a number of variables described in the following section.

2.2 Today's State of the art CNNs

Some of the highest performing algorithms used for object detection tasks today are R-CNN [29], Fast R-CNN [48], and YOLO [13] architectures. However, at the time of this publication, the YOLO (You Only Look Once) architecture is frequently regarded as an industry leader in the field of computer vision. YOLO derives its detection speed and efficiency in part from the introduction of Anchor boxes, defined as a collection of boxes spread out over an input image, with widths and heights chosen to match object sizes in a particular data set or some other criteria based on the particular problem. More traditional sliding window methods depend upon arbitrarily sized boxes requiring multiple passes over an image in hopes of finding an object feature that matches predefined sliding window dimensions near perfectly. Anchor boxes disperse thousands of dynamic candidate boxes throughout a given input image. Each anchor box is used to calculate the percentage of

Intersection over Union (IOU), demonstrated in Figure 2.8, by assigning an object score used to estimate the probability that an object exists within a given anchor box regardless of the predicated label, thereby only requiring a single pass over an image. Object scores combined with object classification probability determines *where an object is* and *what an object is* for a given input image. Both of these parameters are incrementally updated via backpropagation, adjusting and improving the size and shape of the anchor boxes as the model is trained. This ability to maintain neural network and frame rate performance even when deployed on smaller edged compute devices makes YOLO and other computer vision architectures like it uniquely suited for spaceborne applications.

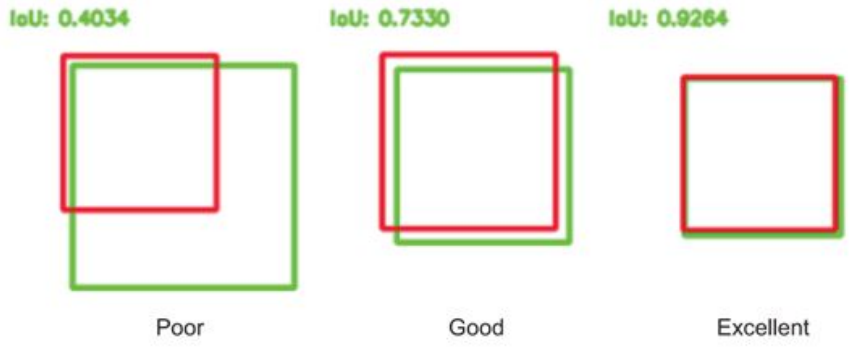

Figure 2.8: Intersection over Union (IoU) bounding box threshold examples.

2.3 Domain Randomization

In the past, the use of synthetic data has led to a problem known as the *reality gap*, defined as the unavoidable deviation of any simulation from reality. *Domain Randomization* is a method for creating simulated data by constantly changing the environment of any given data set. Domain randomization is the process of introducing variations into a given data set. When applied to a representative synthetically generated environment, real-world objects will eventually be generalized as just another randomized permutation. Domain Randomization

has the added benefit of allowing models to generalize unforeseen circumstances far more reliably than non-domain randomized data sets. Applying this method to synthetic data generation and production processes has generally improved a neural network's ability to detect and classify real-world objects inside and outside the intended domain. Thus, enabling researchers to apply synthetically trained computer vision models to real-world environments, in a process commonly called "cross-domain" or "cross-domaining." Given the near limitless range of parameters that can be edited dynamically within a synthetically rendered world, it is intuitive to ask which parameters incur the most significant impact on neural network and associated computer vision performance. Ablation studies like those conducted at the University of Toronto [55] evaluate the performance impacts of several parameters such as lighting, orientation, asset textures, and the use of flying distractors. Each parameter individually affected overall neural network performance ranging from 2%-7%, with lighting variation having the most significant impact when evaluating on the KITTI data sets [28]. This approach applied to a self-driving application is illustrated in Figure 2.9.

Figure 2.9: Synthetic Domain Randomization applied to self driving cars published by Nvidia [55]

When applied to spaceborne computer vision tasks (Figure 2.10), we posed the following question. Can domain randomization be used to solve or mitigate some of the following

vision-specific problems?

- Variations in size and texture of object of interest
- Distance and orientation of object relative to camera sensor
- Light source intensity/degradation
- Computer vision inferencing using supplemental lighting only
- background variations including deep space and planetary

Figure 2.10: Synthetic Data Domain Randomization (Unreal Engine 4)

2.4 Computer Vision in the Space Industry

Several thousand publications from leading conferences such as the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), and the

Society of Photo-Optical Instrumentation Engineers (SPIE) illustrate solutions to several critical industry problems related to in-space assembly and on-orbit servicing. Manuscripts detailing satellite teaming & path planning, spaceborne manufacturing techniques and materials, spaceborne inverse kinematics, and the economics of such ventures have been introduced over the last five years.

However, the number of publications demonstrating true human out-of-the-loop solutions leveraging machine learning and computer vision capabilities needed for orbital and deep space systems are far less numerous, as detailed in Figure 2.11. Methods to simulate and validate spaceborne vision-based algorithms and methodologies are even less multiple.

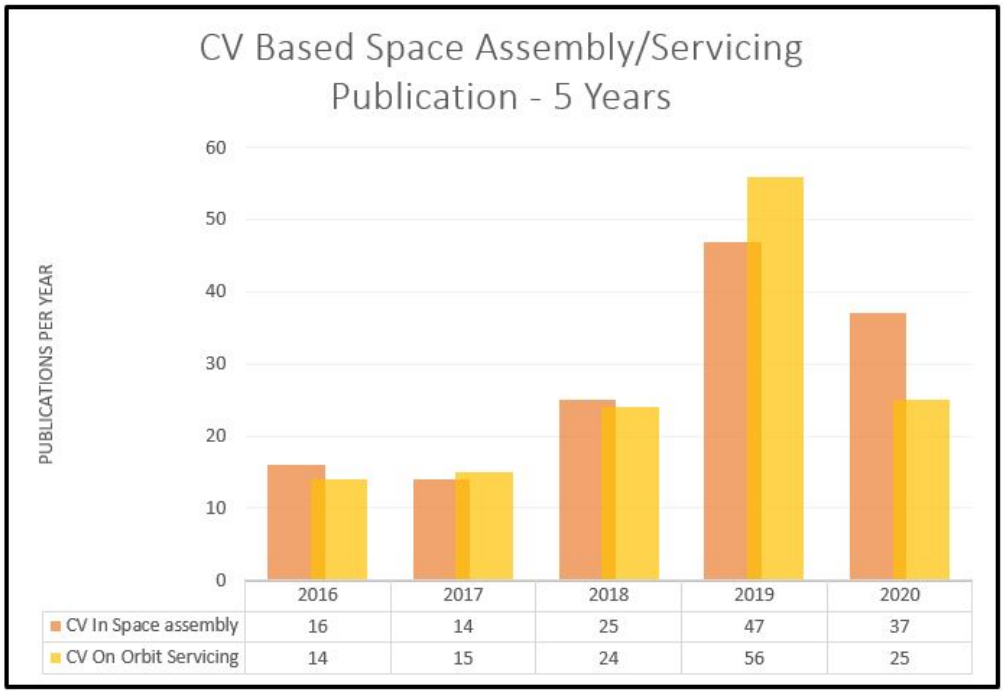

Figure 2.11: Major Conference and Journal papers (2016 - 2020) detailing vision based solutions for space-based assembly, servicing, and manufacturing [45]

To date, few publications employing domain randomization for computer vision tasking have been demonstrated, as illustrated by Figure 2.12. To my knowledge, no other literature exists utilizing domain randomization as a means to solve the optical variation problems associated

with spaceborne applications has been published.

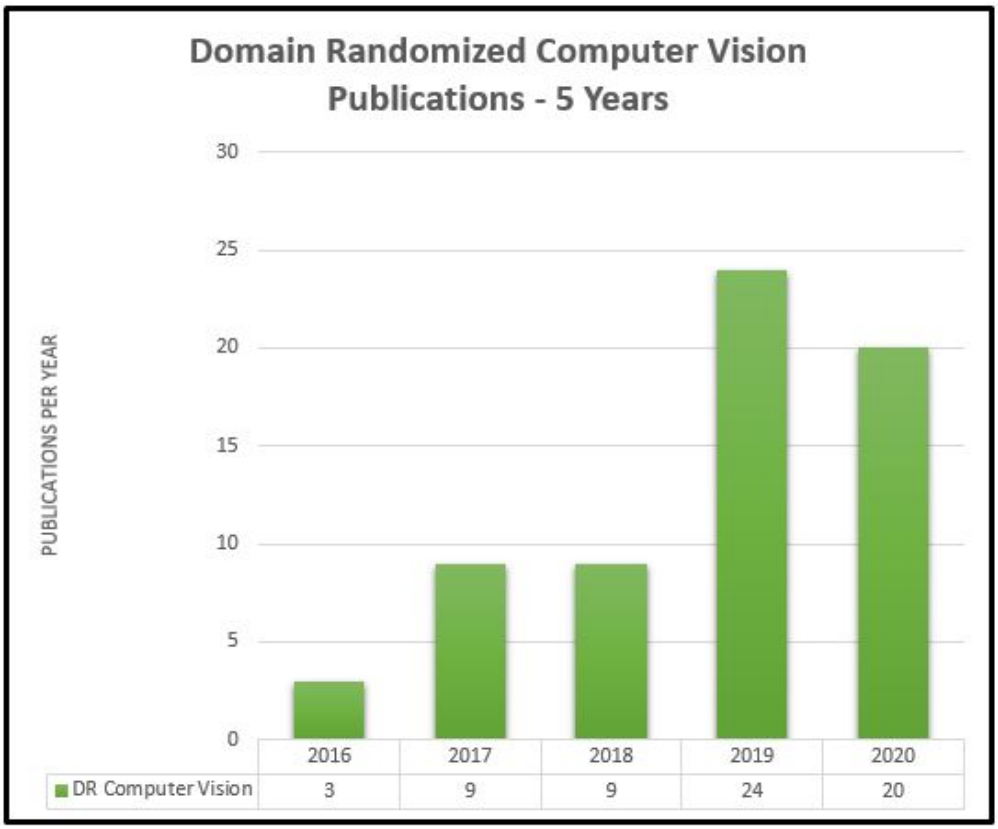

Figure 2.12: Major Conference and Journal papers (2016 - 2020) illustrating domain randomized computer vision solutions [45]

2.5 Ground Truth Tracking

Any hardware-in-the-loop validation system requiring object motion and interaction needs the ability to establish a known ground truth position and orientation. This has been done with many off-the-shelf motion capture solutions, but perhaps the most widely available systems pre-packaged with integration APIs are developed for the virtual reality industry. As such many scientific organizations, including NASA, have leveraged this technology for verification and validation of vehicle motion. The Astrobee program detailed in Figure 2.13 utilized the steam VR tracking system to capture dynamic motion while simulating an ISS environment. This is the same motion capture system deployed onboard the SpaceDrones robotic platform to establish ground truth movement, which is later be discussed in the Material and Methods chapter of this paper.

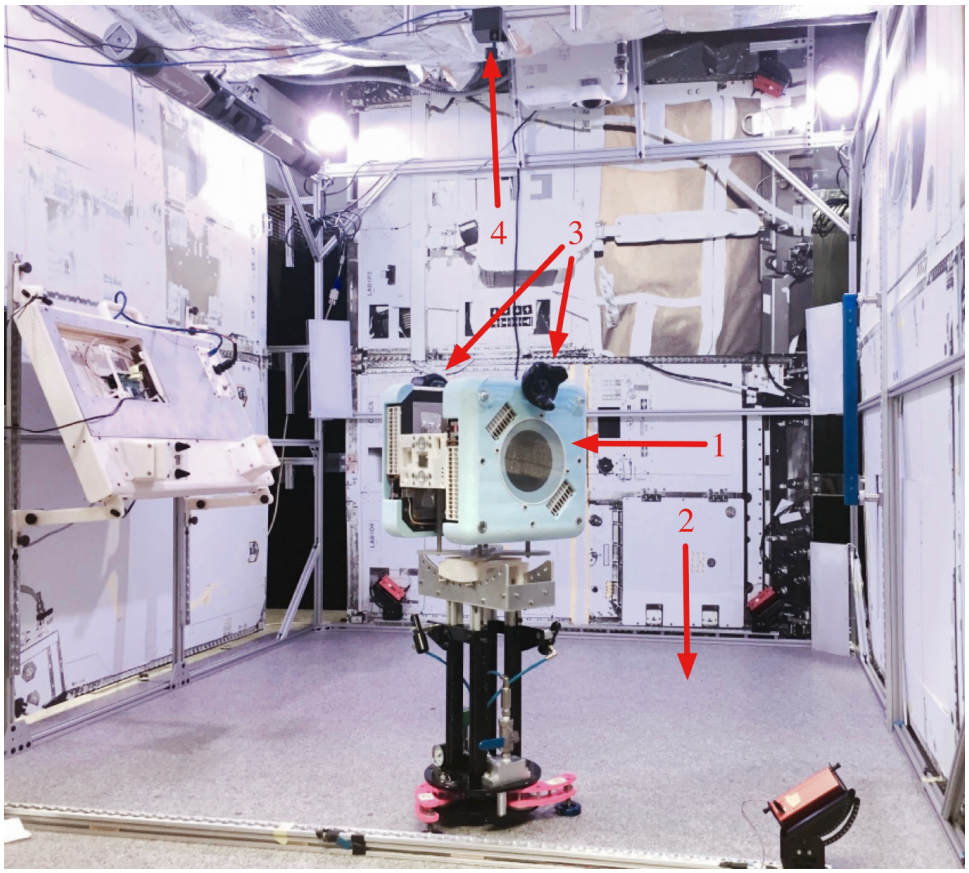

Figure 2.13: Astrobee (1) shown on a level granite surface (2), which is used to simulate a 2D microgravity environment. The prototype has trackers mounted to its port and starboard sides (3), and a single lighthouse (4) mounted overhead [15]

Chapter 3

Neural Network Modeling

This chapter gives a short, high-level overview of neural network architecture frameworks used for computer vision applications for audiences that may not be familiar with the topic. Convolutional neural networks (CNNs), the most common neural networks for computer vision, are networks characterized as computational layers performing a series of convolutions over grid-like topology data (e.g., images or time-series data) to extract feature maps from a given input. Due to CNNs unique ability to deduce relationships and dependencies between pixel structures, these architectures are uniquely capable of deriving features from images and videos, enabling viable object detection functionality. Neural Network Modeling is a rapidly growing and evolving industry. New architectures are developed monthly (if not weekly) that advance state-of-the-art, it would be futile to describe the most recent developments within a static piece of literature. Still, almost all Convolutional neural networks use the same fundamental building blocks comprised of two parts: 1) feature extraction and 2) classification.

1) Feature extraction is accomplished via the convolutional block, which is primarily divided into three subsections. The first of these sections is the convolution operation itself, governed by the size and weights of the convolution kernel. A kernel is a simple matrix, which slides across a given image and multiplied with the topological input matrix (Usually from an image) such that the output is enhanced in a certain desirable manner. Downstream from the convolution operation, an activation function (generally zero-centered) defines the

output neurons and adds nonlinearity to the CNN. Lastly, a pooling operation is often included, which refines a large matrix space into smaller feature maps by prioritizing the most-pronounced components of the input and disposing of the rest, resulting in a feature map matrix containing only the most relevant information encoded within the input image, completing a convolutional block. All neural networks do not follow the same order, but the parts of the block are generally consistent across different neural network types and architectures. Multiple convolutional blocks can be chained together to refine the feature map further.

2) Classification is the last operation employed by (most) CNNs. During this step, the feature map is processed via a series of fully connected layers. Because these fully connected neurons expect a 1xN vector as an input to the network, the final feature matrix produced by the last Convolutional block in the chain is reshaped into a 1-dimensional vector, and input neurons are assigned to each index of this feature vector. Neurons then perform a series of operations based on initial weights, bias values, and any activation functions within the network. This results in a probability score representing the CNNs confidence that the input image contains/matches a defined object of interest. This classification is then compared against the ground truth of the (manually) labeled input, and an error metric is computed. This error is then backpropagated throughout the neural network to each weight via (stochastic) gradient descent. This eventually minimizes the error between the final probability output and the ground truth. Depending on any number of parameters, methods, and data sets used for training outlined below, can result in a robust neural network capable of identifying any number of objects of interest.

3.0.1 Activation Functions

Activation functions are used both in the feature extraction stage and the fully connected learning stage of CNNs. They serve as nonlinear functions implemented to aid the network in solving complex nonlinear solutions for a given data sets. They also push values to a desired range for easy probability determination. Without them, a network would only learn linear solutions regardless of the network's depth, which is often an inappropriate solution to real world problems.

The sigmoid function was once widely regarded as the most effective and popular activation function for training neural networks. Sigmoid functions effectively map an input value to an output value between zero and one according to a logistic curve. This is useful for interpreting probabilities for logistic regression problems. However, this function is accompanied by two significant issues. 1) The gradient used for backpropagation tends to vanish after reaching a certain threshold because the neuron activation saturates close to zero or one, driving the gradient value very close to zero, thus leaving the network unable to further approximate or improve the model. This often results in the termination of gradient propagation before reaching the desired accuracy threshold. This issue is commonly referred to as the "Vanishing Gradient Problem". 2) The sigmoid function is not zero-centered, causing gradients to tend towards all positive or all negative values, introducing a non-optimal "zig-zag" update to network weights when performing gradient descent optimization.

Another popular activation function is the Rectified Linear Unit (ReLU) [7]. This activation function solves the vanishing gradient problem by remapping all negative values to zero, preventing negatively weighted neuron from reaching gradient saturation. ReLU also avoids expensive computations by remapping negative values to zero. These attributes have propelled ReLU to the top of most neural architectures as the activation function of choice.

However, the strict negative to zero mappings caused negatively weighted neurons to not activate in a problem coined "the dying ReLU Problem." This issue has been addressed by allowing small negative values to exist and allowing negative neurons to fire and contribute to the network's propagation process. This augmented version of ReLU is called "Leaky ReLU" (illustrated in Figure 3.1) and, as of today, is the industry standard for activation functions, except for the final output activation layers, which are often softmax activation functions detailed in the next subsection.

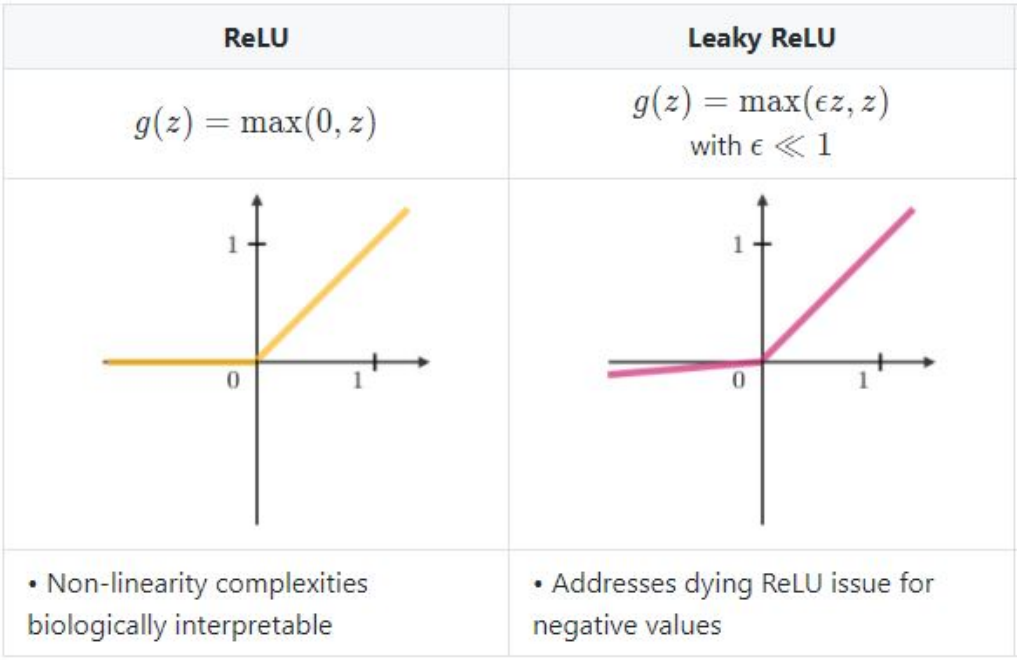

Figure 3.1: ReLu vs Leaky ReLu nonlinear activation functions

For non-binary classification problems (i.e. problems in which more than two types of objects are to be detected), softmax activation functions detailed in Equation 3.1 are often used to map all outputs to a probability range of zero to one such that all mapped outputs sum to one for the final output layer. Softmax mathematically represents a normalized version of the winner-takes-all activation model in which the output neuron with the largest value is deemed the correct classification.

$$\text{softmax}(z) = \frac{e^z}{\sum_z e^z} \tag{3.1}$$

3.0.2 Pooling

In general, pooling serves to reduce dimensionality and preserve spatial invariation. Because neighboring pixels are strongly correlated (especially in lower fully connected layers), it makes sense to reduce the size or "dimensionality" of the output by subsampling (pooling) the kernel/filter response. The further apart two pixels are from each other, the less correlated they become. Pooling is often performed more than once during feature extraction. A 2x2 pooling convolution matrix combined with a pooling stride length of two results in a sliding window with zero overlaps, reducing the feature map dimensionality by 75 percent. Pooling has the benefit of making the feature map representation invariant to small translations. If a small amount translates the input, the values of most of the pooled outputs will not change. Max pooling also serves as a noise suppressant by discarding the noisy activations altogether.

Most modern CNNs use two types of pooling techniques (shown in Figure 3.2): max pooling and average pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel (usually 2x2), while average pooling returns the average of all the values. Average pooling smooths out the image rendering sharp features less identifiable, while max-pooling generally extracts sharp features such as edges. For image data, max-pooling often has more favorable performance characteristics This is often the last step of a given convolutional block.

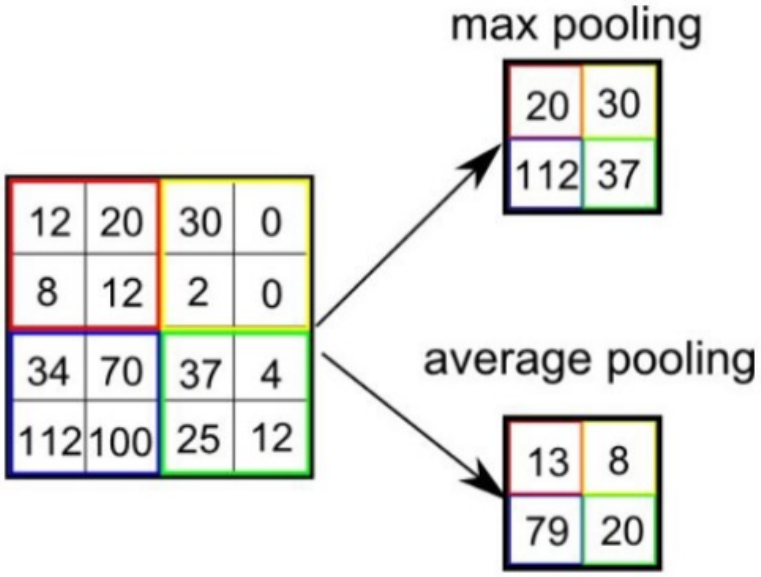

Figure 3.2: Example of Pooling methods

3.0.3 Fully Connected Layers

Implementing a fully-Connected layer is (usually) a simple way of learning nonlinear combinations for high-level features extracted by the feature map. After the feature map matrix is stacked into an nx1 vector and forward-propagated through the neural network, some error will be identified between the predicated value and actual value, and then back-propagation is applied to reduce error. Performing this process over a series of epochs allows the model to distinguish between dominating and certain low-level features in images and classify them using the softmax classification technique.

Figure 3.3 illustrates an example of the ResNet [31] neural network architecture using "residual blocks" with many of the elements described above as well as forward and back propagations that are detailed in the following sections.

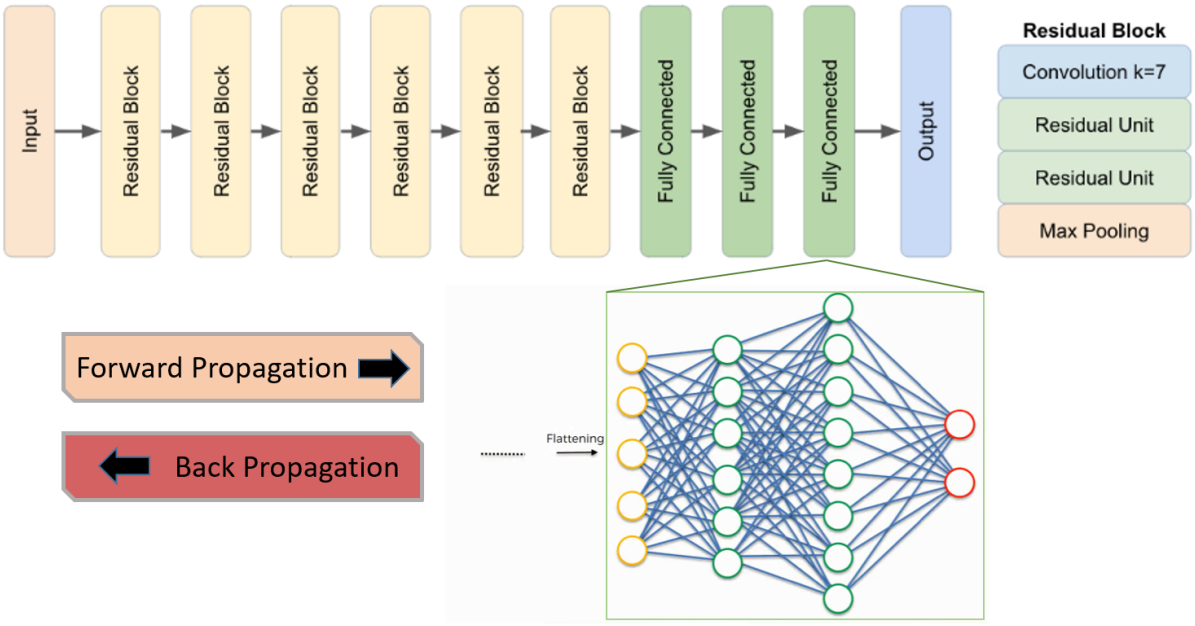

Figure 3.3: Example of a convolutional neural network (CNN) architecture. This particular architecture is a CNN known as ResNet [31]

3.0.4 Forward Propagation

After all the layers are constructed and connected, an input X provides initial information that can then be propagated through hidden units (either fully connected neurons or convolutional filter values) at each layer via a series of learned weight values to produce some output. The value at each neuron in the network is recursively dependent on every other neuron before it. Depending on the subsequent connection's weighted values, a neuron can have a large or negligible effect on its neighbor in the next layer. This dependency is defined as a receptive field and is one of the significant strengths contributed to neural networks. Neurons eventually become tuned to learn specific features of a data sets and, thus, have a more substantial impact on the output layer when that feature is detected or a minimal impact when it is not. Neurons closest to the output layer have the most significant influence on the outcome, while layers most proximate to the input layers have diminished effects. Because of this, the addition of an infinite number of layers to a neural network does

not increase performance. Eventually, layers that are an n distance away from the output layer will have zero effect on the outcome due to the calculated error gradient vanishing as it is backpropagated through the network. An arbitrary bias term is artificially added to each neuron value to ensure this does not occur at the start of training. There are methods for increasing neuron participation as networks become deeper such as neuron drop-out and layer skipping, but are outside the scope of this particular dissertation. The process of forward propagation is the networks object classification attempt given some input image. This attempt generates some error between the network classification and the ground truth. Minimizing this error is the job of back propagation detailed below.

3.0.5 Back Propagation

In supervised learning, back propagation is the process of comparing a neural network output layer to known ground truths and reducing an error via a cost function (e.g. the L2 cost function). The magnitude of the error for a specific output neuron determines the magnitude of change that must be imparted on the subsequent weights in order to reduce the error closer to zero at a more rapid rate. The cost function's objective is to bring the error for all output neurons to zero in a process called gradient descent.

The following equations detail the back propagation processes mathematically.

$a^i =$ neuron value

$L =$ number of neurons

$C =$ cost

$w =$ weight value

$b =$ bias value

$\sigma =$ activation function (e.g. ReLU)

$y =$ ground truth

In Eqs. 3.2, 3.3, and 3.4, neuron-wise operations are defined.

$$a^{(L)} = \sigma\left(w^{(L)}a^{(L-1)} + b^{(L)}\right) \tag{3.2}$$

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)} \tag{3.3}$$

$$a^{(L)} = \sigma\left(z^{(L)}\right) \tag{3.4}$$

Defined in Eqs. 3.5 through 3.8 is a simplified cost function for of a specific training example during the evaluation a single neuron.

Cost Function.

$$C_0 = \sum_{j=0}^{n_L-1}\left(a_j^{(L)} - y_j\right)^2 \tag{3.5}$$

Weighted Cost to a single Neuron.

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C0}{\partial a^{(L)}} \tag{3.6}$$

Cost of all Neurons in the Network summed over L layers.

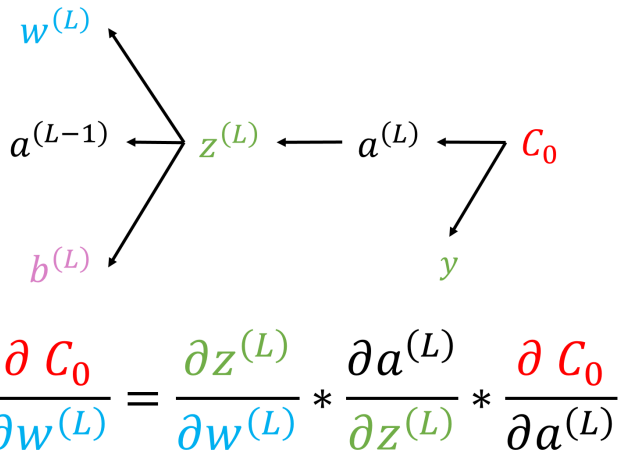

Figure 3.4: Simple single neuron back propagation diagram.

$$\frac{\partial C}{\partial a_k^{(L-1)}} = \frac{1}{n} \sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{L-1}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \tag{3.7}$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix} \tag{3.8}$$

These cost functions can be expanded across multiple neurons over an entire layer as demonstrated in Eqs. 3.9 and 3.10.

$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)} + b_j^{(L)} \tag{3.9}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \tag{3.10}$$

Equation 3.5 is also illustrated graphically in appendix A Figure A.1. Mathematical Figure 3.4 and Equation 3.6 are also illustrated graphically in appendix A Figure A.2.

Chapter 4

Materials and Methods

4.1 CNN Architecture Selection

A dizzying array of convolution network architectures (CNNs) such as Faster R-CNN [48], RetinaNet [38], and Cross Stage Partial ResNet (CSPR) [57], each altering various parameters such as size and number of neural network layers, nonlinear activation functions, pooling functions, image input size, and many more in constant evolution to increase a model's performance. Additionally, entire architectures can be changed, spawning subset architectures such as generative adversarial networks (GANSs) [30], DropConnect networks [53], and Deconvolutional networks [60]. The advantages and disadvantages of each can be narrowed down based on the power, time, or performance constraints for a given problem set.

I ultimately selected the "You Only Look Once (YOLO)" [13] architecture and a modified Pytorch framework (commonly known as YOLO v5) for integration into the SpaceDrones API. The YOLO architecture can be trained and deployed on a single GPU with a smaller mini-batch size, enabling us to deploy the architecture on edge compute devices comparable to those found on a small satellite payload. Figure 4.1 illustrates a simplified example of using synthetic imagery as input to a given neural network for training, to back out objects of interest, in this case, trusses, solar panels, and spacecraft.

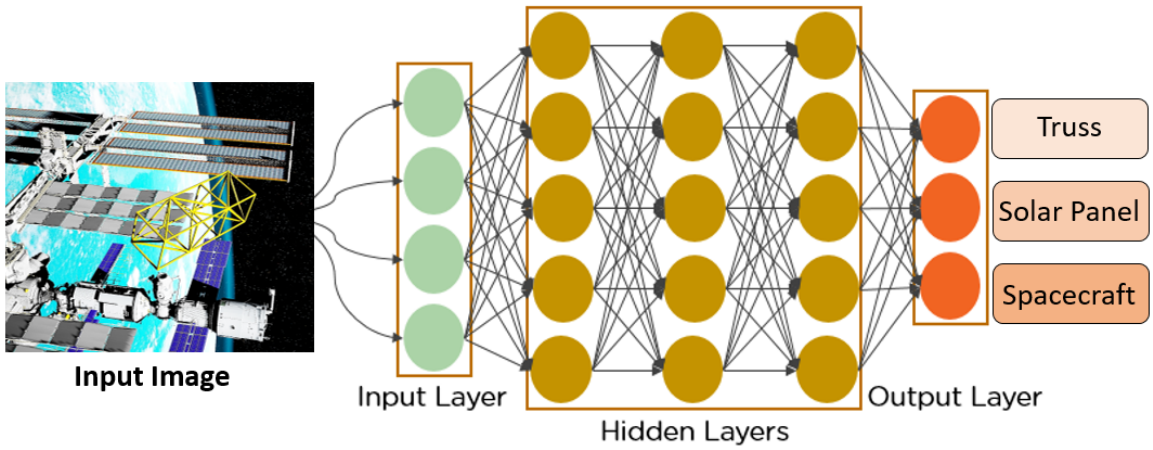

Figure 4.1: Simplified Neural Network architecture diagram.

4.2 Synthetic Reality creation and collection

Environmental factors can significantly influence the training of CNNs for the classification of images, it was therefore also imperative the visual environmental conditions were captured as much as possible when using synthetic data. To generate a robust detection model that is capable of recognizing the relatively complex construction problem set consisting of several geometrically similar objects such as truss sections in a complex orbital regime, synthetic images were generated and stored. This was accomplished using a hyper-realistic Earth/-Moon System environment with ray tracing dynamic lighting was created at a resolution of 15360 by 8640 pixels (16K) detailed in Figure 4.2. The following factors were considered to ensure that the synthetic data was representative of these conditions: asset distancing, perspective projection, dynamic lighting, model detail, color accuracy, etc.

This world building process will differ between particular environments, but the overall development process is generally the same. Starting with a central landscape asset such as planet Earth asset or a textured Mars landscape asset to serve as the foundation for the environment and world central point. Second, the lighting assets were created and placed,

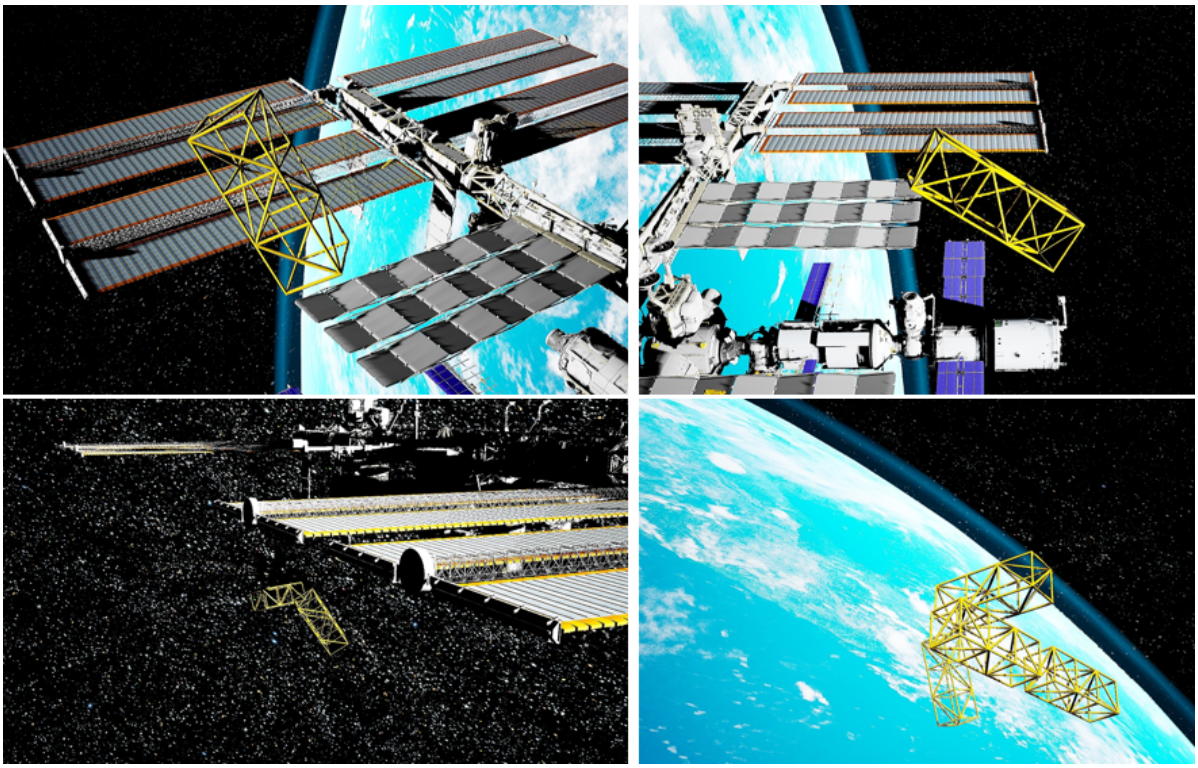

Figure 4.2: Example Orbital Images Generated using Unreal Engine 4

in order to see real time lighting effects on other assets as the are placed into the synthetic environment. Additional lighting parameters such as ray tracing or orbital glare effects are also generated in this step. Once lighting is established, any desired effects such as Earth albedo, cloud distortion, and sky box assets are added. Assets such as sun and moon models, star formations, or martian rock formations are either created or loaded from open source repositories to complete the overall environment. After all environmental assets are generated appropriately, objects of interest to the computer vision algorithm such as truss sections, solar panels, and ISS models are loaded to complete the synthetic scene. This completing the synthetic visual environment setup.

The next step is to create the dynamic conditions governing how objects move and interact with each other with in the synthetic world. These parameters include creating object motion via predefined animation sequences or dependent upon other object or world states such as

object relative location or collision detection. Many of these parameters are controlled and modified via the unreal engine blue printing system.

Finally after the world has been created, object motion defined, and interaction parameters set, imagery is collected either through purpose built visual camera sequences designed to create pre-rendered scenes or, in the case of the domain randomization data sets, through direct game play interaction were a player is physically manipulating assets in the game world.

4.2.1 Synthetic Domain Randomization

To test the effects of domain randomization on a neural network's ability to classify an object of interest, in this case, 3D printed truss pieces for construction tasking, a highly randomized zero-gravity environment was developed within the Unreal Engine. All elements including the skybox, distractors, and objects of interest were programmed to randomize color and lighting effects over a given time frame. This method allowed us to generate thousands of highly randomized images across several object angles relatively quickly and allows us to perform domain randomization at the preprocessing level. Performing randomization operations before image rendering obviously allows for greater control, as appose to applying filter effects on an image at the post-processing level. Domain randomized synthetic environments are illustrated in Figure 4.3 and via video link.

Once all assets in the environment had been rendered and motion was modeled, a built-in camera tool captured the scene at 1080p. This media was then saved to a local hard drive and labeled . Once all images were labeled, they were then uploaded to a SQL database for ease of network upload into the CNN model for training.

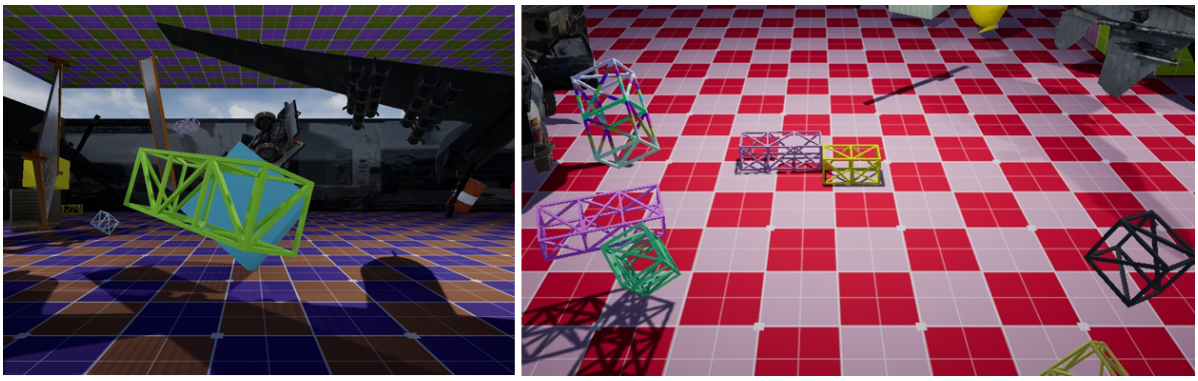

Figure 4.3: Example Domain Randomization Images Generated using Unreal Engine 4 Video illustration

4.3 Porting CAD Models & Hardware into Synthetic Environments

To introduce CAD Models into the synthetic environment, 3D printed STL files were injected into the Unreal Engine upon converting them to OBX files. To accomplish this, Blender was employed to convert appropriate file types and adjust environment scaling. For example, most CAD files define the origin at (0,0) and build the object in quadrant one, thus keeping all values positive and effectively placing the origin in the corner. Attempting to accurately model dynamics creates a conflict due to UE4 (Unreal Engine 4) colocating an object's center of gravity with the origin point, resulting in off-center dynamics when introducing object motion. To solve this, Blender is used to move object origin points to the geometric center of objects and set units of measurement to meters before porting. This workflow file conversion workflow is detailed on Figure 4.4.

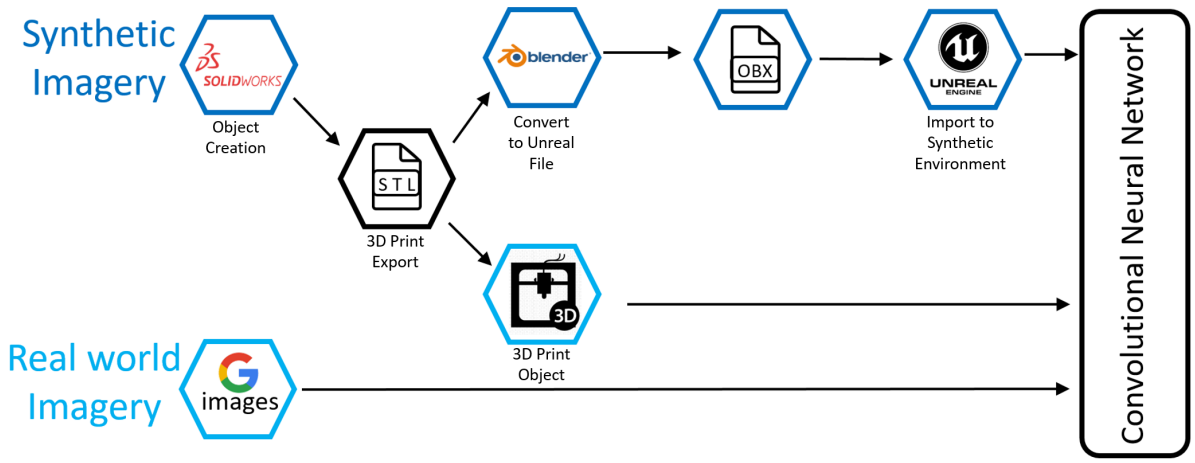

Figure 4.4: Project work flow when creating 3D objects for both synthetic environments and 3D printing

4.4 Methods of Labeling

Due to the inherent similarities between the truss sections, where a 2U truss is simply 2 1U trusses put together and a 3U truss can be any combination of one or two "U" truss segments, neural networks have difficulty differentiating truss objects making this a far more difficult problem to solve. This is particularly true at the edge of the frame - in what we are calling the *"Ground truth vs in frame"* labeling problem. For example, when a 3U truss section is only two thirds visible within a given picture frame, appearing as if to be a 2U truss section as detailed in Figure 4.5, how should that particular truss section be labeled by a human labeler to established ground truth for a neural network?

As the human user performing supervised learning, we often know the actual dimensions of the given truss outside the frame, but labeling it as such presented problems during training. It was discovered that labeling truss sections as they come into view increase CNN performance by 3%, thus this is the solution used for the rest of this dissertation.

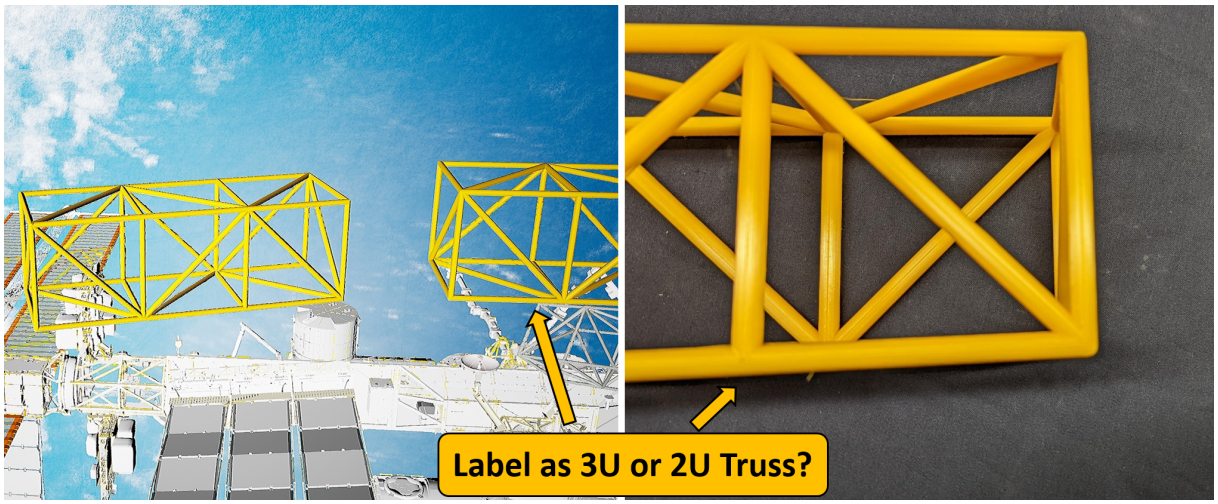

Figure 4.5: Ground truth vs in frame Labeling Problem

4.5 Division of Data sets

Once labeled, all images were compiled and organized by experiment. Synthetic training images, real-world, and synthetic testing images were separated into four primary data sets illustrated in Figure 4.6. Synthetic data sets serve as training data for the neural network, while real-world data sets are evaluated against synthetically trained CNN models to determine the feasibility of artificially trained neural networks for real-world inferencing applications. To further increase the amount of training data, some synthetic images were duplicated and a rotational transformation (90 degrees $< \theta <$ 180 degrees) was applied. This approach is acceptable for ISA applications due to the free flying orbital rotation, the "bottom" of an image is not always considered to be the "ground" as it is in typical terrestrial imagery. After image rotation all training data sets were then compiled to a database where they could be queried as needed by the CNN to run applicable experiments.

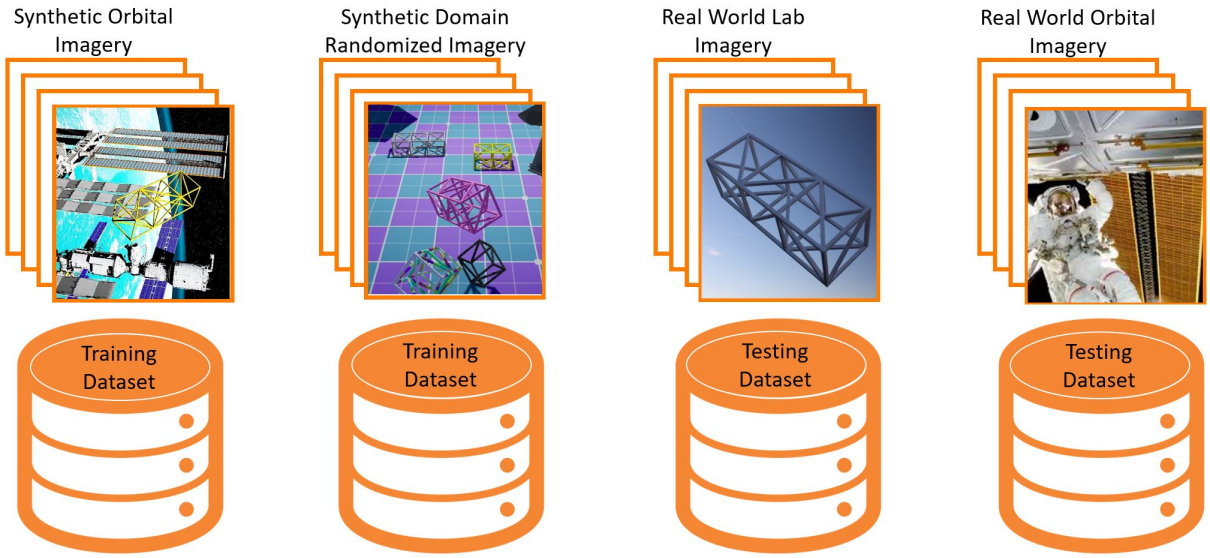

Figure 4.6: Division of imagery data into specific data sets

4.6 Evaluation Metrics

Average precision (AP) is a common metric used for evaluating a single class label when performing object detection. Mean Average Precision (mAP) is the standard for the evaluation of an entire network of n class labels. Both of these metrics are functions of precision and recall (accuracy metrics that consider misclassification and classification accuracy rates. Precision and recall are defined below in Equations 4.1 and 4.2 respectively where TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.2}$$

Average Precision is an area under the curve measurement of the Precision-Recall Curve (PR Curve) [or p(r)] defined in Equation 4.3, which serves as an input to Mean Average Precision defined in Equation 4.4.

$$AP = \int_0^1 p(r)dr \tag{4.3}$$

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{4.4}$$

Note, that the subscripts on Mean Average Precision scores (often 50 or 75) generally denote the IoU threshold (Figure 4.7) at which the score was taken. mAP(0.5) donates a 50% IoU while mAP(0.5:0.95) donates IoU thresholds, from 50% to 95%.

4.7 Flight Hardware

Flight compute hardware and communication protocol, were built using the companion computer infrastructure, utilizing a PID flight controller (in our case, the pixhawk flight

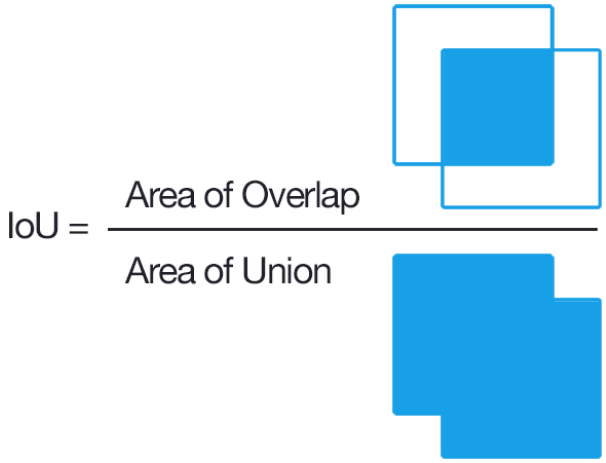

Figure 4.7: Intersection over Union evaluation metric

controller) for drone control. The flight controller is commanded by a more capable edge computing device such as a raspberry pi or Jetson Xavier to store and execute more computationally complex workloads such as path planning, computer vision, and sensor integration. The technical connections between these two boards are detailed below in Figures 4.8 and 4.9.

The first iteration of the SpaceDrones architecture [27] leveraged an onboard Raspberry Pi B+ computer due to its lightweight, low power consumption, and low cost. At only 5 volts, the power requirements could be satisfied via GPO pins, simplifying the drone's power distribution system while preserving limited power resources provided by an onboard battery. The hardware solution is cost-effective, costing under 100 USD, and sufficient for many drone use cases. However, the demanding computational requirements of most neural networking applications are beyond the capabilities of nearly all such single board computers. In the past, drone sensor data such as imagery and point clouds could be sent "off drone" and interpolated using higher-end compute hardware not subject to the weight and power constraints of drone hardware. Instructions interpolated by a neural network architecture derived from drone sensor data could then be beamed back to the drone for execution. This off-drone

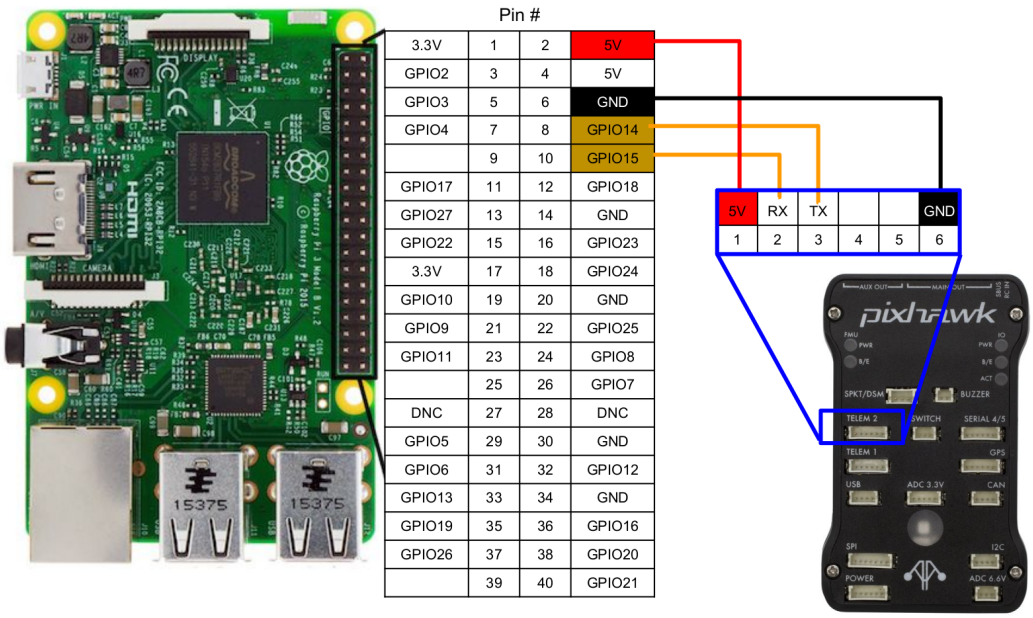

Figure 4.8: Pixhawk to Raspberry PI 3 GPIO Pin out [27]

compute capability is useful for a wide range of use cases but unrealistic for truly autonomous applications such as deep space exploration and tasking and any other applications requiring real-time decision making or sensor awareness/environmental interpolations.

Nvidia's recent introduction of the Jetson [2] line of single-board edge computers capable of neural network tasking meet mass constraints to be flown as a companion computer within the SpaceDrones Architecture with a supplementary power distribution system capable of providing the required 19 volts. Companion computer protocol connection for this architecture is detailed in Figure 4.9. This configuration enables us to perform onboard machine learning/computer vision tasks without the need for an additional "off drone" computer, enabling true autonomous tasking.

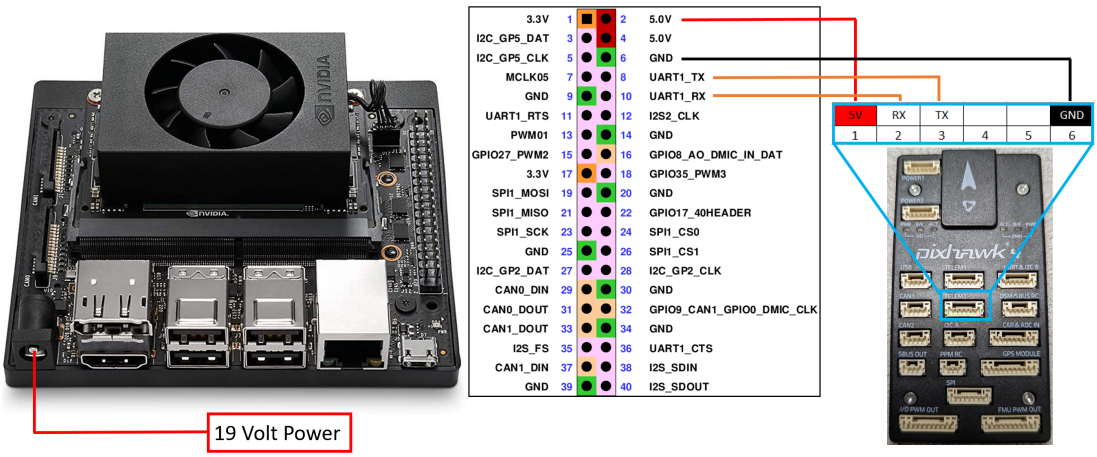

Figure 4.9: Pixhawk to Xavier NX GPIO Pin out

4.8 SpaceDrones 2.0 Architecture

The SpaceDrones architecture is primarily built on top of the Robotic Operating System (ROS) [47]. The ROS framework allows us to send and receive sensor data, object position data, and drone diagnostic data over a local area network via the publishing/subscribe functions. This local network solution also enables us to monitor neural network and associated drone performance in real time during flight instead of relying on saved data local to the drone storage device.

The SpaceDrones API is capable of integrating two position tracking systems (Opt-Track [22] and Steam VR [4]) for drone control, testing, and safety. Both systems utilize the MAVROS [1] Package library for companion computer control over an onboard PID flight controller, allowing tracking systems to update position data will all nodes on a ROS network.

The first of these tracking systems is the Opti-track, an “outside in” tracking system capable of tracking rigid and flexible structures via retroreflective trackers returning Infra-red light. This method allows for almost any object regardless of shape to be accurately modeled and tracked. This data was set to broadcast over a local area network at 60 frames per second

(60 FPS), and pulled down by both the drone and ground station computer, detailed in Figure 4.10. Full system development and details of this capability can be found here [27].

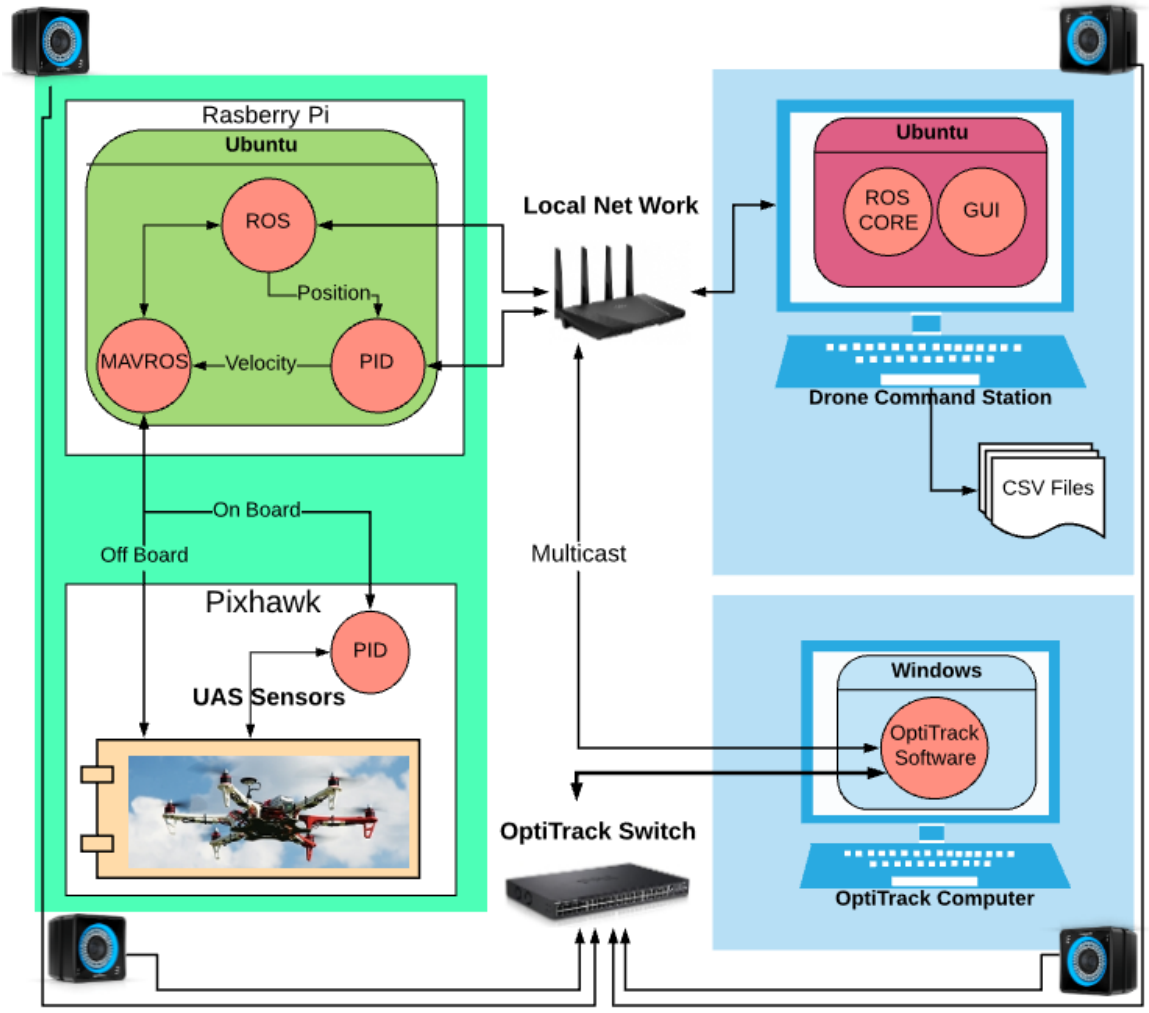

Figure 4.10: SpaceDrones Opti-Track tracking and control architecture

The second of these tracking systems is Steam's lighthouse tracking suite [**?**], and is the solution used in the methodologies described in this work. Initially designed for VR asset tracking, this is an "inside out" tracking solution that uses a proprietary tracking sensor with onboard IMU and one or more photodiode tracking bay stations, providing a position accuracy of 0.3mm with the tracker Orthogonal to the bay station and 0.5mm when in the direction of the bay station. VR system architecture is illustrated in Figure 4.11. This

method is more limited in its ability to track nonrigid objects when compared to "outside in" solutions such as Opti-Track, but this is a none constraining factor in this particular use case due to the rigid materials required to support drone flight. Non rigid structures are not required. The Steam VR system has the added benefit of being far cheaper to acquire and operate than the opti-track system as well as being easier to deploy to the field due to fewer hardware requirements and Opti-track's tendency for IR sunlight to corrupt tracking in outdoor environments.

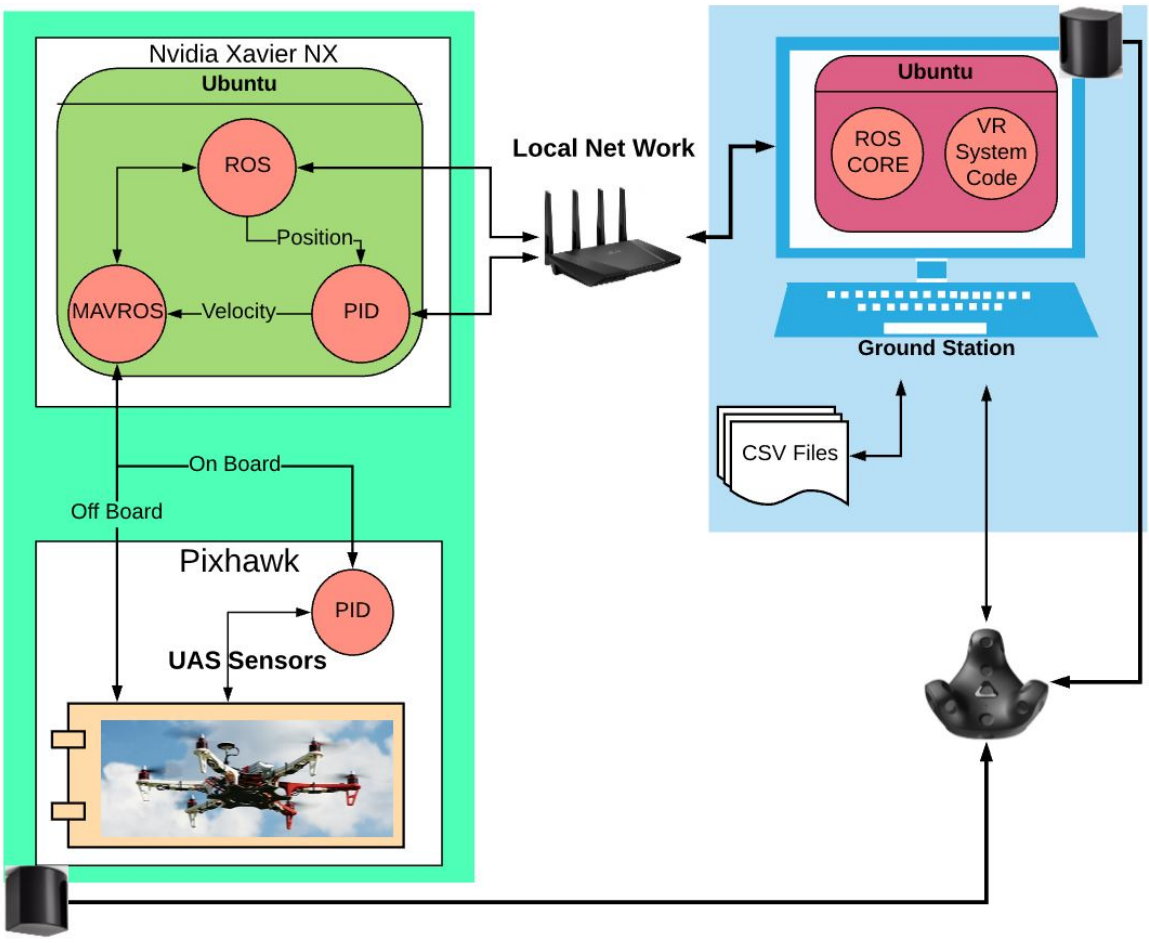

Figure 4.11: SpaceDrones VR tracking and control architecture

4.8.1 Target Localization

While YOLO can detect and output the 2D target position in a picture frame, any robotic operations involving object manipulation will require a 3D location of the target relative to either the camera body frame or local inertial frame. Here an additional LiDAR system was implemented to provide distance data for 3D localization.

The LiDAR system used for this study is an all-in-one Intel RealSense L515 system that includes a RGB (Red-Green-Blue/color) camera, a solid-state LiDAR module, and onboard Inertial Measurement Unit (IMU). The RGB image stream can be fed directly to the YOLO detection algorithm for object detection, while the depth image stream can provide point cloud data for object localization detailed in Figure 4.12.

The default Python script originally used to run the YOLO v5 neural network (detect.py) flying onboard the drone was modified to publish string message streams to the ROS network. The message stream contains basic detection results from each frame, which includes the name of the object, the confidence, and the relative 3D location of the object from the perspective of the camera. The SpaceDrones API has a built-in re-projection function that takes 2D location of the object provided by the YOLO detection program, as well as the distance from the camera to the specific depth pixels corresponding to the object location, and outputs the 3D location of the target voxel (volumetric pixel) from the point cloud. The string message can then be interpreted further based on the location of the camera relative to the local inertial frame and completes the target localization process for each object detected. The software diagram depicting this software and sensor interaction can be found in appendix B Figure [B.1].

Other than being geometrically similar and being difficult to distinguish, truss sections presented another problem. Given the porous structure of truss segments, a LiDAR scan

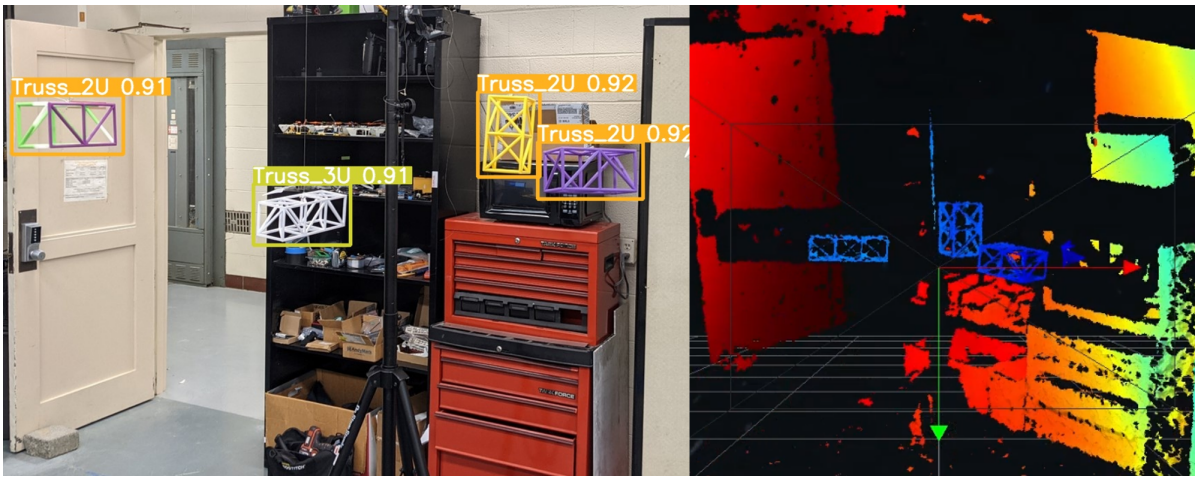

Figure 4.12: RGB Camera and LiDAR Point cloud integration senor data side-by-side

can pass through the center point of a given voxel returned from a bounding box position, resulting in a depth measurement that is past the object we are trying to capture. To solve this problem, an internal box consisting of 70 percent of the original bounding box is scanned with a LiDAR sensor. To reduce computational complexity, every other pixel is skipped. The LiDAR return that is located closest to the center point of the original bounding box limits is taken as the depth measurement, which is then used for capture operations. Sensor integration order of operations is illustrated in Figure 4.13.

This method was tested in flight by scanning a test volume in the lab and a returning the 3D localized positions of objects identified by the CNN. The results of this localization method are detailed in Figure 4.14. Identifying truss sections at five meters results in an average 3D position error of 0.17m in the direction of drone movement, and 0.76m in the perpendicular direction. Stationary error when both the camera/drone reference frame and object of interest are stationary is reduced to 0.26m in the perpendicular axis.

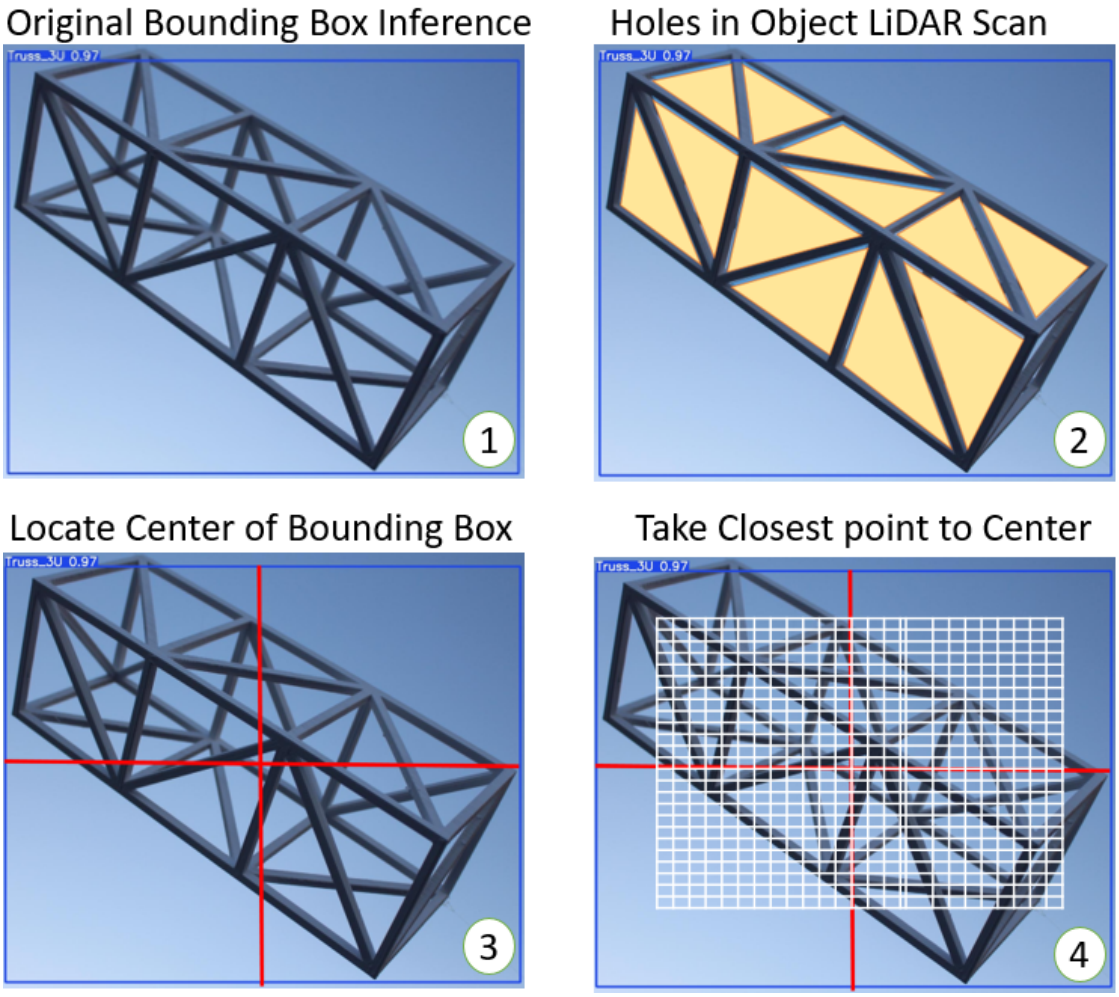

Figure 4.13: Porous structure LiDAR Depth Problem

4.9 Drone Design

4.9.1 Drone Body

The Space Drones physical chassis has gone through many iterations of quad and hex copters, including off the shelf (OTS) solutions such as DJI flame wheel 450 and 550 Chassis, however, no OTS bodies provided the internal volume needed to accommodate both a pix hawk flight controller and the new Jetson Xavier computer. This shortfall necessitated the development of the in-house solution illustrated in Figure 4.15. The mass requirement for larger

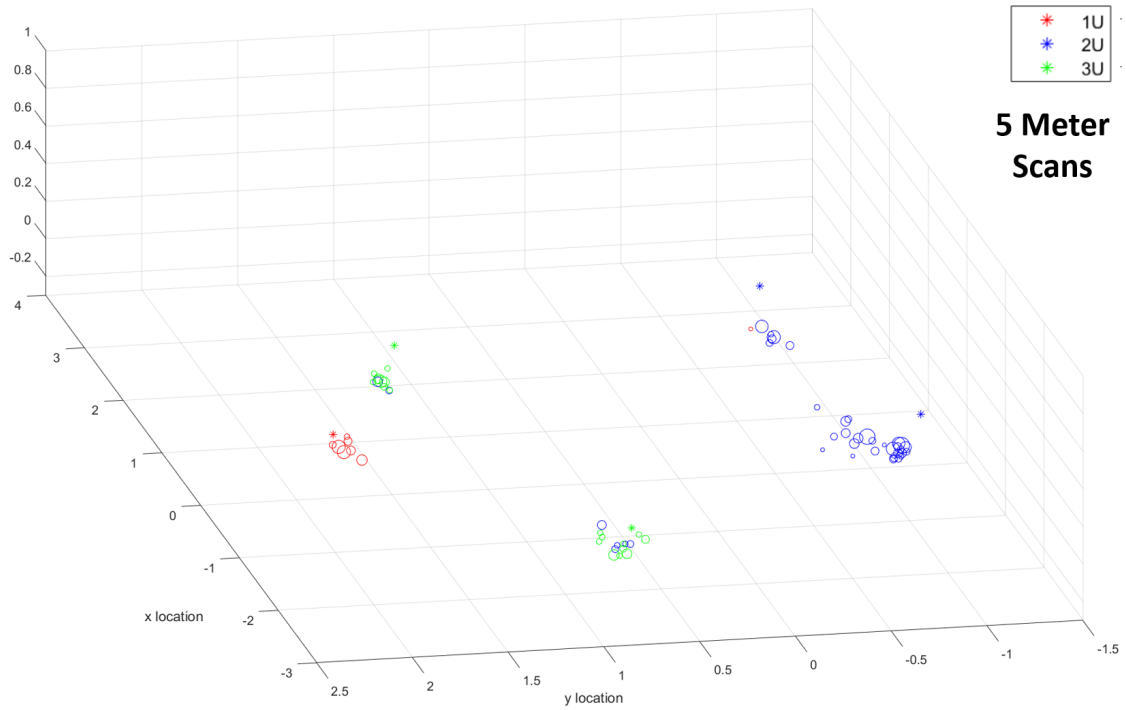

Figure 4.14: 5 meter LiDAR scan localization result when integrating Computer vision bounding box data with LiDAR point cloud data.

computers, two onboard sensors (Realsense L515 camera/LiDAR and VR tracker), as well as an under-slung robotic arm, required a six rotor body. The entire body was 3D printed allowing for rapid prototyping and parts replacement in the event of a crash.

4.9.2 Drone Power Distribution

The increased power consumption of the Jetson Xavier, coupled with the need to power all six motors, the radio controller, and camera systems at the same time, pulled more voltage them most off-the-shelf batteries could provide. Thus, two batteries connected in series delivered a combined 18 volts to the drone for flight operations. An additionally third independent 12-volt battery was added to power the under-slung robotic arm.

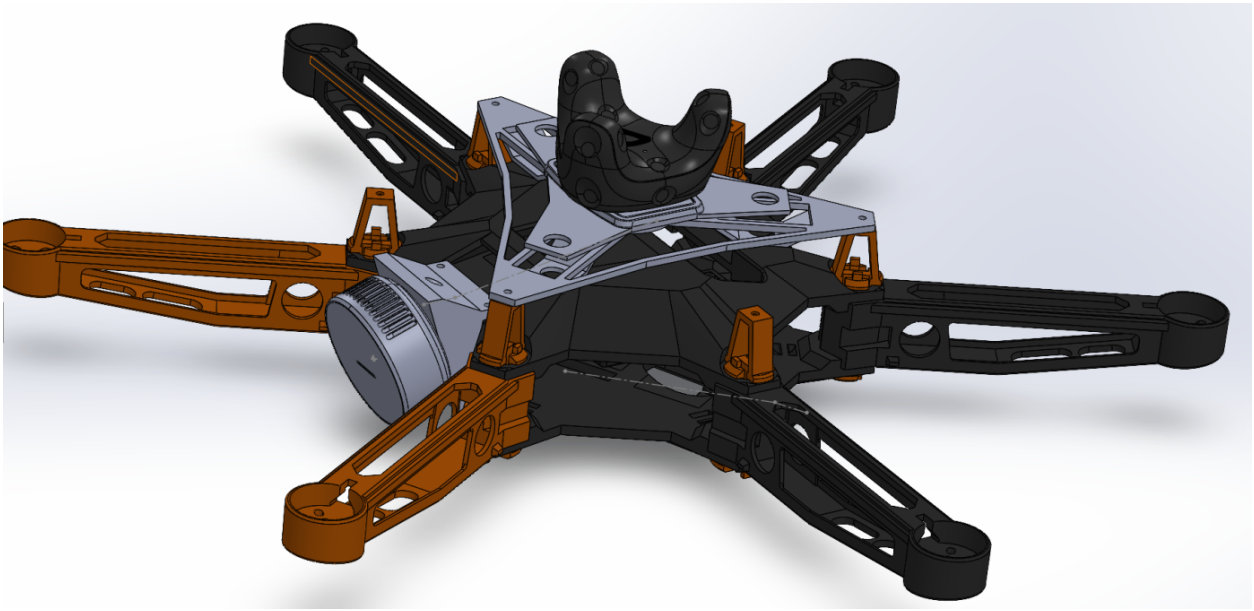

Figure 4.15: Drone frame with onboard tracking, computer vision, machine learning computation, LiDAR sencing, and VR tracking

4.9.3 Drone Reference Frames

Drone Body Frames

The overall world reference frame is defined by the VR tracking system, and subsequent "world zero" can be defined to be anywhere by the user. Drone zero is defined by the VR Tracker located on the top of the drone as it moves through 3D space. All drone transforms are derived from this zero. In order to locate objects in space relative to the drone body frame, a known camera offset of 59.8mm in the Y direction, 117.58mm in the Z direction, as well as a 20-degree rotation along the X-axis defines the camera frame relative to the drone frame illustrated in Figure 4.16.

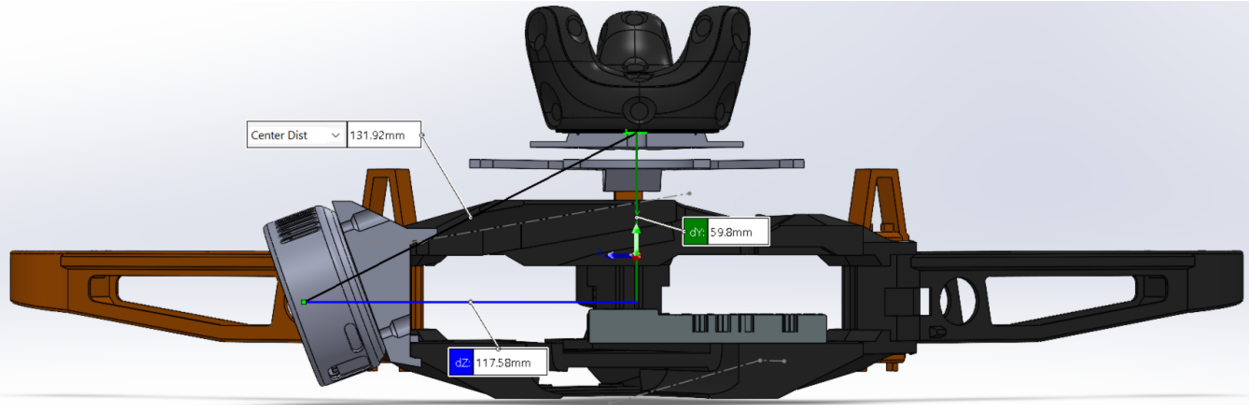

Figure 4.16: Drone Camera frame offset

4.9.4 Drone Mounted Robotic Arm

The robotic arm is a PincherX 100 [3] manufactured by trossenrobotics, mounted in an underslung position under the drone chassis illustrated in Figure 4.17. This 4 degree-of-freedom (DOF) arm has a reach of 300mm and a total rotating span of 600mm, giving the robotic arm a total workspace that covers 112 percent of the drone's 533.4mm wingspan. Drone Manipulation and end effector position is governed by imagery and LiDAR sensor input data that has been interpolated by the onboard neural network detailed in the target localization section. Once an object of interest has been identified, its 3D location is used to drive the applicable inverse kinematics required to execute a successful object capture.

Robotic Arm Airborne Inverse Kinematics

The robotic arm's base frame is coubled to the drone body frame as it flies through a space and maneuvers into place for object capture by the drone. In turn, the arm is governed by the overall world frame tracked by the VR world space diagrammed in Figure 4.18. Once an object is identified by onbord computer vision and LiDAR integration, the drone controller maneuvers to the object designated for capture, placing the drone within reach of the robot

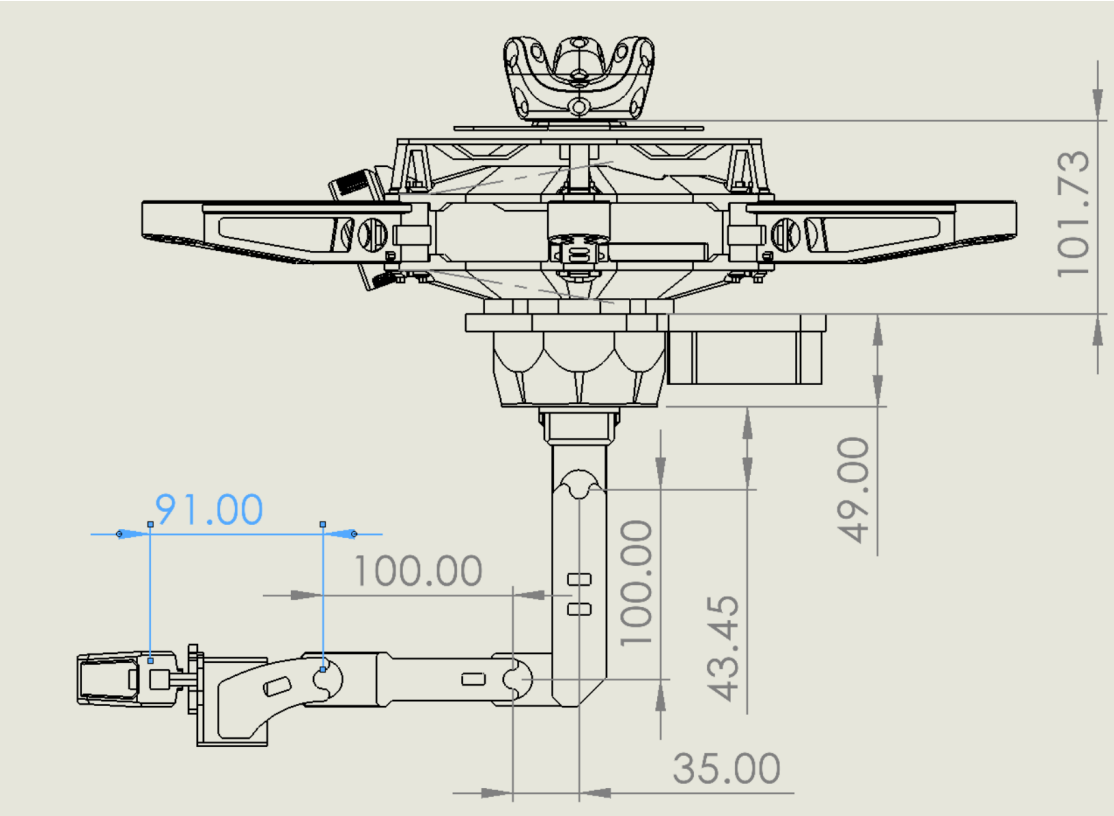

Figure 4.17: Drone Arm link measurements (mm)

arm's workspace, any error between the drone's position and the object's position will be corrected with the reach of the robotic arm. Once in position, an inverse kinematic (IK) controller calculates the distance between the object and the current end effector position, then maneuvers the end effector to execute capture.

The motion of the robotic arm is governed by three revolute motor joints that drive end effector position, an additional motor controlls gripper open and close functions. Desired robotic arm motion is calculated using the Jacobian iterative inverse kinematics method defined in equations 4.5 through 4.9.

$p_i =$ the vector from the origin of the world coordinate system to the origin of the i-th link coordinate system.

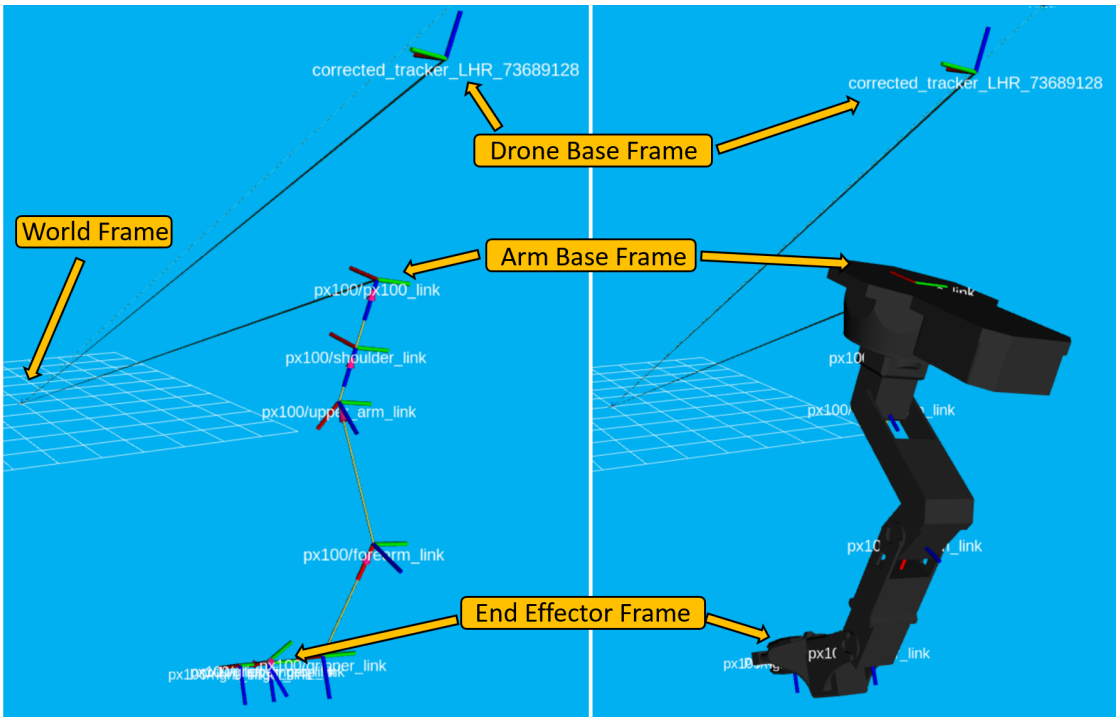

Figure 4.18: Drone Arm Base Frame relative to other relevant world frames. Raw Links [Left, with arm model animated [Right]

$p =$ the vector from the origin to the end effector end

$z =$ the i-th joint axis

$J =$ Jacobian Matrix

$\Delta p =$ End Effector Position

$\theta = \theta_1, \theta_2, ...\theta_n$ (All joint angles)

$$\begin{bmatrix} j_{Pi} \\ j_{oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{i-1} \\ \mathbf{0} \end{bmatrix} & \text{for a prismatic joint} \\ \begin{bmatrix} z_{i-1} \times (\mathbf{p} - \mathbf{p}_{i-1}) \\ z_{i-1} \end{bmatrix} & \text{for a revolute joint} \end{cases} \tag{4.5}$$

$$J(\theta) = \begin{pmatrix} j_{P1} & & j_{P_n} \\ & \dots & \\ j_{o1} & & j_{o_n} \end{pmatrix} \tag{4.6}$$

$$\Delta\theta = \theta_{\text{goal}} - \theta_{\text{current}} \tag{4.7}$$

$$\Delta p = \text{ Goal } - p_{\text{current}} \tag{4.8}$$

Equation 4.9 defines the necessary joint angles needed to achieve a desired end effector position Δp for a given starting position.

$$\Delta\theta = \mathbf{J}\,(\theta_{\text{current}}\,)^{-1} * \Delta p \tag{4.9}$$

The Jacobian matrix may not always be invertible, in which case calculating a pseudo inverse via singular value decomposition is necessary.

4.9.5 Drone Limitations

The complete integrated system is illustrated in Figure 4.19. The platform is stable even when the robotic arm is at full extension due to the under-slung configuration. However, the robotic arm does have a working payload limit of 50g. This limitation can be increased with more numerous or more powerful motors, but this will reduce the current carrying capacity of the overall drone of one kg and reduce the current battery flight time of about 3.5 minutes. When all three batteries are attached, the total drone weight is 3.4 kgs. Naturally, all these

limitations will gradually reduce with larger drone platforms with eight or more motors and ever-larger lifting surfaces.

Figure 4.19: Depiction of the integrated Drone, Camera, Robotic Arm System

4.9.6 Sensor Limitations

It should be noted that applying computer vision to space-based applications will be incredibly useful, but such capabilities will be highly dependent upon the sensors used when applied to a specific use case. For example, using RGB camera sensors for in-space assembly tasks may not function during high saturation events like those depicted in Figure 4.20, that results in sensor "washout". This can be solved by using dynamic sensors capable of adjusting parameters such as aperture and shutter speeds or using sensors less susceptible to such fluctuations.

4.10 Virginia Tech CUBE

Wrapping up all the previously mentioned computer vision and robotics methodologies and merging the rapid and dynamic world manipulations offered by the synthetic environment

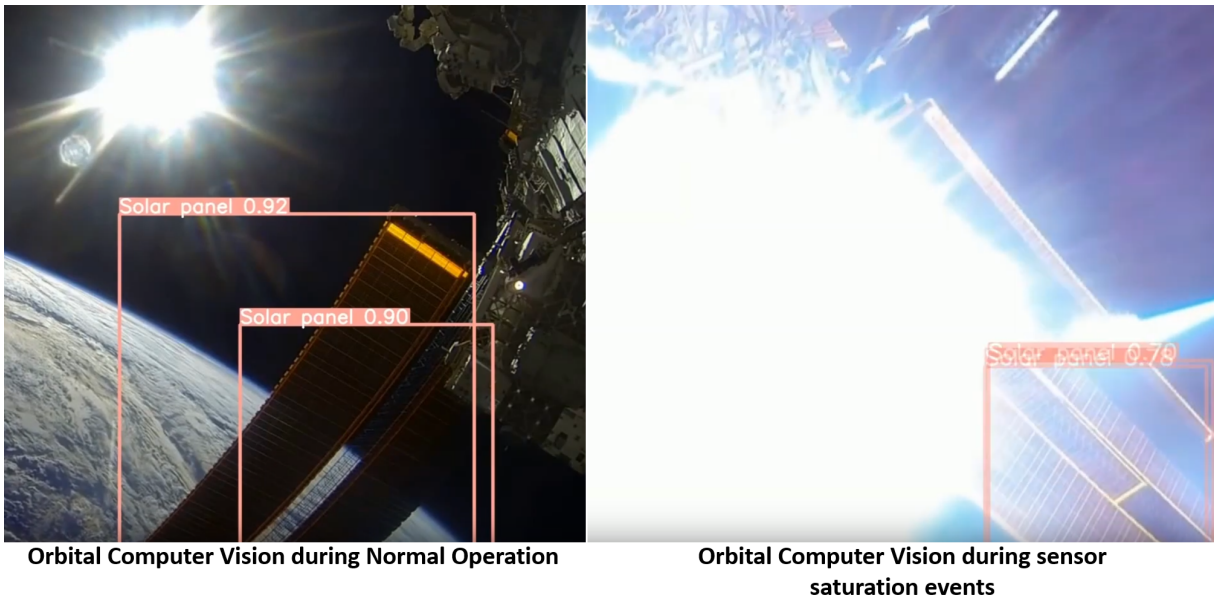

Figure 4.20: RGB Computer Vision during sensor saturation events

with real-world hardware-in-the-loop testing needed before hardware and software deployment. The Virginia Tech CUBE [39] (pictured in Figure 4.21) is capable of surround visual and audio projection, enabling enables nearly 360 surround projection of pre-rendered or live Unreal Engine levels on 9 foot high screens with very low shadow projection distances at a resolution of 1920x1200 at 13,000 lumens, effectively allowing optical computer vision testing and development. Standing four stories high, consisting of 21,000 cubic meters of indoor space, this facility also allows large-scale system testing. By projecting the desired synthetic environment, we were able to repeatably and reliably test machine learning architectures in a physical space while projecting orbital scenes, then use that computer vision sensor data to manipulate real-world hardware within the same room - in our case, testing drone and robotic arm controls relative to an object of interest using synthetically inferenced sensor data

Figure 4.21: onboard drone Computer Vision testing inside a physical/virtual simulation projection at Virginia Tech's CUBE facility [39]

Chapter 5

Results

5.1 Synthetic Data vs Real World Results

To answer Research Question 1, "Can synthetic data be used for reliable real world object tracking?" synthetic ISS Imagery was generated and compiled from the unreal engine into a training data set used to train the neural network as described in the data set division section. The neural network will never see a real-world image during the training phase, only during testing. Real-world objects of interest are broken out into two different testing data sets. The first testing data set consists of imagery collected from real-world 3D printed trusses, the identical trusses synthetically rendered and captured in the unreal engine. The second testing data set consists of solar panels collected and compiled in an orbital environment from several open-source databases, including NASA images and google images.

The ultimate goal of this method is to eliminate the need for real-world training imagery, solely or mostly relying on synthetic images. Thus, to form the basis of control, both synthetic imagery and real-world imagery data sets were evaluated against themselves in order to find the mean average precision performance within their own perspective domains.

When evaluating a synthetic orbital environment data set against itself via the YOLO v5 Neural Network architecture results in an mAP(0.5) of 94% and mAP(0.5:0.95) of 77%, respectively (Figure 5.1), which is exceptionally high performance. However, this performance

is to be expected as a certain level of neural network memorization or overfitting occurs when training within the confines of a particular single domain environment.

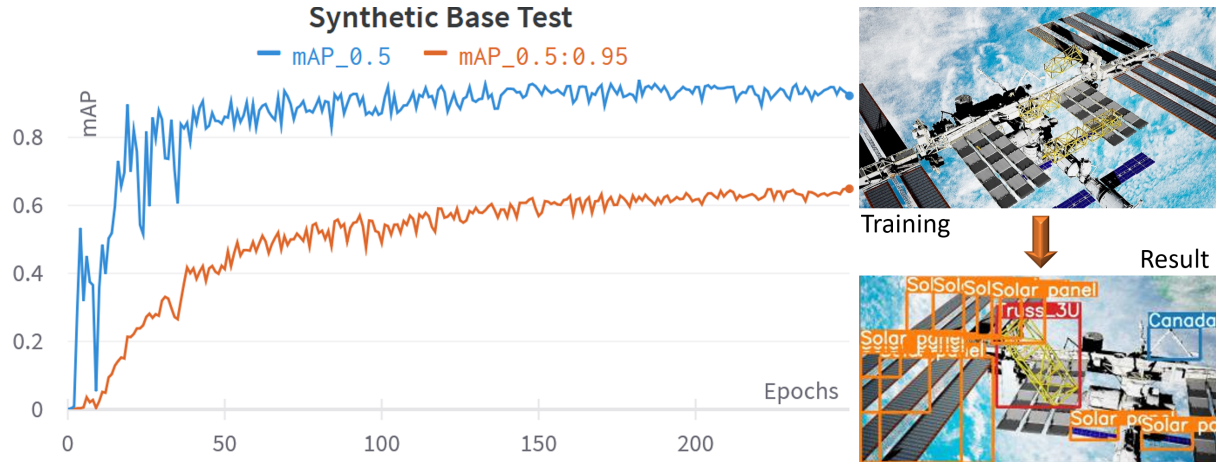

Figure 5.1: Synthetic orbital data set self evaluation results showing relatively high mean average precision over 250 training epochs. Establishing a baseline performance metric.

Evaluating the real world a truss deadset against itself yielded an mAP(0.50) of 98% and mAP(0.5:0.95) of 90% (Figure 5.2), which is nearly perfect across all metrics including precision, recall, and confidence. Ideally all of the future CNNs would achieve this is the level of precision when evaluating the synthetic data sets and domain randomization methods below.

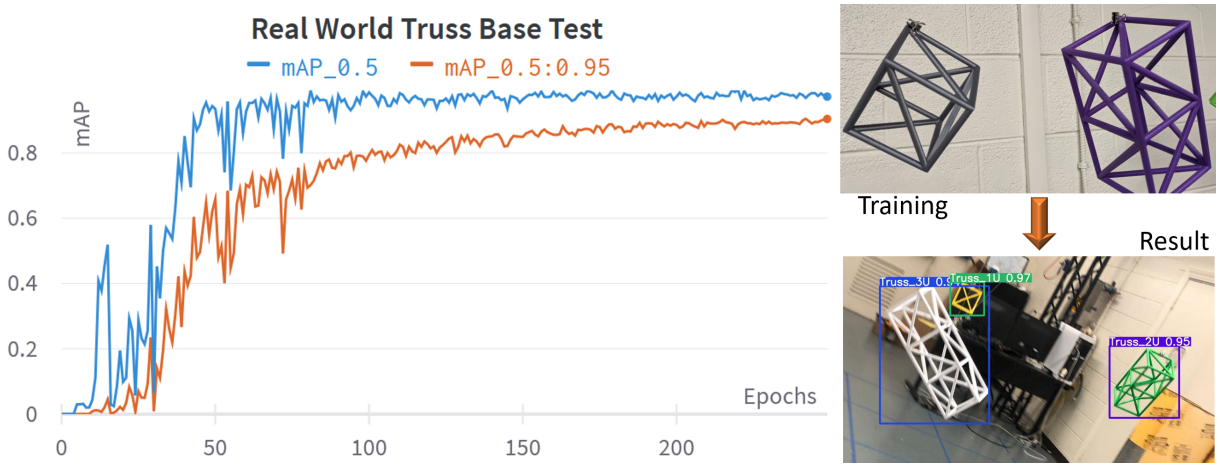

Figure 5.2: Real World Truss Data Set self evaluation mean average precision (mAP)

The resulting "F1" curve is displayed in Figure 5.3 depicted on the left side, is derived by combining both precision/confidence and recall/confidence curves into a single chart. More area under the curve demonstrates better neural network performance. The resulting confusion matrix (shown on the right side) further breaks down the percentage of true positives (TP), false positives, (FP), true negatives (TN), and false negatives (FN) for each class/object of interest. Numbers along the matrix diagonal indicate that an object is being identified correctly. Numbers outside the diagonal indicate a miss-classification and subsequent loss in performance, either as another object or background. The bottom row represents background false negatives, while the last column represents background false positives. These metrics aid both researchers and observers to see where the CNN model is performing well and what areas are bringing down overall network performance.

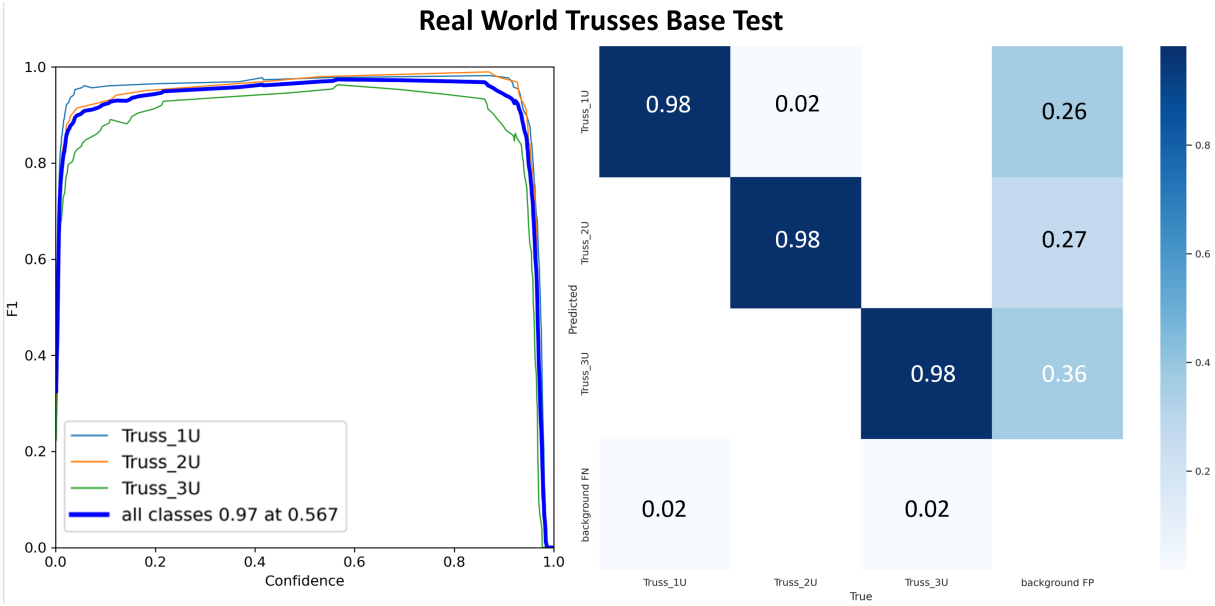

Figure 5.3: Real World Truss Data Set self evaluation F1 curve (left), confusion matrix (right)

Due to the relatively large amount of imagery publicly available of solar panel systems currently deployed in orbit, real world imagery was also evaluated against itself, yielding similar results of mAP(0.50) of 96% and mAP(0.5:0.95) of 78%. This provides us with a real

world baseline when evaluating the use of synthetic imagery with real world objects already in orbit with acceptable accuracy.

With base case performance established with in their on perspective synthetic and real world domains, the next step is to determine if imagery from a synthetic domain can be used to train a network that can be applied to a real world domain. Evaluating synthetically generated imagery of truss sections against real world truss sections resulted in a mAP(0.5) of 51% and mAP(0.5:0.95) of 38% (Figure 5.4) When compared to other results in computer science literature [54] [55]. This is a performance metric that may be deemed usable for other computer vision applications, how ever given the high risk operational environment, space agencies such as NASA, ESA, and SpaceX will more than likely require more robust an accurate models before in trusting major mission tasks to such a autonomous system. However, model performance can be improved further using methods below.

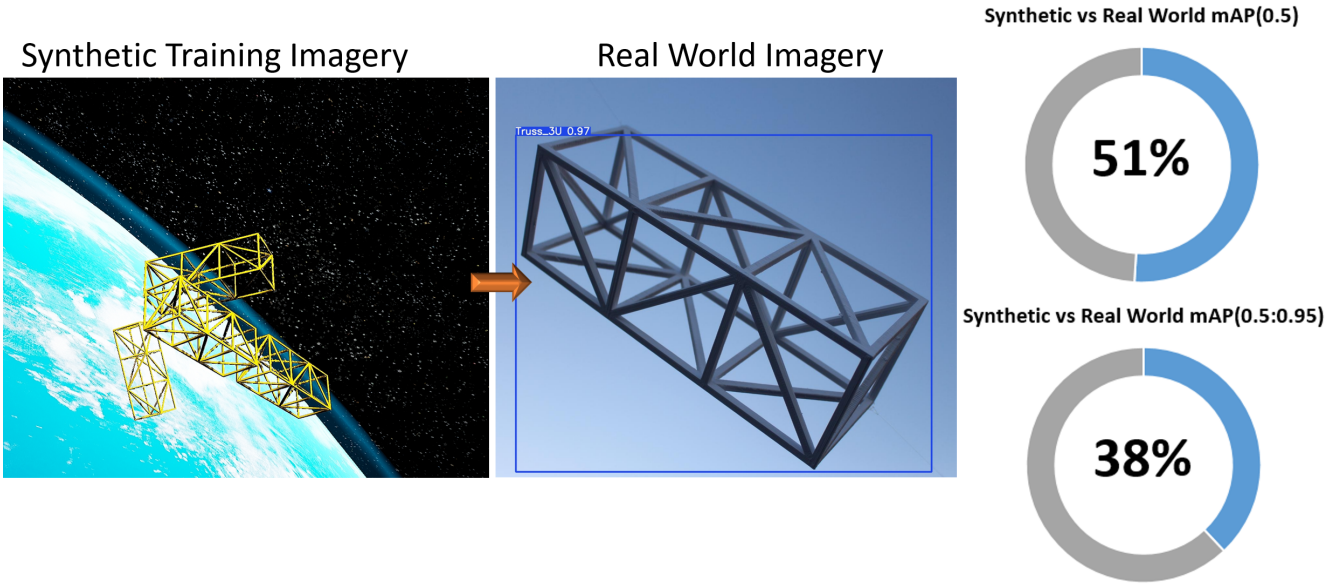

Figure 5.4: Mean average precision result when training with synthetic imagery and testing against real world imagery

The same experiment was conducted evaluating a neural network trained on synthetic solar panels against real world orbital solar panel imagery collected from open source repositories

such as google images. The experiment was far more favorable, yielding a result of mAP(0.5) of 0.76 and mAP(0.5:0.95) of 0.69. Given the nonporous, uniform, and often similar color texture inherent to solar panel design, neural network transfer between the synthetic and real world domains is much easier for this particular object of interest when compared to porous structures such as trusses or other types of construction scaffolding material. This fundamental challenge is addressed using the techniques outlined below.

The metric of mean average precision can be a little misleading by itself because it does not explain the capabilities of the model trained, only the confidence in a particular model when attempting to identify classes within a given frame (in this case truss sections). For example, mean average precision alone does not define if a model will perform well when alterations are made to the environment or object. These differences in model capability are often eliminated by evaluating models on standardized data sets, enabling a one-to-one comparison with other model. However, since standardized data sets for this space problem do not exist, direct comparis are not possible. For example, non-standard objects of interest such as damaged, discolored, distorted, or an object not meeting manufacturing standards a computer vision model is trained to recognize will degrade any systems ability to perform autonomous tasking. Introducing nonstandard structures like Figure 5.5 are often segmented or misclassified by traditionally trained computer vision models, especially if those models are transitioning across domains. Leveraging the domain randomization methods detailed below will also enable models that are more capable at correctly identifying these objects.

5.1.1 Domain Randomization Results

To make a more robust model to address some of the previously mentioned issues we introduced domain randomized data sets into the CNN training data sets. Using pre-rendered

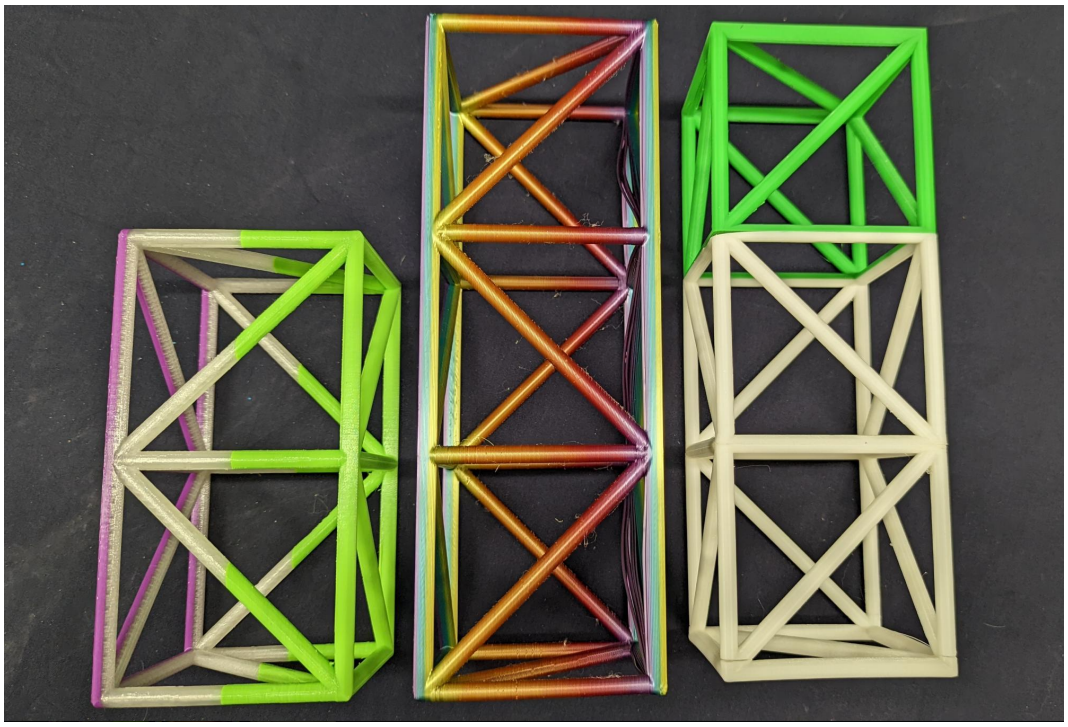

Figure 5.5: Non-Standard trusses used to evaluate neural network performance before and after introducing Domain Randomization

synthetic worlds we introduce distractors none uniform truss sections, backgrounds, and lighting conditions In a zero-gravity simulation environment at the preprocessing level. Introducing these domain randomized data sets into the YOLO v5 neural network architecture yielded a mAP(0.50) of 56% and a mAP(0.50:0.95) of 43%, an increase of 5% for both perspective mAP metrics (Figures 5.6 and 5.7). This performance increase shows that domain randomization can help bridge the reality gap when attempting to use synthetic optical imagery for real-world computer vision applications.

In addition to increasing mean average precision (mAP), domain randomized models are far more generalizable, capable of more complex inferencing tasks shown in Figure 5.8. Demonstrating reliable detection regardless of structure, color, or background. This ability will be essential when a satellite or robotic platform repairing space infrastructure or exploring celestial bodies encounters a situation that it has not encountered but can generalize and

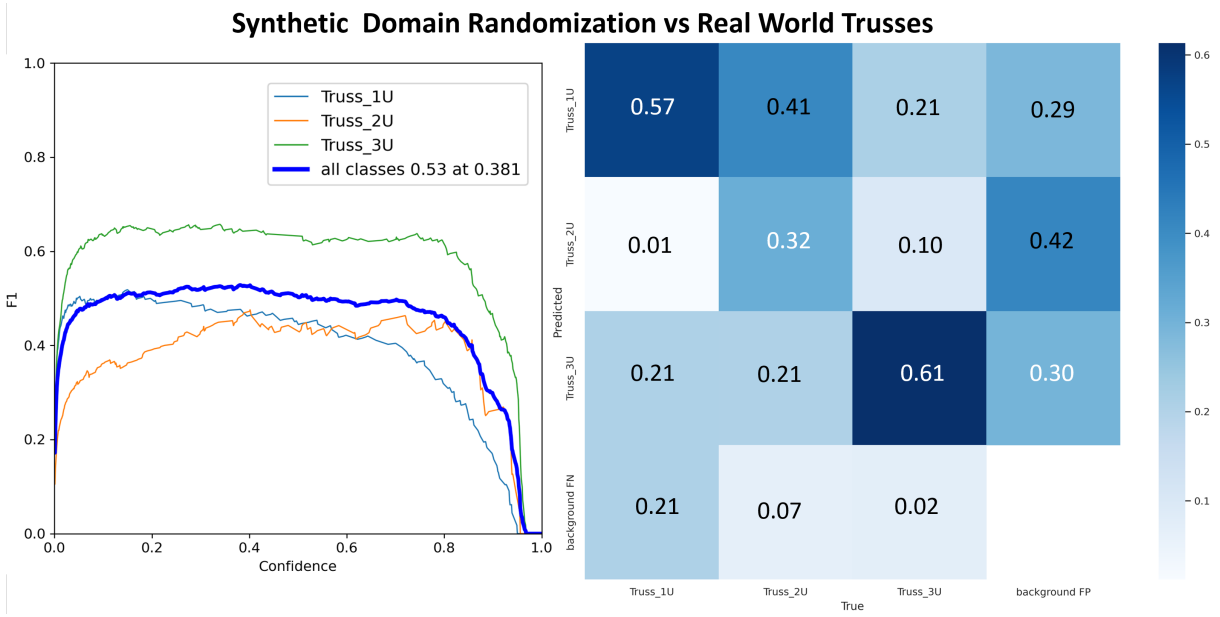

Figure 5.6: Domain Randomized Synthetic Data Set self evaluation F1 curve (left), confusion matrix (right)

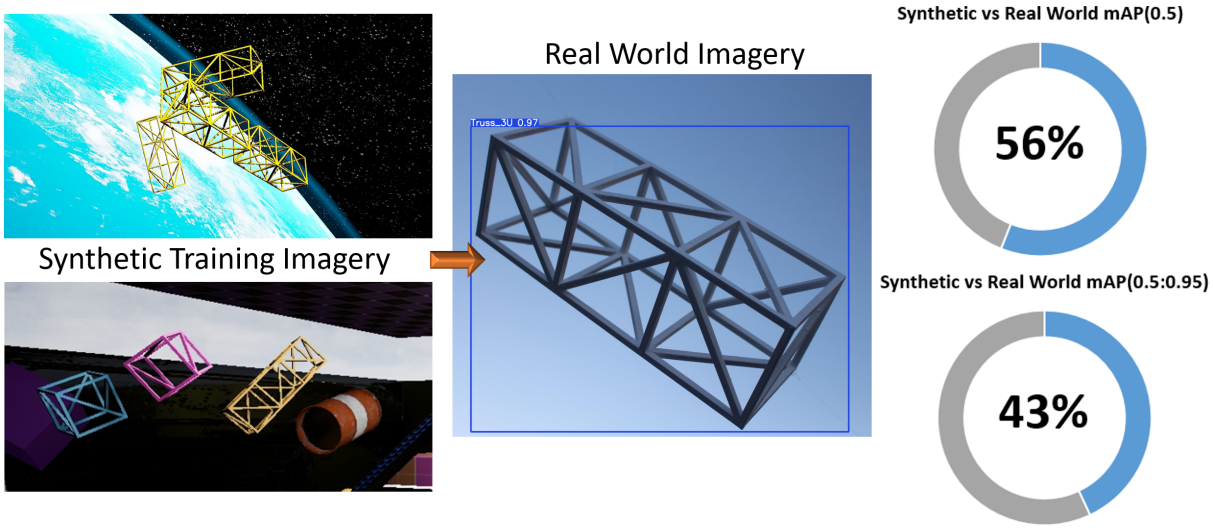

Figure 5.7: mean average precision (mAP) result when training with Domain Randomized synthetic imagery and testing against real world imagery

inference similar objects based on what it has been trained to see or do.

Finally, the foundation of this research assumes that a user has no imagery of the environment for which they are trying to train a computer vision architecture. However, what if a user

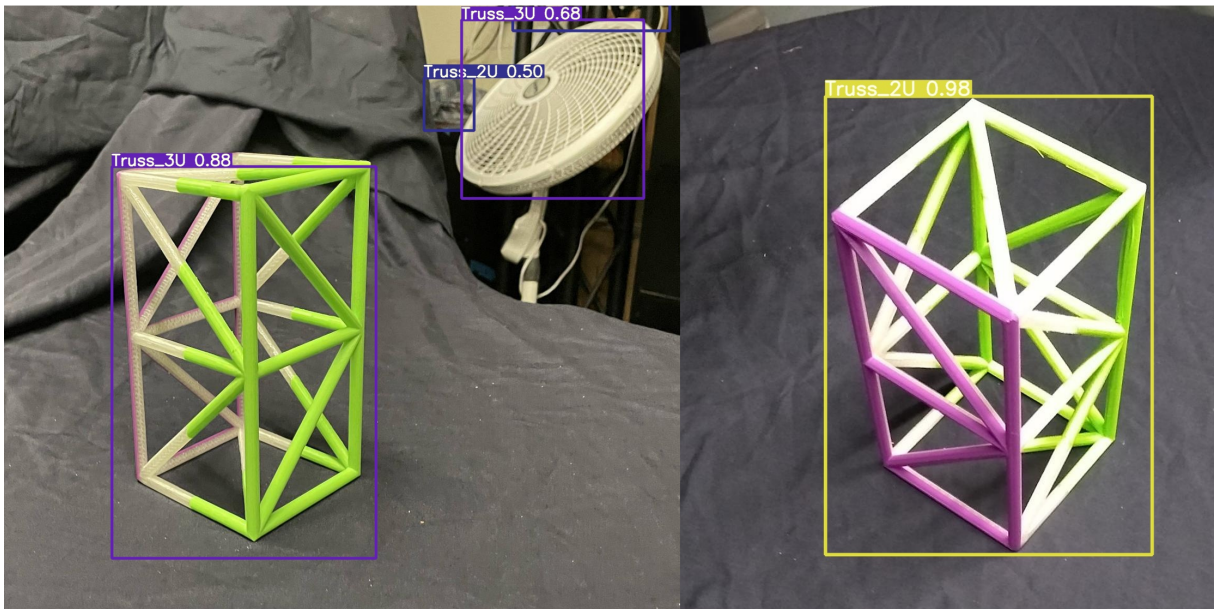

Figure 5.8: Domain Randomized vs None Domain Randomized results

has limited data of a particular environment? Can synthetic imagery fill in the gaps? To answer this question, we supplemented the purely synthetic training data set with real-world imagery at a ratio of 10% and then evaluated that new merged data set while ensuring that no real-world images were shared between the training and testing data sets. This small addition dramatically increased performance to mAP(0.50) of 87% and an mAP(0.50:0.95) of 73% (Figure 5.9), providing yet another data input solution to improve computer vision performance.

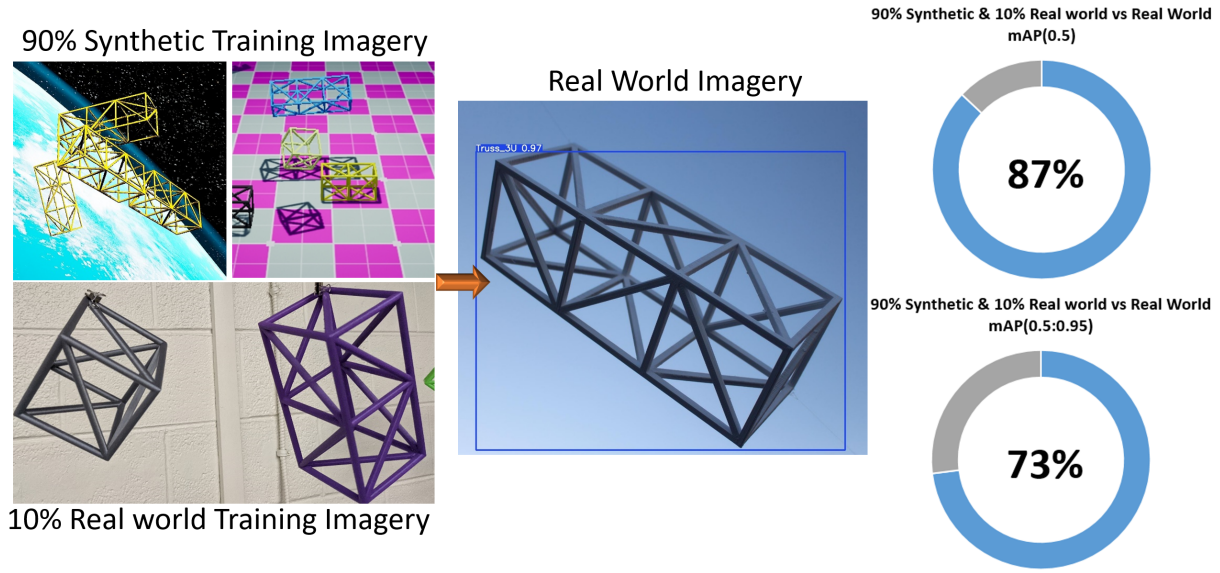

Figure 5.9: Mean average precision result when training 90% domain randomized synthetic imagery and 10% real world imagery, CNN evaluated against a real world data set

5.1.2 Database Augmentations

To further harden the model against rotational variations, random flips and rotations were applied to 30% of all images across all training, validation, and testing sub-data sets, thus resulting in an invariant orientation model that had little impact on mean average precision.

Additionally, lighting augmentation was also added by increasing and decreasing image brightness by 50%. However, it was discovered that this method reduces mean average precision dramatically. Some already low-light images become too dark to discern meaningful features. Thus we reduced image brightness augmentation to 25% and introduced a new data sets for both real-world and synthetic imagery into the stack that consisted of near dark images illuminated with supplemental lighting shown in Figures 5.10 and 5.11. Simulating light provided by a spacecraft or robotic arm. Due to the near-perfect lighting conditions provided by the supplemental lighting when an object of interest was in frame combined with nearly zero background clutter, these images resulted in some of the highest

mAP results, returning a mAP(0.50) of 78% and a mAP(0.50:0.95) of 69%.

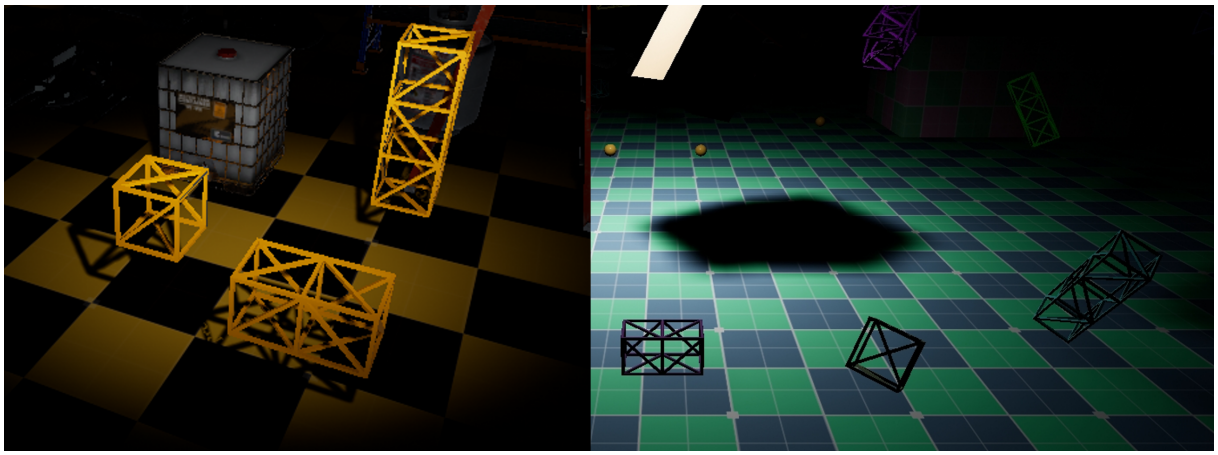

Figure 5.10: Synthetic Low light / supplemental lighting training images

This finding resulted in a hardware change, culminating in the addition of a 100 lumens flashlight being mounted on the drone for low light conditions. Of course, the range at which objects can be identified using computer vision sensing is directly proportional to the lumen output of your light source during low light conditions. However, more powerful lights consume precious power and may not be feasible.

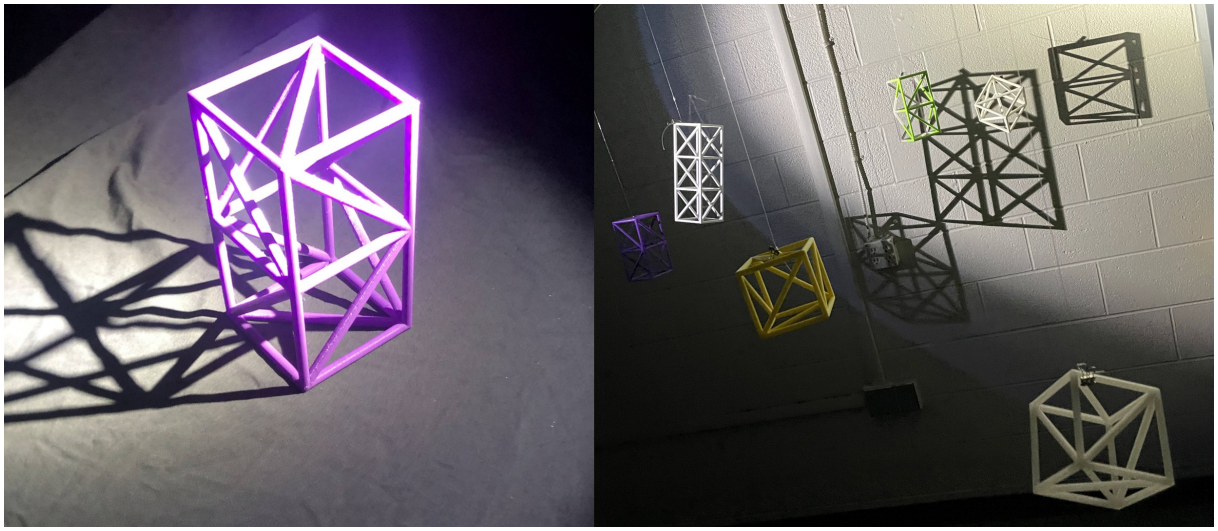

Figure 5.11: Low light / supplemental lighting testing images

5.1.3 Computer Vision Results Summery

With results within 11% mAP(0.50) of a traditionally trained neural network utilizing 100% real-world imagery, and within 4% mAP(0.50:0.95) of the synthetic baseline; domain randomized synthetic visual data is a viable solution for training a conventional neural network for real-world deployment if limited imagery data is available. Figure 5.12 and Table 5.1 summarizes computer vision performance starting with the 3 baseline performance metrics on the far left, moving to the right establishing cross domain performance metrics for both solar panels and trusses, applying domain randomization and then applying both domain randomization (DR) and a 10% real world data set to the training model.

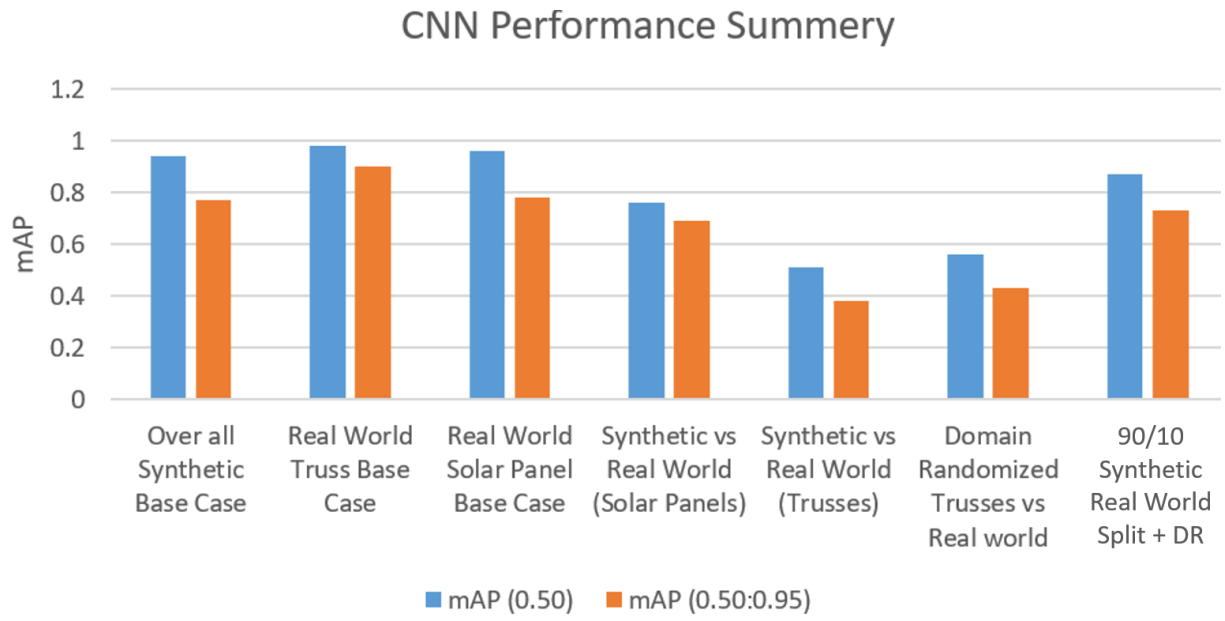

Figure 5.12: Overall summery of all major neural network performance evaluations across data sets. Synthetic Imagery can achieve results with in 11% mAP(0.50) of real world base lines, and with in 4% mAP(0.50:0.95) of the synthetic baseline when incorporating Domain Randomization and Augmenting with 10% real world data.

Other works [55] [63] have employed mixed real world/synthetic; as well as strictly synthetic data sets to improve neural network performance by as much as 13% when applying domain randomization to the self driving car computer vision problem. However, both of these

Table 5.1: Overall summery table of all major neural network (YOLOv5) performance evaluations across data sets.

YOLO Neural Network Performance		
Data set(s)	mAP_{50}	$mAP_{50:95}$
Overall Synthetic Base Case	94%	77%
Real world Truss Base Case	98%	90%
Real world Solar Panel Base Case	96%	78%
Synthetic vs Real World (Solar Panel)	76%	69%
Synthetic vs Real World (Trusses)	51%	38%
DR vs Real World (Trusses)	56%	43%
90/10 Synthetic Real World split + DR	87%	73%

studies had the advantage of evaluating against hundreds of thousands of road like car images within the COCO [37] and VKITTI [28] data sets. As of today, a purpose-built data set for autonomous orbital tasking for computer vision training and testing does not exist other than perhaps Stanford's SPEED data set [51] used for Spacecraft Pose Estimation. This paper provides an existing domain randomized orbital data set as well as the methodologies to generate and test future data sets for future applications.

Additionally, identifying geometrically similar objects across different sizing SKUs (i.e. 1U, 2U, 3U trusses) is a significantly harder problem to solve when compared to identifying cars on a road surface. It is likely that construction elements for orbital assets and celestial habitats such as Mars will be made up of modular designs to simplify the process, thus a computer vision architecture will need to be able to routinely tell the difference between similar parts during the assembly process.

5.1.4 CUBE Computer Vision Results

All the previous results and capabilities were integrated into a single hardware/software in the loop simulation environment for optical/visual machine learning decision-making validation and testing within the CUBE environment illustrated in Figure 5.14. Placing real-world

mock-up models and objects in the center and synthetic environments projected to the surrounding screens, resulting in a unique optical visual simulation space. In regards to neural network performance, the particular simulation space resulted in a drop in the hybrid 90/10 CNN Model(synthetic/real-world) stack of only 4% and 5% respectively, yielding a mAP(0.50) of 83% and an mAP(0.50:0.95) of 68% (Table 5.2) at approximately 20 frames per second (FPS) in flight running on the Jetson Xavier board. The loss in performance is due to the inherent curvature of the screens not retaining true geometric proportions. However, training models within the CUBE that are robust to this simulation anomaly should render this a non-issue. The drone camera point of view (POV) during flight operations in the CUBE is shown in Figure 5.13. Drone flight video with onboard computer vision can be found at SpaceDrones CUBE Flight.

Figure 5.13: Drone camera POV (point of view) running computer vision on objects of interest against a synthetic orbital background, Video illustration

Regardless, this methodology proves that such an environment is both practical for optical machine learning computer vision testing and validation, and can be used to leverage real world HIL testing for physical hardware such as platforms, sensors, robotic arms, and their accompanying software within a synthetic space.

Figure 5.14: Virginia Tech CUBE Optical Space Simulation

Table 5.2: Table of CUBE computer vision Results

YOLO Neural Network Performance		
Data set(s)	mAP_{50}	$mAP_{50:95}$
CUBE Computer Vision Results	83%	68%

5.1.5 Robotic Arm Capture Sequence

Using a synthetic environment of the user's choice in the case of orbital truss sections in and around an existing space station, a SpaceDrone is deployed to perform autonomous pick and place operations solely using synthetically trained onboard RGB and LiDAR sensor data to perform real-time neural network inferencing to identify objects of interest. As described in the methodology chapter, this integration is done by overlaying bounding box X and Y positions with LiDAR depth data to form a complete 3-dimensional picture of an object or

objects relative to the drone. Once relative location is determined, the capture sequence is initiated, maneuvering the drone to the proximity of the object, where the robotic arm is released from the stow position and inverse kinematics (IK) solutions drive the arm to the desired capture position at which point the end effector gripper is closed, completing the capture sequence. The object capture sequence is systematically accomplished via the following four steps illustrated in Figure 5.15. 1) Perform Object classification and localization. 2) Move to object and deploy robotic arm for object capture. 3) Stow captured object for transport. 4) Transport object to the desired location for release or placement. This capture sequence is better illustrated in video format found here.

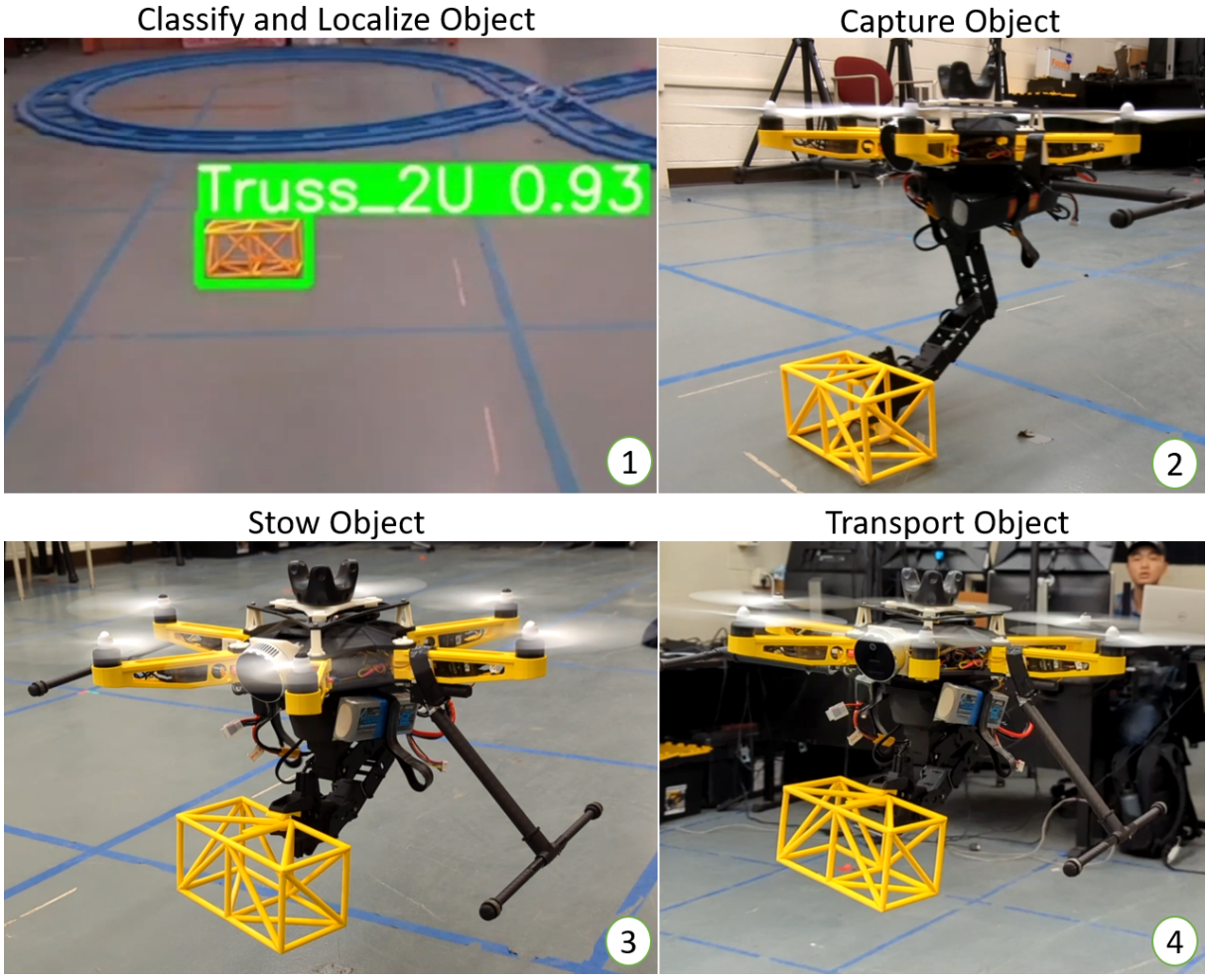

Figure 5.15: Chronological illustration of drone pick sequence, Video illustration

The object capture sequence is also described via Algorithm 1 below.

Algorithm 1 Sensor detection and Robotic Arm capture.

Require: Input for desired object

```
Conduct CNN object search
if CNN confidence > 0.60 then
    (x,y) object location = bounding box center
    (z) object location = LiDAR point cloud of bounding box center
end if

if drone location ≠ Object location then
    Maneuver drone to object of interest
end if

if drone location = Object location then
    initiate robotic arm IK for object capture
    if capture = True then
        Stow captured object for transport
    end if
end if

if capture = False then
    Conduct CNN object search
    re-initiate robotic arm IK for object capture
end if
```

After capture is complete the object can now be autonomously moved through space by any number of path planning optimization algorithms or other parameters set by the user.

Chapter 6

Discussion

This research has demonstrated the ability to use infinitely adjustable synthetic world creation tools to generate synthetic training data for a computer vision architecture. These trained CV models were then deployed onboard mobile manipulation drone platforms complete with sensor integration APIs for purposes of autonomous tasking and object capture using mostly COTS compute hardware. This free-flying drone capability was then successfully deployed in a uniquely capable synthetic environment projected within a real-world space using large-scale theater technologies, demonstrating a hardware-in-the-loop testing methodology to evaluate onboard computer vision and drone control for a wide range of environments, including dynamic orbital environments. Further domain randomization of these synthetic worlds was used to bridge the reality gap between synthetically trained computer vision models and real-world target domains that these systems will ultimately be deployed to. To my knowledge, this is the first study to integrate computer vision, domain randomization, environmental projection, and free-flying mobile manipulation platforms into a complete end-to-end solution. As of the time of this publication, the current computer vision simulation capabilities meet NASA Technology readiness level five specifications (TRL5), with the Virginia Tech Cube providing component validation within a relevant environment. HIL simulation and testing to include both the drone platform and robotic arm are TLR4, capable of verification in a laboratory environment. To achieve TLR4, a full 6DOF motion must be achieved before it can be considered a “relevant environment” These

readiness levels will only increase as the future work capabilities detailed in the following section are implemented. As one can imagine, this capability is not limited to space-based applications, but any and all projects in need of real-time onboard detection of large numbers of heterogeneous objects, or in need of a flying sensor platform for system testing or autonomous tasking.

6.1 Future Work

6.1.1 Orbital Dynamics Control Simulation

This project was not developed in a vacuum, but designed to be integrated into a larger space domain awareness simulation and testing suite. The system can achieve four DOF virtual simulations with a constant downward gravity vector, which effectively couples pitch and roll motion with translational movement. In an effort to simulate uncoupled 6DOF motion, a sister lab is developing an omnidirectional drone platform [23] [10] [11] capable of six DOF motion, better known as the "Onmi-Drone" illustrated in Figure 6.1. This 6 DOF platform combined with orbital motion simulation will provide a very capable HIL simulation platform. Additionally, we are endeavoring to simulate a frictionless motion simulator to an air-bearing table via drone PID controllers. Such capabilities will enable true six DOF HIL testing for spaceborne applications and testing. When paired with a facility such as the Virginia Tech CUBE.

Another implementation of this system is to deploy onboard relative motion dynamics via drone controller augmentation. The SpaceDrones platform has already proven that it can do decentralized multi-drone swarming [27] [40], Which opens up a unique opportunity to label multiple drones as deputy and chief relative motion objects and then alter onboard

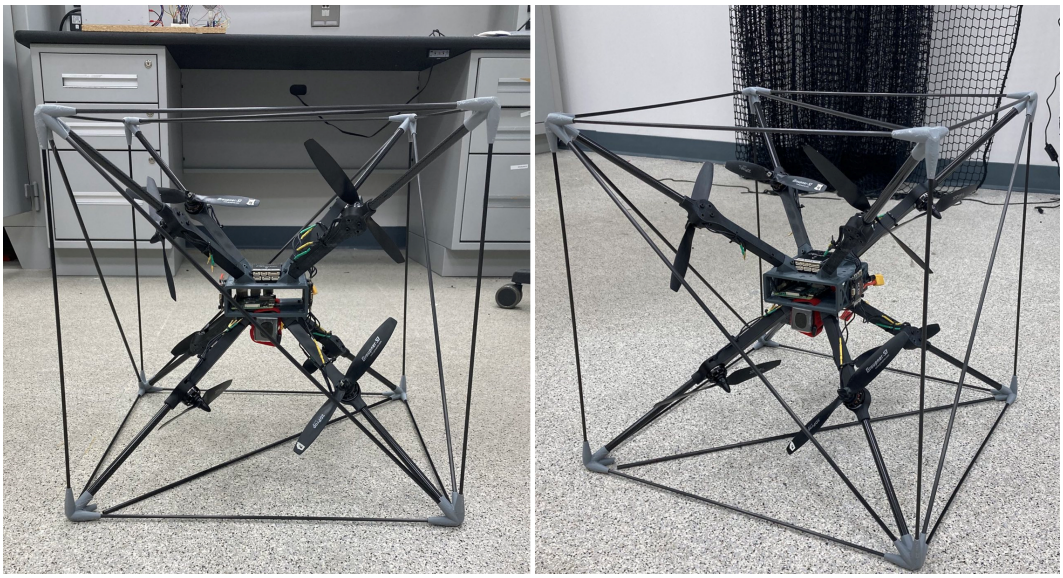

Figure 6.1: Virginia Tech Omni-directional drone (Omni Copter)

flight controllers to simulate orbital dynamics. This endeavor has already been started at Virginia Tech, achieving a orbital motion simulation using small (0.127m wide) free-flying drones depicted in Figure 6.2 by establishing a fixed coordinate system defined by an Opti-Track motion capture array, thus allowing simulation to leave the planer constraints of an air bearing table. Using a small drone with a large 685 square meter 3D motion capture space, this system achieved a maximum tracking error of 0.123 meters.

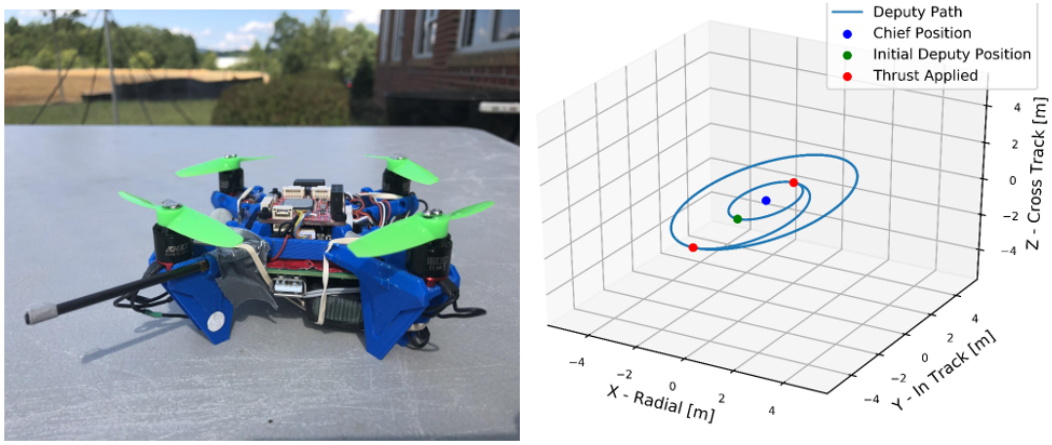

Figure 6.2: Relative motion simulation using micro drones

6.1.2 Deep Space Simulation Environments

Expanding the existing visual simulation capability to other deep space environments such as Mars to test autonomous habitat assembly, inspection, and design is currently in development. Synthetic images in Engine and projected in the Virginia Tech Cube are illustrated in Figure 6.3. The bottom picture illustrates a real-world drone being mirrored as a virtual reality drone within the Unreal Engine. Porting position data of real-world and virtual objects to and from synthetic environments allows researchers to test for relatively dangerous operations using real-world hardware without risking damage when motion is projected into a synthetic space. Video of this process can be found at SpaceDrones Mars VR Projection.

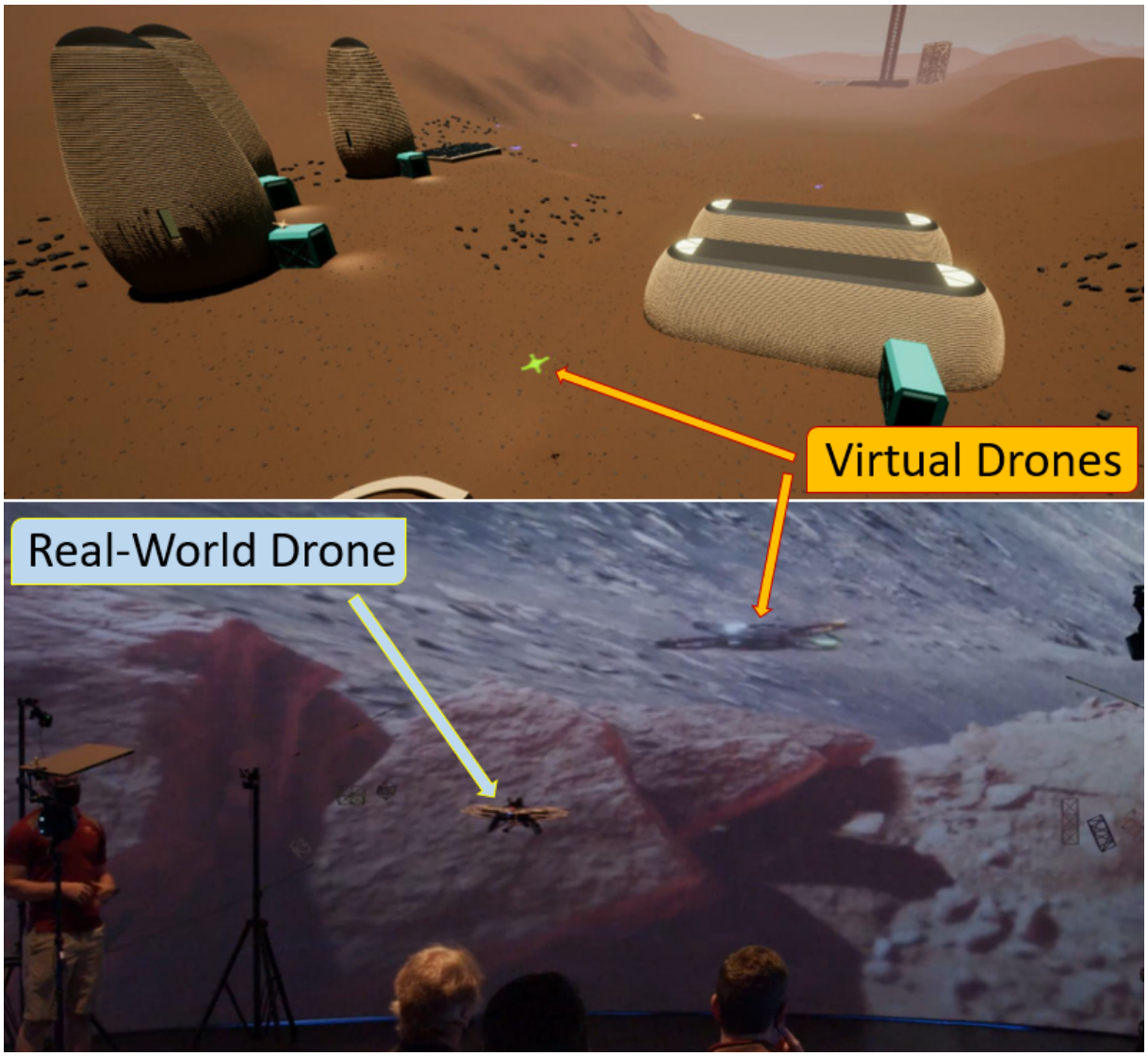

Figure 6.3: Simulated Mars environment with virtual reality drone bodies performing autonomous tasking [Top]. Mars environment projection in the Virginia Tech Cube with real-world drone motioned being mirrored to a virtual drone vehicle [Bottom]. Video illustration

6.1.3 Autonomous Path Planning

Path planning is an essential part of autonomous tasking. Organizations like the University of Washington's Autonomous Control Laboratory have achieved high fidelity drone control and path planning architecture [62] [61]. An Example of such a system is illustrated in Figure

6.4. Implementing these capabilities into the existing SpaceDrones architecture is currently in development and is a high priority for future operations.

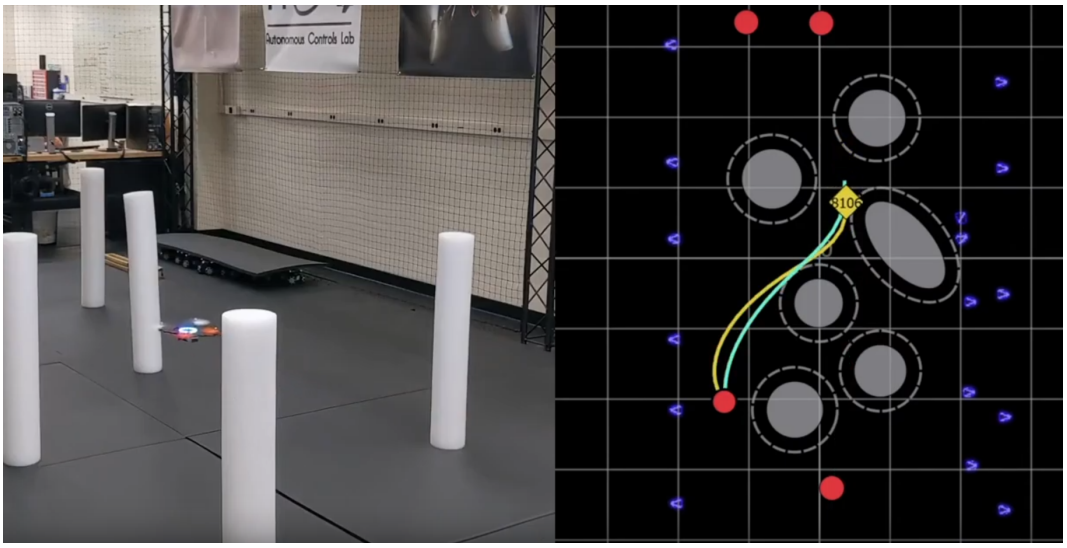

Figure 6.4: Vehicle Path Planning by University of Washington's Autonomous Control Laboratory [62], Video illustration

6.2 Domain Adaptation

Domain shift dramatically affects a deep network's performance due to feature extraction becoming more and more specific with each training epoch for any specified domain/environment, causing the network to become too specific, otherwise known as overfitting. Thus, the neural network fails to generalize objects within the domain from which it was trained and therefore, is ineffective for other domains. While domain randomization attempts to train one model for deployment over several domains and use cases, domain adaptation creates new models (source model) trained for a specific target domain using learned neuron weights from another model from a different domain as a starting point. This methodology [36] [58] is categorized under a CNN training methodology known as "transfer learning".

Several Domain Adaptation training implementations have been proposed, including re-training the model in the target domain, adapting the model's weights based on the statistics of the source and target domains, learning variation features between domains, and learning a mapping from the target domain to the source domain. Implementing this capability of cross-domain training between synthetic and real-world imagery is one of the next implementation steps for the continuation of the work described in this dissertation.

6.2.1 Defective Manufacturing Computer Vision capability

Throughout this project, I experimented with detecting malfunctions or failure states with objects of interest (illustrated in Figure 6.5). The ability to detect imperfections in manufacturing or failures in assembly before they happen would be an invaluable technology for future deep space or autonomous orbital platforms. During the process of fabricating truss pieces for our experiments, some trusses failed to print due to a wide range of issues. For example, the most prevalent failure was extruding support material sections perpendicular to the gravity vector, causing them to sag or splay out. Preliminary image data collection and neural network modeling were started to determine and identify failed truss sections. Initial results yielded a mean average precision of only 40%. However, I am confident with more image data on a broader range of failure states could increase model performance for this problem set.

This experiment was also expanded to a real-world small satellite in the process of being deployed by the Virginia Tech satellite team. This particular satellite has a spring-loaded boom that is designed to extend once reaching orbit. If this particular boom does not extend to its full length, it is classified as a failed state. Determining whether or not the boom deployment has failed, and other satellite related failures purely using computer vision

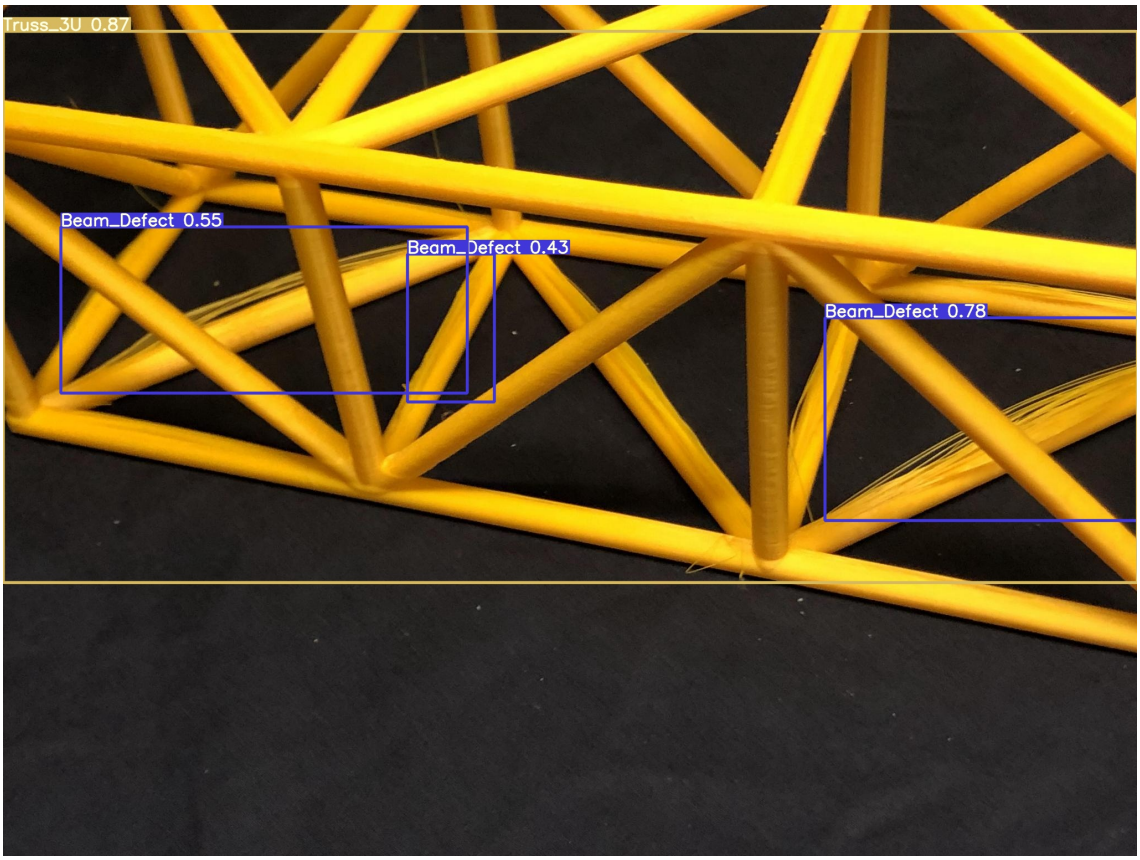

Figure 6.5: Initial Detection results additive Manufacturing imperfections using neural networking

(Synthetic or otherwise) is another experimental result that can be further expanded upon in future work.

6.2.2 Space Compute Hardware Needed

Once a computer vision model has been sufficiently trained for the desired task, deploying that model onboard a satellite, rover, or space station with the available hardware remains quite challenging.

Conventional avionics control architectures use one or more central onboard computers (OBC), acting as a central hub for all other subordinate sub-systems, including sensor data.

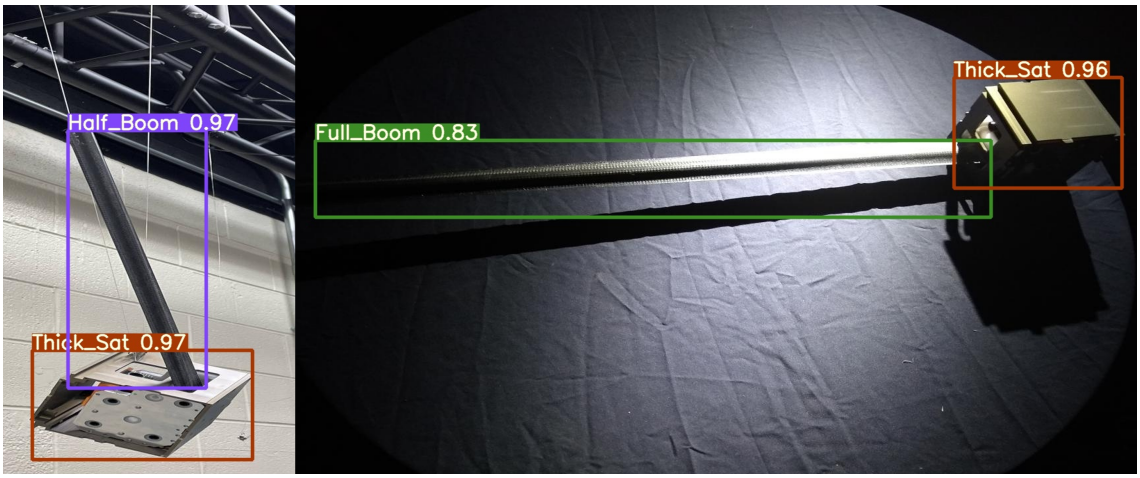

Figure 6.6: Full vs Half Boom extension using Computer Vision to determine deployment failures on real world space flight hardware

However, performing object inferencing in real time is computationally demanding, requiring a high-performance embedded computer capable of high bandwidth communication protocols.

Providing space-grade radiation hardening for any high-performance compute capability, for now, is expensive and time-consuming. The computational architecture will more than likely be modeled after a CUDA [16] capable Graphics Processing Unit (GPU) with several Gigabytes of onboard virtual memory (VRAM) that will need to be protected against extreme vibrations, hard vacuum, extreme thermal fluctuations, and radiation.

As of today, the latest radiation-tolerant 32-bit microprocessors, such as LEON4 [52] or RAD750 [9] are capable of just over 400 DMIPS (Dhrystone Million Instructions per Second). Compared to the 32 TeraOPS (trillion Operations Per Second) required to run the most modern computer vision architectures 30 frames per second (FPS), space-grade CPUs are orders of magnitude behind in needed computational complexity. However, the 32-bit floating-point arithmetic used by dedicated GPUs for rendering rasterization polygons operate at a level of mathematical precision not required for simple multiply and accumulate

matrix operations used for machine learning applications. Adopting dedicated analog or digital machine learning integrated circuits capable of high volumes of low precision calculations such as 8-bit integer computational architectures will allow for significantly more training calculations at a lower energy cost. However, this work is just now being pioneered.

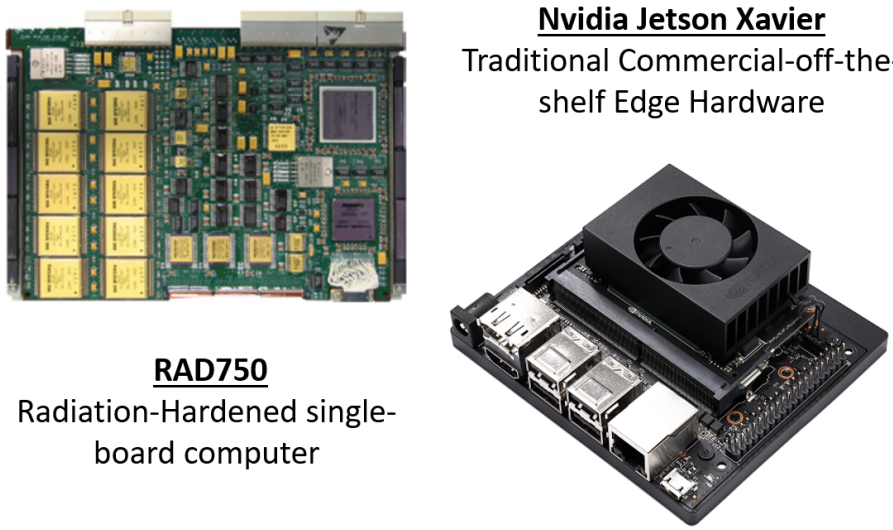

Figure 6.7: Space grade radiation hardened computational hardware vs. traditional commercial-off-the-shelf computational hardware

6.2.3 Effects of Ionizing Radiation on Neural Networks

The space industry's adoption of machine learning/artificial intelligence technologies poses its own set of unique challenges. A robust machine learning model capable of performing autonomous tasks may work when evaluated inside the protective magnetosphere of planet earth. However, the effects of ionizing radiation on a trained machine learning model or neural network architecture to date have not been done. In the event of a single or successive single event upsets (SEUs), commonly referred to as bit flips, how many neurons or neuron connections can be lost before performance degrades to an unacceptable level? What neural network layers are most susceptible [8], can redundant layers be retroactively activated in

flight? Can a new model be uploaded to counteract dead neurons, or can the compute hardware responsible for ML computation be sufficiently shielded to mitigate or prevent such effects?

All of these are questions that will need to be answered before deploying machine learning algorithms beyond Earth's atmosphere.

Chapter 7

Conclusions

This research intersected many disciplines requiring a working knowledge of several fields including software engineering, robotics, computer vision, machine learning, fabrication, animation, theater projection, and visual design. All culminating in a successful deployment of onboard automated tasking via computer vision and sensor localization for the purposes of automated robotic tasking and object capture/manipulation for in-space assembly or on orbit servicing implementations. This dissertation has demonstrated the applicability of Convolutional Neural Networks, synthetic data generation, and domain randomization to detect various orbital objects of interest for In-space assembly (ISA) on-orbit servicing (OOS) operations in simulated environments, achieving a mean average precision within 11% of none synthetically trained models for a rather difficult array of geometrically similar objects. That simulated environment was then pulled out of a relatively small computer screen and projected into a large 21,000 cubic meters space for large scale visual simulation, enabling researches to visually interact with optical sensors as well as physically interact with real world mocks ups with in a visual simulation environment with real world hardware. This solution has opened the door for hardware-in-the-loop computer vision testing, deployed on real world free flying hardware with environments that are difficult to capture or recreate, or safely operate hardware in a controlled testing regime. Figure 7.1 compares the capabilities of the current SpaceDrones Architectures with recent literature, detailing the combination of many areas of study to solve a critical need with in the space industry. However, the problem

of hardware-in-the-loop testing for any number of parameters or environmental factors is not on specific to the aerospace community. This simulation solution can and should be applied to any number of research, search and rescue, defence, and general automation applications. A short video detailing this projects methodologies in chronological order can be found at SpaceDrones 2.0 Video Summery.

Research Comparison

	Tron Lab	DLR RPO	DLR ATON	USN Pose Estimation	Airbus SurRender	Domain Randomization (Tobin)	Domain Randomization (Nvidia)	**SpaceDrones (Ours)**
4 DOF Motion			✓					✓
5 DOF Motion								
6 DOF Motion	✓	✓						
Free Flight Motion			✓					✓
Onboard Computer Vision	✓	✓	✓	✓		✓	✓	✓
Synthetic Data Training			✓	✓	✓	✓	✓	✓
Domain Randmization				✓		✓	✓	✓
Object Localization	✓	✓	✓		✓		✓	✓
Robotic Arm Manipulation		✓						✓
Hardware-in-the-Loop	✓	✓	✓					✓

Figure 7.1: Research Comparison with current literature. Tron Lab [25], DLR RPO [21], DLR ATON [56], United States Navy (USN) Pose Estimation, [17], Airbuse SurRender [35], Jet Propulsion Laboratory (JPL) Formation Control Testbed (FCT) [49], Domain Randomization (Tobin) [54], Domain Randomization (Nvidia)[55]

Bibliography

[1] Ros with mavros installation guide. URL https://docs.px4.io/master/en/ros/mavros_installation.html.

[2] Jetson agx xavier and the new era of autonomous. URL https://info.nvidia.com/rs/156-OFN-742/images/Jetson_AGX_Xavier_New_Era_Autonomous_Machines.pdf.

[3] Pincherx 100 robot arm. URL https://www.trossenrobotics.com/pincherx-100-robot-arm.aspx.

[4] About steamvr base station 2.0. URL https://www.vive.com/us/support/vive-pro/category_howto/about-the-base-stations.html.

[5] Vive tracker: Vive united states. URL https://www.vive.com/us/accessory/vive-tracker/.

[6] Spiderfab. 2014. URL https://www.nasa.gov/content/spiderfab.

[7] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[8] Austin P Arechiga, Alan J Michaels, and Jonathan T Black. Onboard image processing for small satellites. In *NAECON 2018-IEEE National Aerospace and Electronics Conference*, pages 234–240. IEEE, 2018.

[9] Richard W Berger, Devin Bayles, Ronald Brown, Scott Doyle, Abbas Kazemzadeh, Ken Knowles, David Moser, John Rodgers, Brian Saari, Dan Stanley, et al. The rad750/sup tm/-a radiation hardened powerpc/sup tm/processor for high performance spaceborne applications. In *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, volume 5, pages 2263–2272. IEEE, 2001.

[10] Theresa Blandino, Alexander Leonessa, Daniel Doyle, and Jonathan Black. Position control of an omni-directional aerial vehicle for simulating free-flyer in-space assembly operations. In *ASCEND 2021*, page 4100. 2021.

[11] Theresa Blandino, Alexander Leonessa, Daniel Doyle, and Jonathan Black. Position control of an omni-directional aerial vehicle for simulating free-flyer in-space assembly operations. In *ASCEND 2021*, page 4100. 2021.

[12] DART Mishap Investigation Board. Overview of the dart mishap investigation results. Technical report, Tech. rep., NASA, 2006. Available at dart mishap investigation, 2006.

[13] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[14] BG Boone, JR Bruzzi, WF Dellinger, BE Kluga, and KM Strobehn. Optical simulator and testbed for spacecraft star tracker development. In *Optical Modeling and Performance Predictions II*, volume 5867, page 586711. International Society for Optics and Photonics, 2005.

[15] Miguel Borges, Andrew Symington, Brian Coltin, Trey Smith, and Rodrigo Ventura. Htc vive: Analysis and accuracy improvement. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2610–2615. IEEE, 2018.

[16] Ian Buck. Gpu computing with nvidia cuda. In *ACM SIGGRAPH 2007 courses*, pages 6–es. 2007.

[17] Tae W. Lim Charles Oestreich. On-orbit relative pose initialization via convolutional neural networks. *AIAA*, 2020.

[18] Brandon J Christensen, Gustavo Gargioni, Daniel Doyle, Kevin Schroeder, and Jonathan Black. Space simulation overview: Leading developments towards using multi-

rotors to simulate space vehicle dynamics. In *AIAA Scitech 2020 Forum*, page 1134, 2020.

[19] Blender Online Community. *Blender 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[20] Simone D'Amico, J-S Ardaens, and Robin Larsson. Spaceborne autonomous formation-flying experiment on the prisma mission. *Journal of Guidance, Control, and Dynamics*, 35(3):834–850, 2012.

[21] Armin Hauke Daniel Weber, Marcin Gnat and Felix Huber. End-to-end simulation of on-orbit-servicing: Technical implementation of communications. *AIAA Journal*, 2018.

[22] Inc. DBA. Optitrack url = https://optitrack.com/,.

[23] Collin Deans, Theresa Furgiuele, Daniel Doyle, and Jonathan Black. Simulating omni-directional aerial vehicle operations for modeling satellite dynamics. In *ASCEND 2020*, page 4023. 2020.

[24] Minzhen Du, Gustavo Gargioni, Daniel Doyle, and Jonathan Black. Assessment of tracking small uas using ir based laser and monocular-vision pose estimation. In *2020 IEEE Aerospace Conference*, pages 1–10. IEEE, 2020.

[25] Simone D'Amico. Space rendezvous laboratory. 2020. URL https://damicos.people.stanford.edu/.

[26] Epic Games. Unreal engine. *Online: https://www.unrealengine.com*, 3, 2007.

[27] Gustavo Gargioni, Marco Peterson, JB Persons, Kevin Schroeder, and Jonathan Black. A full distributed multipurpose autonomous flight system using 3d position tracking

and ros. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1458–1466. IEEE, 2019.

[28] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[29] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[30] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[32] Juniper C Jairala, Robert Durkin, Ralph J Marak, Stephanie A Sipila, Zane A Ney, Scott E Parazynski, and Arthur H Thomason. Eva development and verification testing at nasa's neutral buoyancy laboratory. *JSC/EC5 US Spacesuit Knowledge Capture (KC) Series Synopsis*, 2012.

[33] Ksenia Klionovska and Matthias Burri. Hardware-in-the-loop simulations with umbra conditions for spacecraft rendezvous with PMD visual sensors. *Sensors*, 21(4):1–17, feb 2021. ISSN 14248220. doi: 10.3390/s21041455.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

[35] Jérémy Lebreton, Roland Brochard, Matthieu Baudry, Grégory Jonniaux, Adrien Hadj Salah, Keyvan Kanani, Matthieu Le Goff, Aurore Masson, Nicolas Ollagnier, Paolo

Panicucci, et al. Image simulation for space applications with the surrender software. *arXiv preprint arXiv:2106.11322*, 2021.

[36] Da Li, Jianshu Zhang, Yongxin Yang, Cong Liu, Yi-Zhe Song, and Timothy M Hospedales. Episodic training for domain generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1446–1455, 2019.

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[39] Eric Lyon, Terence Caulkins, Denis Blount, Ivica Ico Bukvic, Charles Nichols, Michael Roan, and Tanner Upthegrove. Genesis of the cube: The design and deployment of an hdla-based performance and research facility. *Computer Music Journal*, 40(4):62–78, 2016.

[40] Hannah Dornath Mohr. *Target Locating in Unknown Environments Using Distributed Autonomous Coordination of Aerial Vehicles*. PhD thesis, Virginia Tech, 2019.

[41] Jeremy Patrick Ogorzalek. *Computer Vision Tracking of sUAS from a Pan/Tilt Platform*. PhD thesis, Virginia Tech, 2019.

[42] Michele Ostovar. Space environments complex, Mar 2021. URL https://www.nasa.gov/offices/setmo/facilities/space_environments_complex.

[43] Marco Peterson. Spacedrones labeled training images and results. 2022. URL https://doi.org/10.7294/19241844.

[44] Marco Peterson and Minzhen Du. Spacedrones unreal engine projects. 2022. URL https://doi.org/10.7294/19386125.

[45] Marco Peterson, Minzhen Du, Nadhir Cherfaoui, Alby Koolipurackal, Daniel D Doyle, and Jonathan T Black. Comprehensive assessment of neural network synthetic training methods using domain randomization for orbital and space-based applications. In *2022 IEEE/SICE International Symposium on System Integration (SII)*, pages 329–335. IEEE, 2022.

[46] Tommaso Guffanti Matthew Willis Vince Giralo Michelle Chernick Sebastian Grimberg Adam Koenig Sumant Sharma Connor Beierle Lukas Steindorf Prof. Simone D'Amico, Duncan Eddy and Ibrahim Makhadmi. Tron – the testbed for rendezvous and optical navigation – enters the grid. 2020.

[47] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[48] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[49] Daniel P. Scharf, Fred Y. Hadaegh, Jason A. Keim, Edward G. Benowitz, and Peter R. Lawson. Flight-like ground demonstration of precision formation flying spacecraft. volume 6693, page 669307. SPIE, 9 2007. doi: 10.1117/12.735125.

[50] Brad Sease, Brien Flewelling, and Jonathan Black. Automatic streak endpoint localization from the cornerness metric. *Acta astronautica*, 134:345–354, 2017.

[51] Sumant Sharma and Simone D'Amico. Pose estimation for non-cooperative rendezvous using neural networks, 2019.

[52] Magnus Själander, Sandi Habinc, and Jiri Gaisler. Leon4: Fourth generation of the leon processor. *Proceedings of Data Systems in Aerospace*.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[54] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[55] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[56] Guilherme F Trigo, Bolko Maass, Hans Krüger, and Stephan Theil. Hybrid optical navigation by crater detection for lunar pin-point landing: trajectories from helicopter flight tests. *CEAS Space Journal*, 10(4):567–581, 2018.

[57] Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring, 2019.

[58] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey, 2018.

[59] Pedro F. Proenca Yang Gao. Deep learning for spacecraft pose estimation from photo-realistic rendering. *Cornell Universirty*, 2019.

[60] Chengxi Ye, Matthew Evanusa, Hua He, Anton Mitrokhin, Tom Goldstein, James A. Yorke, Cornelia Fermüller, and Yiannis Aloimonos. Network deconvolution, 2020.

[61] Yue Yu and Ufuk Topcu. Proportional-integral projected gradient method for infeasibility detection in conic optimization. *arXiv preprint arXiv:2109.02756*, 2021.

[62] Yue Yu, Purnanand Elango, Ufuk Topcu, and Behçet Açıkmeşe. Proportional-integral projected gradient method for conic optimization. *https://doi.org/10.48550/arxiv.2109.02756*, 2021.

[63] Xiangyu Yue, Yang Zhang, Sicheng Zhao, Alberto Sangiovanni-Vincentelli, Kurt Keutzer, and Boqing Gong. Domain randomization and pyramid consistency: Simulation-to-real generalization without accessing target domain data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2100–2110, 2019.

Appendices

Appendix A

Neural Network Mathematical Diagram

A.1 Weighted Sum Cost

Single neuron per layer diagram of the contributing mathematical functions to each weight in the layer.

A.2 Summed Cost

Cost of all Neurons in a Network summed over L Layers

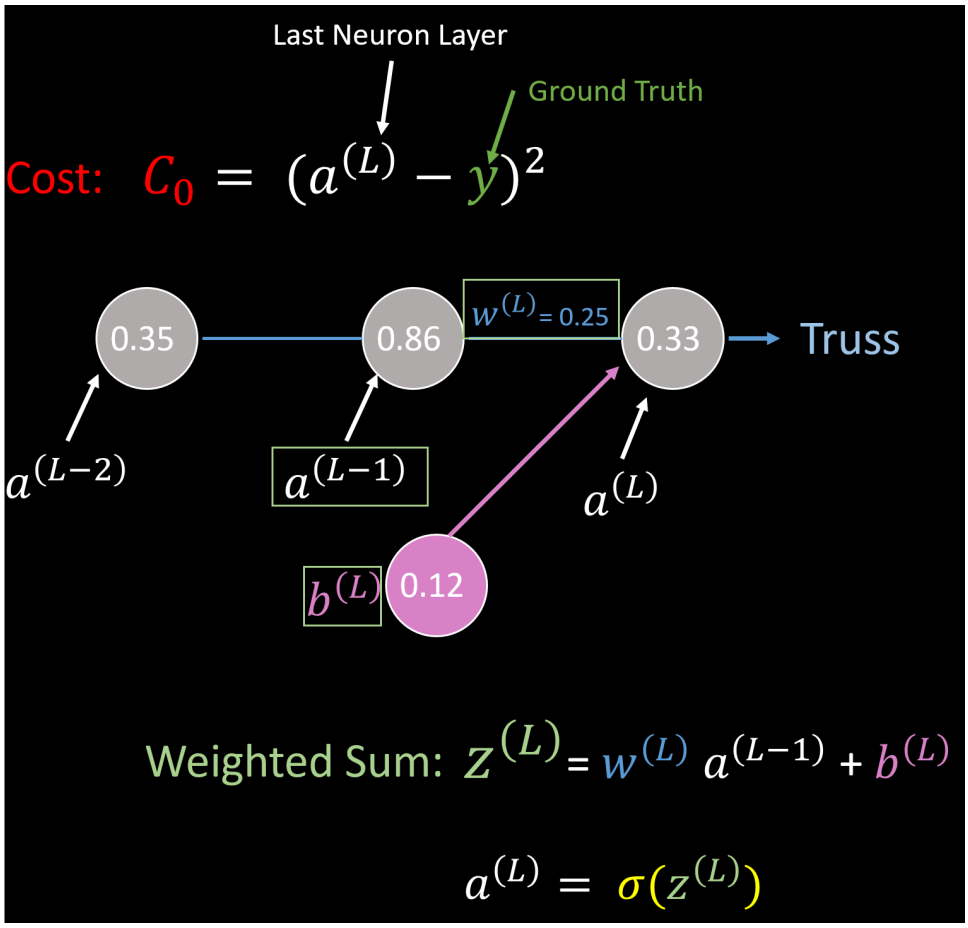

Figure A.1: Weighted Sum mathematical Diagram

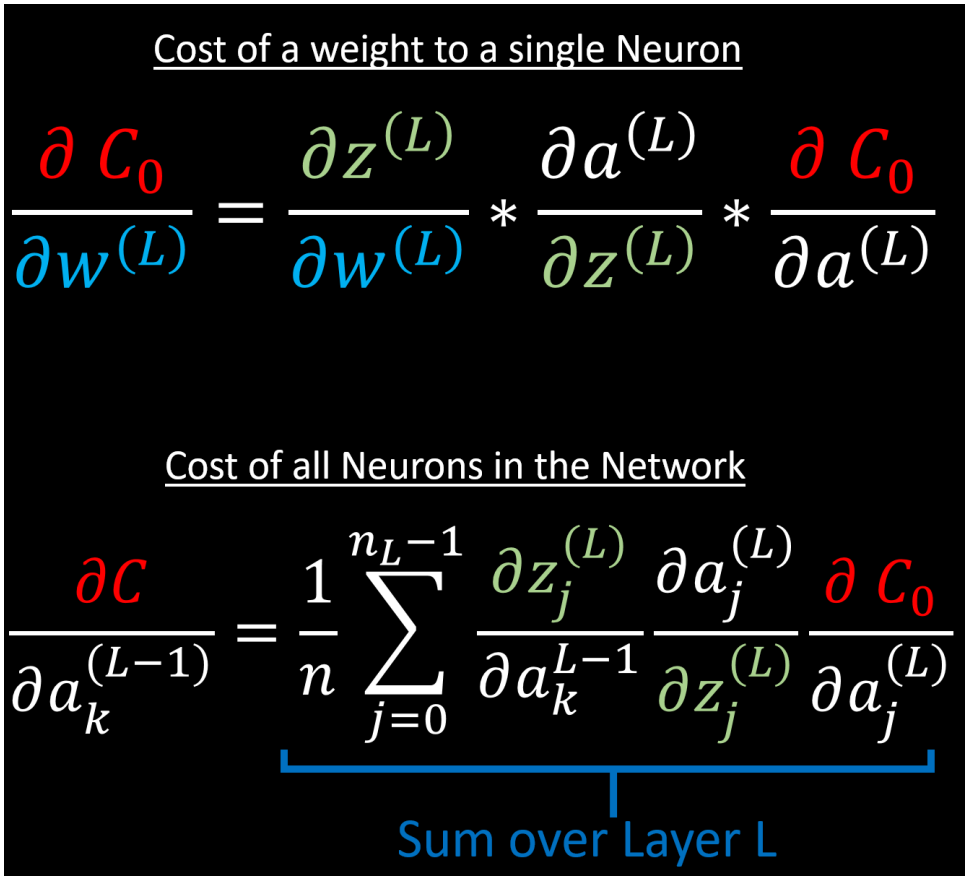

Figure A.2: Summation Cost over L Layers

Appendix B

System UML Diagram

Software dependencies and data interlinks for the neural network inferancing, object localization, and capture is described in [B.1].

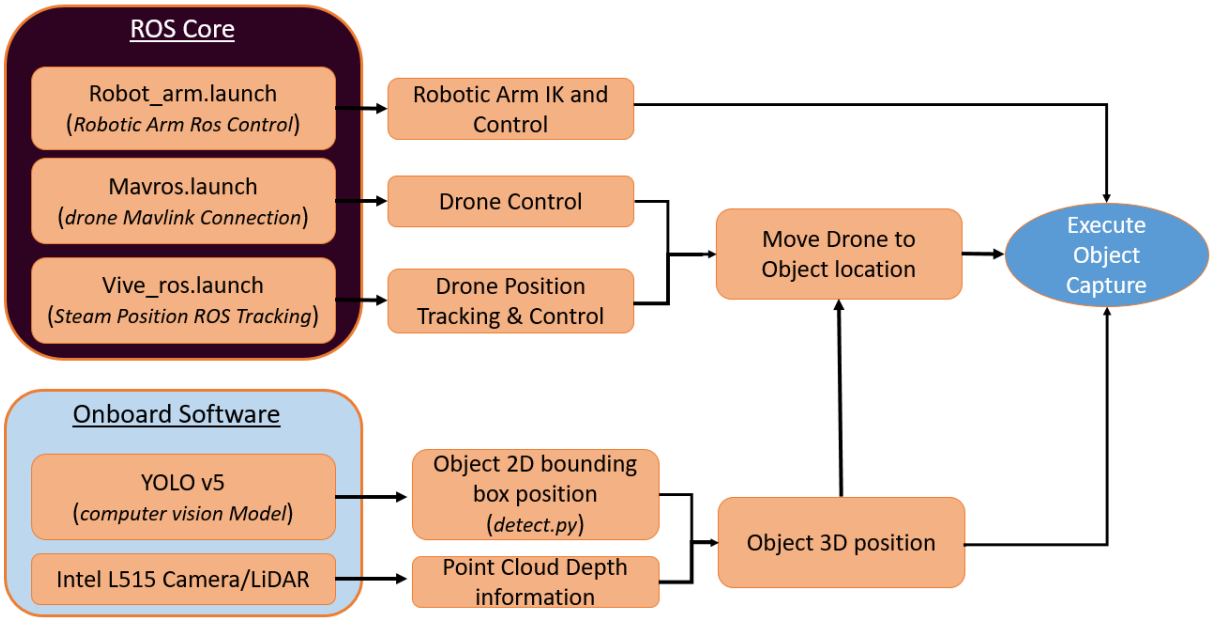

Figure B.1: Drone Localization and Capture Software Diagram