2018

Integrated Sensing Systems for On-Orbit Robotic Satellite Servicing Applications

Eric Loy

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Loy, Eric, "Integrated Sensing Systems for On-Orbit Robotic Satellite Servicing Applications" (2018). *Graduate Theses, Dissertations, and Problem Reports*. 6115.
<https://researchrepository.wvu.edu/etd/6115>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

INTEGRATED SENSING SYSTEMS FOR ON-ORBIT ROBOTIC SATELLITE SERVICING APPLICATIONS

Eric Loy

**Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University**

in partial fulfillment of the requirements for the degree of

**Master of Science in
Electrical Engineering**

**Powsiri Klinkhachorn, Ph.D., Chair
Thomas Evans, Ph.D., Co-Chair
Roy S. Nutter, Ph.D.
Chad C. Panther, Ph.D.**

Lane Department of Computer Science and Electrical Engineering

**Morgantown, West Virginia
2017**

**Keywords: Satellite Servicing, Marman Ring, Strain Gauge,
Load Cell, Time of Flight, Range Sensors, GUI**

Copyright 2017 Eric Loy

ABSTRACT

INTEGRATED SENSING SYSTEMS FOR ON-ORBIT ROBOTIC SATELLITE SERVICING APPLICATIONS

Eric Loy

There are a variety of stakeholders when it comes to space (the military, other governmental entities, private sector companies, consumers, the scientific community, etc.), and they would all prefer the operational lives of their satellites, observatories, space stations, and other assets to be maximized. Out of the large number of assets in space, only a few were designed with serviceability as a goal. The ability to service “non-cooperative” satellites is an ongoing project being investigated by the National Aeronautics and Space Administration’s (NASA) Goddard Space Flight Center (GSFC) and Satellite Servicing Projects Division (SSPD), along with other industrial partners. This mission is known as Restore-L, which is also the name of the servicing vehicle being developed to capture, refuel, and relocate a government satellite. One critical aspect of this mission is capturing and securely docking the satellite to the servicer to allow robotic manipulators to refuel it. The Client Berthing System (CBS) is a custom grappling tool on the servicer that enables docking with a Marman ring present on most satellites. A wide variety of sensory systems can be employed on the CBS to increase the likelihood of a successful satellite capture and to reduce the chance of catastrophic failure.

This research explores two systems derived from widely available commercial-off-the-shelf (COTS) range and force sensors to allow the CBS to detect the pose of an inbound satellite and when contact is established. Monitoring pose enables nominal orientation of a satellite with the servicing vehicle, and sensing force allows the CBS to provide state information during the docking procedure. Testing revealed that the range sensors were able to track distances to a Marman ring with average errors of 3-8.5%. Testing of many CBS to Marman ring contact configurations revealed that the force sensors were within a 5% average error of the applied load. Additionally, they were able to determine x- and y-torque on the CBS within a 20% average error. Only one particular contact configuration resulted in average force and torque errors of 15% and 30%, respectively. This work provided the proof-of-concept for two systems intended to enhance the CBS. Future work involving these systems will include flight qualifications for an actual space mission.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Klink, for all the input and guidance over the course of this research. His wisdom, unwavering work ethic, and availability inside the classroom, outside the classroom, weekdays, weekends, day, and night were instrumental to the completion of my degree. Without him, I would not have been able to finish. I would also like to express a huge thanks to my co-advisor, Dr. Evans. Thank you for this great opportunity that allowed me to work with WVRTC staff and meet some prominent NASA figures. Thank you for allowing me access to WVRTC equipment and facilities for testing. Many thanks go out to Dr. Panther as well for providing invaluable input and direction throughout this project. Thank you for providing specific test plan procedures and CAD support when needed. I would also like to thank Dr. Nutter for keeping me motivated every time I saw him. Completion of this research would not have been possible without calling upon knowledge gained while I was his student, as well as Dr. Klink's. To my examining committee in general, thank you all very much for being flexible as I scheduled my defense. Thank you all for providing valuable input regarding my thesis.

In addition, I would like to thank WVRTC staff both past and present for assisting me. Thank you, Jason Battin, for getting me started with this research and providing the necessary background information from years past. Thank you, Michael Campbell and Patrick Lewis, for robot operations during testing and data analytic strategies. Thank you, Dr. Kevyn McBride, for providing robot log information that was relevant to data analysis. Without these WVRTC staff members, I could not have tested and evaluated my prototype. I would also like to thank graduate research assistant, Kavin Sivaneri, for helping me with the Shimadzu for load cell calibrations. Without him I would not have had accurate load cell software models.

Finally, endless love and thanks goes out to my family: Mom, Dad, and Sister Emily. We have all been through a lot in the past few years, and without their support, I would not have continued. I would not be where I am today without them. Their encouragement and reassurance throughout my graduate education was, at times, the only thing that kept me moving forward. Thank you all so much.

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGEMENTS	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	viii
LIST OF TABLES	xiii
LIST OF ABBREVIATIONS	xiv
LIST OF SYMBOLS	xv
CHAPTER 1 INTRODUCTION	1
1.1 Brief Overview	1
1.2 Purpose	7
1.3 Scope	7
1.4 Research Goals	8
1.5 Approach	9
1.6 Tools & Technologies Used	12
1.7 Thesis Organization.....	13
CHAPTER 2 LITERATURE REVIEW.....	14
2.1 Previous Work.....	14
2.1.1 WVRTC Research from 2011.....	14
2.1.2 WVRTC Research from 2012.....	15
2.1.3 WVRTC Research from 2012-2015	16
2.2 Hardware Background – VL6180X & Nucleo.....	17
2.2.1 VL6180X Sensor	17
2.2.2 Mbed and the STM32 Nucleo-64 Development Board with STM32L053R8 MCU ..	21
2.2.3 COTS VL6180X Hardware on a Nucleo L053R8.....	23
2.3 Hardware Background – Load Cells & Phidgets	26
2.3.1 Load Cell Fundamentals	26
2.3.2 Load Cells for this Research	29
2.3.3 Sampling Hardware	30

CHAPTER 3 CBSS DESIGN AND CONCEPTS	32
3.1 Design of the FSS	32
3.1.1 Phidgets Wiring Diagram	33
3.1.2 Load Cell Calibration on Shimadzu.....	34
3.1.3 RSS Design	38
3.1.4 The FSS and Fundamental Concepts	41
3.2 Design of the PSS.....	45
3.2.1 Initial PSS Hardware Setup	45
3.2.2 Nucleo Code for PSS	46
3.2.3 VL6180X Placement in Palm Tools and on CBSS.....	49
3.2.4 Sensor Issues, Fix, and Schematic	51
3.2.5 PSS Fundamental Concepts	54
3.3 CBSS Hardware Setup	56
CHAPTER 4 CBSS SOFTWARE TOOLS AND EXPERIMENTAL OVERVIEW	57
4.1 CBSS Software Overview	57
4.2 CBSS GUI.....	58
4.2.1 Dependencies and Assumptions	58
4.2.2 Operating the GUI	59
4.2.3 Software Flowcharts	64
4.3 CBSS C++ Data Logger	70
4.3.1 Dependencies and Assumptions	70
4.3.2 Compilation and Execution	71
4.3.3 Software Flowchart.....	71
4.4 Testing Overview	73
4.4.1 WVRTC Lab Equipment Used	73
4.4.2 General Test Setup.....	77
4.4.3 Data Management	78
4.4.4 Independent Testing Variables	78
4.5 Nominal Contact FSS Testing.....	78
4.6 Misalignment to Nominal Contact FSS Testing.....	80
4.7 Nominal Alignment PSS Testing	81

4.7.1 Leica Tracking TMAC.....	81
4.7.2 Moto 50 Robot Kinematics	82
4.8 FSS Test Plan	83
4.8.1 Nominal Contact	83
4.8.2 Misalignment One – Two Palm, Two Rail Contact.....	84
4.8.3 Misalignment Two – Single Palm, Two Rail Contact	86
4.8.4 Misalignment Three – Single Palm, Single Rail Contact	88
4.9 Performance Evaluation	89
CHAPTER 5 RESULTS.....	91
5.1 Post Processing and Data Conditioning	91
5.1.1 CBSS Data – Force Conditioning.....	92
5.1.2 CBSS Data – Proximity Conditioning	95
5.1.3 Validation Data – JR3 Conditioning.....	96
5.1.4 Validation Data – Moto 50 Cartesian Position Conditioning	97
5.1.5 Data Alignment.....	98
5.2 Nominal Contact FSS Testing.....	99
5.3 Misalignment to Nominal Contact FSS Testing.....	111
5.4 Nominal Alignment PSS Testing	121
5.4.1 Leica Tracking T-Mac	121
5.4.2 Moto 50 Robot Kinematics	124
5.5 FSS Test Plan, Round One Testing	128
5.5.1 Nominal Contact.....	129
5.5.2 Misalignment One – Two Palm, Two Rail Contact	136
5.5.3 Misalignment Two – Single Palm, Two Rail Contact.....	142
5.5.4 Misalignment Three – Single Palm, Single Rail Contact.....	148
5.6 Load Cell Drift from Zero Force	154
CHAPTER 6 CONCLUSION.....	155
6.1 Summary	155
6.2 Future Work	156
REFERENCES.....	159
APPENDIX A CBSS Nucleo Code	165

APPENDIX B C++ CBSS Data Logger	172
APPENDIX C REMAINING NOMINAL CONTACT FSS TESTS.....	182
C.1 Test #2 Plots	182
C.2 Test #3 Plots	185
APPENDIX D REMAINING AUGUST 2016 TESTS.....	188
D.1 Test #2 Plots	188
D.2 Test #3 Plots	191
D.3 Test #4 Plots	194
D.4 Test #5 Plots	197
D.5 Test #6 Plots	200
D.6 Test #7 Plots	203
D.7 Test #8 Plots	206
D.8 Test #9 Plots	209
D.9 Test #10 Plots	212
D.10 Test #11 Plots	215

LIST OF FIGURES

Figure 1.1: Restore-L Objectives [3]	2
Figure 1.2: Docking a Satellite to the CBS [1]	3
Figure 1.3: Marman Ring and Clamp [6].....	4
Figure 1.4: Marman Ring in Palm Tool Jaw.....	5
Figure 1.5: WVRTC CBSS Mechanical Platform	6
Figure 1.6: Retrofitted Sensor Systems	9
Figure 2.1: Proposed 2011 System	15
Figure 2.2: 2012 Gripper and Sensor Systems.....	15
Figure 2.3: Research from 2012 Onward.....	16
Figure 2.4: 2014/15 Work Expanded.....	17
Figure 2.5: Time of Flight Concept [12].....	18
Figure 2.6: Conical Properties of VL6180X On-Board Sensors [13].....	19
Figure 2.7: Sensor Architecture [13]	20
Figure 2.8: I2C Addressing [15]	20
Figure 2.9: Nucleo L053R8 [17].....	21
Figure 2.10: Nucleo Pinout 1 [19]	22
Figure 2.11: Typical Sensor Setup [13]	23
Figure 2.12: Satellite Board Layout [20]	23
Figure 2.13: Satellite Board Schematic [20].....	24
Figure 2.14: Nucleo VL6180X Expansion Board [21]	24
Figure 2.15: VL6180X Expansion Board Satellite Connectors [21]	25
Figure 2.16: GPIO Expander on Nucleo VL6180X Expansion Board [21]	25
Figure 2.17: Bonded Foil Strain Gauge object [25].....	26
Figure 2.18: Quarter Bridge Strain Gauge Circuit [25]	27
Figure 2.19: Half Bridge Circuit (left) and Physical Example (right) [25]	28
Figure 2.20: The Full Bridge Strain Gauge Circuit [25].....	28
Figure 2.21: LCL-40 Load Cells and Mounting Kits	29
Figure 2.22: Load Cell Sampling Hardware [27]	30
Figure 3.1: 2015/16 Replica of CBS Palm Tool.....	32
Figure 3.2: Phidgets Wiring Diagram.....	33
Figure 3.3: Calibrating on Shimadzu	34
Figure 3.4: PhidgetBridge Monitoring Program.....	35
Figure 3.5: 3D Printer [31].....	38
Figure 3.6: RSS Exploded View	39
Figure 3.7: Internals of RSS Assembly.....	39
Figure 3.8: RSS Placement on CBS Palm Tool.....	40
Figure 3.9: RSS Height above Palm Surface	40

Figure 3.10: FSS Naming Convention on the CBSS	41
Figure 3.11: CBSS Forces.....	42
Figure 3.12: Length to Palm Tool.....	43
Figure 3.13: Palm Tool Cavities	45
Figure 3.14: Foundation of PSS Hardware	46
Figure 3.15: PSS Nucleo Code	47
Figure 3.16: PSS Snap-In Fixture and Components	49
Figure 3.17: VL6180X Placement in Palm Tool and Recess	49
Figure 3.18: PSS Naming Convention on the CBSS	50
Figure 3.19: I2C Clock and Data Lines without Pull-up Resistors.....	51
Figure 3.20: I2C Clock and Data Lines with Pull-up Resistors.....	52
Figure 3.21: Final PSS Wiring Diagram.....	53
Figure 3.22: Calculating Y-Axis Pose Correction	54
Figure 3.23: Calculating X-Axis Pose Correction	55
Figure 3.24: Complete CBSS Hardware Setup.....	56
Figure 4.1: CBSS Software GUI.....	60
Figure 4.2: RSS Rails in GUI and Sensor Readouts	61
Figure 4.3: CBSS Dimensions in GUI.....	61
Figure 4.4: Close Up of GUI Force Graphic.....	62
Figure 4.5: Proximity Plotting in the GUI	63
Figure 4.6: PhidgetBridge Properties in the GUI.....	63
Figure 4.7: GUI Logging	64
Figure 4.8: Overview of GUI Code	65
Figure 4.9: Load Cell Data Event Handler	67
Figure 4.10: Nucleo Serial Port Data Event Handler.....	68
Figure 4.11: Logging Event	69
Figure 4.12: C++ Logging Code Overview	72
Figure 4.13: Motoman-SIA50D [51]	73
Figure 4.14: Motoman-MH250 II [53]	74
Figure 4.15: Motoman-MH250 II and FSAB [54].....	74
Figure 4.16: JR3 Force Torque Sensor [55].....	75
Figure 4.17: Leica Tracking a T-MAC Sensor [59]	76
Figure 4.18: Equipment Locations.....	77
Figure 4.19: Nominal Contact (3 Palm Tools, 6 RSS, $\alpha=0^\circ$)	79
Figure 4.20: Misalignment (1 Palm Tool, 2 RSS, $\alpha=5^\circ$) to Nominal Contact (3 Palm Tools, 6 RSS, $\alpha=0^\circ$).....	80
Figure 4.21: Misalignment One (2 Palm Tools, 2 RSS, $\alpha=5^\circ$).....	84
Figure 4.22: Misalignment Two (1 Palm Tool, 2 RSS, $\alpha=10^\circ$)	86
Figure 4.23: Misalignment Three (1 Palm Tool, 1 RSS, $\alpha=5^\circ$)	88

Figure 5.1: Data Preparation for Analysis	91
Figure 5.2: Load Cell Calibration Lines	92
Figure 5.3: Interpolation Techniques	94
Figure 5.4: Coordinate Rotation [64].....	96
Figure 5.5: Finding Signal Delay.....	98
Figure 5.6: FSS Fundamental Concepts (June 2016, Test #1).....	99
Figure 5.7: FSS to JR3 Force-Torque Comparison (June 2016, Test #1).....	100
Figure 5.8: FSS Percent Errors without Data Deletion (June 2016, Test #1).....	101
Figure 5.9: Data Selection (June 2016, Test #1).....	102
Figure 5.10: Stitched Data (June 2016, Test #1)	103
Figure 5.11: FSS Percent Errors with Data Deletion (June 2016, Test #1)	104
Figure 5.12: Video Playback Tool (June 2016, Test #1)	106
Figure 5.13: Differences in FSS and JR3 Global Force Locations (June 2016, Test #1)	107
Figure 5.14: Average FSS Percent Errors at each Applied Force (June 2016, Test #1).....	108
Figure 5.15: Alignment Correction Maneuver.....	111
Figure 5.16: CBSS Fundamental Concepts (August 2016, Test #1)	112
Figure 5.17: FSS to JR3 Force-Torque Comparison (August 2016, Test #1)	113
Figure 5.18: Data Selection (August 2016, Test #1)	114
Figure 5.19: FSS Percent Errors with Data Deletion (August 2016, Test #1).....	115
Figure 5.20: Differences in FSS and JR3 Global Force Locations (August 2016, Test #1).....	117
Figure 5.21: Average FSS Percent Errors for both Test Regions (August 2016, Test #1).....	118
Figure 5.22: PSS to Leica Comparison (October 2016, Test #1)	122
Figure 5.23: Average PSS Percent Errors at each Leica Distance (October 2016, Test #1)	123
Figure 5.24: PSS to Robot Arm Comparison (October 2016, Test #2)	124
Figure 5.25: Data Selection (October 2016, Test #2)	124
Figure 5.26: PSS Percent Errors with Data Deletion (October 2016, Test #2)	125
Figure 5.27: Average PSS Percent Errors at each Arm Distance (October 2016, Test #2).....	126
Figure 5.28: Contact Scenarios	128
Figure 5.29: FSS Verification of Nominal Contact	129
Figure 5.30: FSS to JR3 Comparison (October 2016, Nominal Contact)	130
Figure 5.31: Data Selection (October 2016, Nominal Contact).....	131
Figure 5.32: FSS Percent Errors with Data Deletion (October 2016, Nominal Contact).....	132
Figure 5.33: Differences in FSS and JR3 Global Force Locations (October 2016, Nominal Contact).....	133
Figure 5.34: FSS Y-Axis Torque Error Ignoring Outliers (October 2016, Nominal Contact)...	133
Figure 5.35: Average FSS Percent Error per Region (October 2016, Nominal Contact).....	134
Figure 5.36: FSS Verification of Misalignment One.....	136
Figure 5.37: FSS to JR3 Comparison (October 2016, Misalignment One)	137
Figure 5.38: Data Selection (October 2016, Misalignment One)	138
Figure 5.39: FSS Percent Errors with Data Deletion (October 2016, Misalignment One)	139

Figure 5.40: Average FSS Percent Error per Region (October 2016, Misalignment One)	140
Figure 5.41 FSS Verification of Misalignment Two	142
Figure 5.42: FSS to JR3 Comparison (October 2016, Misalignment Two)	143
Figure 5.43: Data Selection (October 2016, Misalignment Two)	144
Figure 5.44: FSS Percent Errors with Data Deletion (October 2016, Misalignment Two).....	145
Figure 5.45: Average FSS Percent Error per Region (October 2016, Misalignment Two).....	146
Figure 5.46 FSS Verification of Misalignment Three	148
Figure 5.47: FSS to JR3 Comparison (October 2016, Misalignment Three)	149
Figure 5.48: Data Selection (October 2016, Misalignment Three)	150
Figure 5.49: FSS Percent Errors with Data Deletion (October 2016, Misalignment Three).....	151
Figure 5.50 Average FSS Percent Error per Region (October 2016, Misalignment Three).....	152
Figure C.1: FSS to JR3 Force-Torque Comparison (June 2016, Test #2).....	182
Figure C.2: FSS Percent Errors with Data Deletion (June 2016, Test #2)	183
Figure C.3: Average FSS Percent Errors at each Applied Force (June 2016, Test #2)	184
Figure C.4: FSS to JR3 Force-Torque Comparison (June 2016, Test #3).....	185
Figure C.5: FSS Percent Errors with Data Deletion (June 2016, Test #3)	186
Figure C.6: Average FSS Percent Errors at each Applied Force (June 2016, Test #3)	187
Figure D.1: FSS to JR3 Force-Torque Comparison (August 2016, Test #2)	188
Figure D.2: FSS Percent Errors with Data Deletion (August 2016, Test #2).....	189
Figure D.3: Average FSS Percent Errors for both Test Regions (August 2016, Test #2)	190
Figure D.4: FSS to JR3 Force-Torque Comparison (August 2016, Test #3)	191
Figure D.5: FSS Percent Errors with Data Deletion (August 2016, Test #3).....	192
Figure D.6: Average FSS Percent Errors for both Test Regions (August 2016, Test #3)	193
Figure D.7: FSS to JR3 Force-Torque Comparison (August 2016, Test #4)	194
Figure D.8: FSS Percent Errors with Data Deletion (August 2016, Test #4).....	195
Figure D.9: Average FSS Percent Errors for both Test Regions (August 2016, Test #4)	196
Figure D.10: FSS to JR3 Force-Torque Comparison (August 2016, Test #5)	197
Figure D.11: FSS Percent Errors with Data Deletion (August 2016, Test #5).....	198
Figure D.12: Average FSS Percent Errors for both Test Regions (August 2016, Test #5)	199
Figure D.13: FSS to JR3 Force-Torque Comparison (August 2016, Test #6)	200
Figure D.14: FSS Percent Errors with Data Deletion (August 2016, Test #6).....	201
Figure D.15: Average FSS Percent Errors for both Test Regions (August 2016, Test #6)	202
Figure D.16: FSS to JR3 Force-Torque Comparison (August 2016, Test #7)	203
Figure D.17: FSS Percent Errors with Data Deletion (August 2016, Test #7).....	204
Figure D.18: Average FSS Percent Errors for both Test Regions (August 2016, Test #7)	205
Figure D.19: FSS to JR3 Force-Torque Comparison (August 2016, Test #8)	206
Figure D.20: FSS Percent Errors with Data Deletion (August 2016, Test #8).....	207
Figure D.21: Average FSS Percent Errors for both Test Regions (August 2016, Test #8)	208
Figure D.22: FSS to JR3 Force-Torque Comparison (August 2016, Test #9)	209
Figure D.23: FSS Percent Errors with Data Deletion (August 2016, Test #9).....	210

Figure D.24: Average FSS Percent Errors for both Test Regions (August 2016, Test #9)	211
Figure D.25: FSS to JR3 Force-Torque Comparison (August 2016, Test #10)	212
Figure D.26: FSS Percent Errors with Data Deletion (August 2016, Test #10)	213
Figure D.27: Average FSS Percent Errors for both Test Regions (August 2016, Test #10)	214
Figure D.28: FSS to JR3 Force-Torque Comparison (August 2016, Test #11)	215
Figure D.29: FSS Percent Errors with Data Deletion (August 2016, Test #11)	216
Figure D.30: Average FSS Percent Errors for both Test Regions (August 2016, Test #11)	217

LIST OF TABLES

Table 2.1: Gain vs Resolution and Range [3]	31
Table 3.1: Load Cell Responses to Known Forces	36
Table 4.1: Nominal Contact Test Matrix	79
Table 4.2: Alignment Correction Test Matrix	80
Table 4.3: Proximity Characterization Test Matrix with Leica & T-MAC	81
Table 4.4: Proximity Characterization Test Matrix with Moto 50 Kinematic Models.....	82
Table 4.5: Nominal Contact Test Cases.....	83
Table 4.6: Misalignment One Test Cases	85
Table 4.7: Misalignment Two Test Cases.....	87
Table 4.8: Misalignment Three Test Cases.....	88
Table 5.1: Piecewise MATLAB Load Cell Models.....	93
Table 5.2: June 2016 Overall FSS Error Analysis	105
Table 5.3: June 2016 FSS Error Analysis for each Applied Force	109
Table 5.4: June 2016 FSS Error Analysis for each Applied Force (continued)	110
Table 5.5: August 2016 Overall FSS Error Analysis.....	116
Table 5.6: August 2016 FSS Error Analysis for both Test Regions	120
Table 5.7: Manually Collected Proximity Data (October 2016, Test #1).....	121
Table 5.8: PSS Percent Errors at each Leica Distance (October 2016, Test #1)	122
Table 5.9: PSS Error Analysis at each Arm Distance (October 2016, Test #2)	127
Table 5.10: Test Plan Error Analysis for Nominal Contact.....	135
Table 5.11: Test Plan Error Analysis for Misalignment One	141
Table 5.12: Test Plan Error Analysis for Misalignment Two.....	147
Table 5.13: Test Plan Error Analysis for Misalignment Three.....	153
Table 5.14: Load Cell Zero Drift and Total Drift for all FSS Tests	154

LIST OF ABBREVIATIONS

Abbreviation	Full Meaning	Abbreviation	Full Meaning
2D	Two Dimensional	IR	Infrared
3D	Three Dimensional	LB	Lower Bound
ADC	Analog-to-Digital Converter	MAX	Maximum
ALS	Ambient Light Sensor	MCU	Microcontroller Unit
API	Application Program Interface	MIN	Minimum
AVDD	Analog VDD (Power Source)	MIT	Massachusetts Institute of Technology
AVE	Average	MPE	Maximum Permissible Error
CAD	Computer Aided Design	NASA	National Aeronautics and Space Administration
CBS	Client Berthing System	NVM	Non-Volatile Memory
CBSS	Client Berthing Sensor System	OS	Operating System
CLR	Common Language Runtime	PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
CNC	Computer Numerical Control	PSS	Pose Sensing System
COTS	Commercial Off-The-Shelf	RAM	Random Access Memory
DOF	Degrees of Freedom	RSS	Rail Sensor System
FSAB	Full Satellite Aft Bulkhead	SCL	Serial Clock
FSS	Force Sensing System	SDA	Serial Data
FTS	Force-Torque Sensor	SPAD	Single Photon Avalanche Diode
GND	Ground	SSPD	Satellite Servicing Projects Division
GOES-12	Geostationary Operational Environmental Satellite-12	STD	Standard Deviation
GPIO	General Purpose Input/Output	T-MAC	Tracker-Machine Control Sensor
GSFC	Goddard Space Flight Center	TOF	Time of Flight
GUI	Graphical User Interface	UB	Upper Bound
I2C	Inter-Integrated Circuit	USB	Universal Serial Bus
IDE	Integrated Development Environment	VCSEL	Vertical Cavity, Surface Emitting Laser
INT	Interrupt	WVRTC	West Virginia Robotic Technology Center
IO	Input/Output	WVU	West Virginia University

LIST OF SYMBOLS

Symbol	Meaning	Symbol	Meaning
$\langle \rangle$	Vector	$f(u)$	Arbitrary Rotation Function on Point p
#	Number	F_A	Palm Tool A Force/Local Force A Measurement
%	Percent	F_B	Palm Tool B Force/Local Force B Measurement
	Absolute Value	F_C	Palm Tool C Force/Local Force C Measurement
$\sim$	Approximately	F_G	Global CBSS Force Measurement
+3v3	3.3 Volts	Hz	Hertz
<	Less Than	in	Inch
=	Equal To	in-lb	Inch Pound
>	Greater Than	kHz	Kilohertz
$\pm$	Plus or Minus	kN	Kilonewton
$\sum$	Summation Function	k Ω	Kiloohm
$\sqrt{\quad}$	Square Root	L	Length
$\leq$	Less Than or Equal To	lb	Pound
$\geq$	Greater Than or Equal To	LCF	Load Cell Force
$^\circ$	Degrees	LCR	Load Cell Response
μA	Microampere	m	Meter
μm	Micrometer	mA	Milliampere
μs	Microsecond	MHz	Megahertz
A	Ampere	mm	Millimeter
AE	Average Error of Percent Error Vector	ms	Millisecond
AF	Applied Force	mV	Millivoltage
Ar	Cross-Sectional Area	N	Newton
AT	Average Error of Trial Vector	n	Size of Trial Error Vector
b	Intercept	NaN	MATLAB's Not a Number Value
cm	Centimeter	Nm	Newton Meter
cos	Cosine Funtion	ns	Nanosecond
$\cos^{-1}$	Inverse Cosine Function	p	Arbitrary Point to be Rotated
D	Center-to-Center Distance Between CBSS Palm Tools	p'	Rotated Point Resulting from f(u)
D_z	z-distance from CBSS Contact Plane to Marman Ring Plane	P_A	Palm Tool A Proximity Measurement
F	Contact Force Magnitude	P_B	Palm Tool B Proximity Measurement

Symbol	Meaning	Symbol	Meaning
P_C	Palm Tool C Proximity Measurement	x'	x-Coordinate of Rotated Point p'
PE	Percent Error Vector	x_A	x-Coordinate of F_A
PE(i)	i th Element of Percent Error Vector	x_B	x-Coordinate of F_B
R	Resistance	x_C	x-Coordinate of F_C
S	Size of Percent Error Vector	x_G	x-Coordinate of F_G
s	Second	y	y-Coordinate of Arbitrary Point p
sin	Sine Function	y'	y-Coordinate of Rotated Point p'
$\sin^{-1}$	Inverse Sine Function	y_A	y-Coordinate of F_A
sl	Slope	y_B	y-Coordinate of F_B
TE	Trial Error Vector	y_C	y-Coordinate of F_C
TE(i)	i th Element of Trial Error Vector	y_G	y-Coordinate of F_G
T_X	CBSS x-Axis Torque	α	Angle between CBSS Contact Plane and Marman Ring Plane
T_Y	CBSS y-Axis Torque	θ_X	x-Axis CBSS Correction Angle
V	Voltage	θ_Y	y-Axis CBSS Correction Angle
v1	Vector 1 in CBSS Correction Angle Calculations	ρ	Resistivity
v2	Vector 2 in CBSS Correction Angle Calculations	ϕ	Rotation Resulting from f(u)
Vdc	DC Voltage	Ω	Ohm
x	x-Coordinate of Arbitrary Point p		

CHAPTER 1 | INTRODUCTION

1.1 Brief Overview

The National Aeronautics and Space Administration's (NASA) Goddard Space Flight Center (GSFC) and Satellite Servicing Projects Division (SSPD), along with other industrial partners, are actively working to produce a custom grapple system for on a satellite servicing vehicle. The grapple system, which is the platform for the student's research, is known as the Client Berthing System (CBS) and will allow the servicing vehicle to dock with a captured satellite on orbit. The CBS is one of the many systems being studied for a mission known as Restore-L. Due to the proprietary nature of the CBS and other mission sensitive information, the reader may be asked, at times, to envision certain CBS concepts because detailed pictures could not be provided. Most CBS related information was gained through NASA meetings and collaboration with GSFC and the SSPD.

The Restore-L Mission's core initiative is the on-orbit robotic satellite servicing of a government satellite [1] [2]. Technology developed for Restore-L will allow satellites to be serviced on-orbit as they fail, become damaged, need refueled, reach the end of their usable lifetime, etc [1] [2]. Specifically, Restore-L is a multiphase mission to service Landsat 7 in low earth orbit [3]. It involves launching a servicing vehicle to space, finding Landsat 7, capturing the satellite, refueling said satellite, relocating it, and then decommissioning the servicing vehicle [3]. An overview of the mission can be seen in Figure 1.1. Refueling and servicing tasks will be performed by teleoperators when Restore-L captures Landsat 7 [3]. The Restore-L servicing vehicle will be equipped with an array of tools for cutting space blankets and wires, removing caps, transferring fuel, and closing up the blanket [3].

Figure 1.1: Restore-L Objectives [3]

The CBS plays a pivotal role in the autonomous rendezvous and docking portion of the mission (phase #4 of Figure 1.1). When the servicing vehicle reaches Landsat 7's orbit and orients itself with the satellite, a robotic arm reaches out and grasps its Marman ring [1] [2]. This robotic arm will then slowly pull the satellite to the servicing vehicle and onto the CBS [1] [2]. This is a process known as space berthing, which typically involves a variety of alignment and grappling stages after an initial capture [4] [5]. In the grappling stage, the CBS is intended to interact with and mate to the satellite's Marman ring to dock the satellite to the servicer. Figure 1.2 shows this process and depicts the CBS and Marman ring locations. It can be seen that the CBS is adjacent to the robotic arms on the deck of the servicing vehicle and has three posts oriented on a diameter equivalent to that of the Marman ring. After the robotic arm that is reaching out grasps the asset and berths it to the servicing vehicle, gripping mechanisms (grippers) on the ends of the CBS posts physically hold the Marman ring and, by extension, the satellite in place. Knowledge of ring contact on the CBS posts is vital to trigger the capture drive mechanisms of those grippers. After the functionality of Marman rings clamps is formally discussed, the grippers will be explored in more detail.

Figure 1.2: Docking a Satellite to the CBS [1]

A Marman ring clamp system is a popular method of payload separation from a larger spacecraft [6]. An example of this system is shown in Figure 1.3. It can be seen that tension straps at the top and bottom apply a radial inward force on two rings [6]. The radial force created by these straps holds the two rings together in the “V-Segments” of the straps, which is how a Marman ring permanently fixed on a payload is mated to the ring on another spacecraft [6]. When the straps are released, the payload separates and moves away from the transport vehicle. Many satellites employ this technique to be released into orbit.

Figure 1.3: Marman Ring and Clamp [6]

Back to the CBS, one of the ground based prototypes of NASA’s mechanical platform is comprised of three identical grippers called “palm tools”, each mounted on a motorized linear guide assembly. The linear guides are connected together at the ends through a custom base and oriented 120° apart from one another, forming coplanar contact surfaces across the three palm tools. These palm tools are first seen in Figure 1.4 holding a Marman ring segment. The surfaces that the Marman ring rests on serve as the primary contact points for each gripper, and are thus commonly referred to as “palm tools”. The linear guides that the palm tools are mounted on safeguard against the inability to secure Marman rings of varying diameters by allowing the CBS palms to adjust accordingly in preparation for a capture. Each palm tool leverages a single pivoting jaw that is spring loaded to remain open thereby priming them for accepting an incoming Marman ring.

Figure 1.4: Marman Ring in Palm Tool Jaw

The capture operation initiates when a ring makes contact with all three palm tools, which will be referred to as “full contact” for the remainder of this document. “Nominal contact” is another heavily used term and indicates that the palm tools contact a ring with equal force across all three tools. While maintaining full or nominal contact, the palm tools are drawn inward on the linear guides by the capture drive motors to ease the Marman ring into the jaws. Each jaw is shaped in such a way that contact between the Marman ring and the lower portion of the jaw forces the uppermost part of the jaw onto the back profile of the ring itself. This design methodology provides a simple, straightforward tactic for clamping the Marman ring into the palm tools as they retract inward and establish a significant radial contact force between the jaws and the outside surface of the ring. Figure 1.4 is a Computer Aided Design (CAD) model that provides two views of a Marman ring section captured in the jaw of a palm tool. This offers a great visual of a CBS palm tool, the cross sectional Marman ring profile, and how the jaw makes contact with the ring’s outer ledge to clamp it to the palm. The jaw and palm are rendered in blue and yellow, respectively. The Marman ring and springs that hold the jaw open are metallic gray. Figure 1.4 shows palm tools used in this research that are now obsolete; the current tools or CBS could not be shown.

Areas of concern during the CBS satellite capture phase include determining if a Marman ring is inbound at an optimum orientation as well as determining initial Marman ring contact forces, radial forces on the ring by the jaws while securing it, forces indicative of a successful ring clamp down. Both initial and clampdown forces will be in the same direction but describe two unique events. Once a satellite is successfully mated with the CBS, continuous surveillance of this “docked” state is considered equally important. Applied forces and loads by robotic manipulators to service the asset may result in shifting or unexpected transfer loads to the

servicing vehicle. Active monitoring of the docked state and capture phase that precedes it will provide critical awareness to teleoperators and could even supply input data to autonomous algorithms. Providing innovative technology solutions and comprehensive sensory systems not only for the CBS, but the entire servicing vehicle in general, are imperative for mission success. West Virginia Robotic Technology Center (WVRTC) and West Virginia University (WVU) researchers have been investigating such solutions for NASA GSFC and the SSPD for some time. The WVU and WVRTC research team focuses on satellite servicing mission objectives and asteroid exploration with related research areas in sensor fusion, advanced sensor systems, robot control architectures, autonomy, machine vision, and relative navigation [7].

The current WVRTC prototype of the CBS, dubbed the Client Berthing Sensor System (CBSS), is a limited version of the real system described in this overview with sensor systems retrofitted to three dimensional (3D) printouts of replica CBS palm tools that are now obsolete. The mechanical platform of the CBSS predates the student's involvement with this project, but the overall geometry aligns well with the real system. However, the CBSS does not include the linear guide assemblies, which prevents simulation of Marman ring clamp down when captured. In place of the linear guide assemblies are three large T-slotted aluminum extrusion bars tightly fastened to a custom machined aluminum base that hold up three configurable posts (120° apart from one another). The base is concealed with a 3D printed cover that bears the trademark insignia of WVU. This prototype can be seen in Figure 1.5. It shows the mechanical platform without the palm tools attached and provides a nice representation of the geometry of both the CBS and the CBSS.

Figure 1.5: WVRTC CBSS Mechanical Platform

The CBSS post positions are configurable in the sense that they can be translated along the T-slot arms and manually tightened down. The post height can be altered as well via set screws that hold the posts to the post base pieces that sit atop the arms. Lastly, the 3D printed palm tools are secured to plates that are situated on top of the posts. The plates have an interface on the bottom that slide over the posts and are held in place, once again, with set screws. These were not shown in Figure 1.5. It should be noted that there are no fine tuning methods for precise post configurations. Palm tool height adjustments are facilitated entirely through set screws friction holding the posts in place, which is not the best method for ensuring coplanar palm tools. Palm tool radial adjustments are made by tightening the post bases to the arms via T-slot hardware, which is not the best method to ensure that palm tools lie on a common radius.

1.2 Purpose

One may find themselves asking why space asset servicing is even important or useful. With only two assets being designed with serviceability in mind (the Hubble Space Telescope and the International Space Station), it is clearly a challenge to even accomplish such a feat [8]. Nevertheless, there are over 1000 satellites in space that NASA, the US military, private sector companies, and many others rely on every day [9]. All stakeholders would like to see the operational lives of their satellites extended for as long as possible [9]. Satellite servicing is the solution and provides technology to repair, refuel, relocate, and upgrade satellites on-orbit. Furthermore, technology developed for Restore-L will be applicable to other space endeavors. Possibilities range from in-space assembly of large stations, observatories, and interplanetary ships; sending lighter payloads with less fuel and more features into space with the intention of servicing them later; providing fuel depots for deep space missions like the journey to Mars; and removing space debris [2]. These are some of the reasons why evolving the CBS, and other related on-orbit servicing technologies, is extremely important.

1.3 Scope

This thesis is primarily on the WVRTC prototype CBSS, the sensor systems the student equipped to strategic areas on the replica CBS palm tools, and the resulting performance.

1.4 Research Goals

The main idea is to extend the functionality of the WVRTC CBSS to include proximity sensing and contact force sensing. NASA only has binary switches that are triggered by “flapper” attachments on the sides of the current CBS palm tools. The flappers only allow a Marman ring to be detected close to the palm tools (≤ 18 mm), but no further information can be gained from the system. Proximity sensing will allow the CBS to calculate the pose/alignment of an inbound Marman ring. Force sensing will allow knowledge of initial contact with the ring, determination of ring clampdown, and provide insight concerning shifting loads under sustained contact. Data generated by these system augmentations can be viewed by teleoperators, or could even be input into autonomous algorithms, for correcting satellite pose to the real CBS, providing knowledge of initial contact, indicating successful capture, and offering real time monitoring of torque seen on a servicing vehicle as robotic manipulators work on a client satellite.

The pros of the binary side flapper system are simplicity and reduction of electrical components, while the con is that it only indicates when a Marman ring can be captured. It provides no state information before or after initial contact, thus the Marman ring could be inbound at an off-nominal orientation, ring clampdown cannot be detected if somehow it is actually captured, and no force information under sustained contact (when docked) is available. There is clearly a need for an improved system, which is the justification for this research. The student’s specific objectives are:

1. To design and build a CBS force sensing system (FSS)
2. To design and build a CBS pose sensing system (PSS)
3. To develop a graphical user interface (GUI) monitoring application for showcasing the FSS and PSS functionalities
4. To develop a lightweight software application for logging purposes (without graphics overhead)
5. To evaluate the performance of the FSS and PSS

1.5 Approach

Readily available commercial off-the-shelf (COTS) sensors and the necessary electronic hardware for sampling were incorporated into replica CBS palm tools with a design methodology focused on minimally invasive solutions; this enabled pose estimation and force sensing on the WVRTC CBSS. Retrofitted solutions based on that design methodology were ideal because major modifications to the CBS palms were controlled by NASA and unlikely to occur for only a sensory system. The foundation of the FSS design was Omega beam style load cells for measuring forces as the palm tools interacted with a Marman ring. The foundation of the PSS design was Inter-Integrated Circuit (I2C) VL6180X range sensors for measuring palm tool proximities to a Marman ring. Figure 1.6 gives an idea of the retrofitted sensor systems that were created. The outer, green attachments were rail sensing system (RSS) modules with load cells embedded internally, and the array of square sensors in the palm cavities was the range sensors. It was envisioned to have a grid of the range sensors in the palm tool cavities and ultimately that will happen. However, a single range sensor was embedded in a front center cavity for this research, which included proof of concept testing while refining the prototype sensing systems.

Figure 1.6: Retrofitted Sensor Systems

The first step involved in creating the FSS was the characterization of the load cells on Phidgets sampling hardware by correlating known forces to voltage signals. Proprietary PhidgetsBridge software was employed to sample the load cells in calibration and led to the

development of raw data-to-force conversion software models. Software was then developed that leveraged the sampling hardware's application programming interface (API) to capture data from the load cells. With the load cell models and the sampling software, an RSS had to be designed. RSS modules (green components in Figure 1.6) were envisioned as fixtures that attach to the sides of a palm tool that each house two load cells. Load cell bottom mounts were screwed into the RSS load cell housings that physically mount to palm tools, and cover rails attached to load cell top mounts to provide a Marman ring contact surface. The FSS, with regards to a single palm tool, was two RSS modules on each side of the tool for a total of four load cells. Each RSS top rail surface followed the profile of the palm and rested just above it when mounted, allowing contact to occur on the rails rather than the palms. The FSS, with regards to the CBSS, was all of the RSS modules across the three palm tools, for a total of six RSS modules and 12 load cells. The last steps to complete the FSS were 3D printing the RSS modules, embedding the load cells, mounting them to the palm tools, and wiring them to the sampling hardware.

The first step in creating the PSS was to develop embedded system software for sampling from the sensors over the I2C protocol and relaying that data to a serial port. Proximity data from the sensors was in millimeters, so there were not any characterizations necessary to correlate voltage signals to known distances as required by the load cells. A simple snap-in fixture was then designed and 3D printed for the front cavities of the palm tools. This fixture allowed for the I2C sensors to be embedded into the cavities. For sake of simplicity, the PSS, with regards to a single palm tool, was a single range sensor. The PSS for the global CBSS system was three range sensors across the three palm tools.

With the new capabilities provided by these sensing systems, software solutions for capturing, logging, and presenting the sensor data in useful ways had to be created. A Windows GUI was developed to showcase the PSS and FSS capabilities. The software was aimed at providing critical feedback to remote operators in ground based training applications because systems utilizing COTS hardware are not typically qualified for hazards associated with the space environment. The first step in GUI development was the integration of FSS and PSS software to acquire sensor input for the GUI. The sampling hardware API for the FSS was used, and PSS data provided by an embedded sampling system was read over a serial port. Individual readouts for each load cell and range sensor were updated in the GUI as sensor data became

available. Readouts and a force animation were made to track the resultant global force imparted by the three tool system. Real time plotting of palm tool proximity to a target was also provided. Based on the proximity data, x- and y-axis pose correction angles were calculated and displayed to the user. A logger was inserted for recording raw sensor data for post processing. Lastly, a separate software package was developed for a Linux system for logging raw sensor data, without the overhead associated with the GUI software package.

Once the sensor systems and monitoring software were completed, the performance was then evaluated. The replica CBS palm tools, with retrofitted sensor systems, were attached to the three configurable posts on the WVRTC CBSS mechanical platform. Testing was carried out under different Marman ring approach and contact scenarios with industrial robots and metrology devices at the WVRTC. The CBSS was attached to a robotic arm with a force-torque sensor (FTS) and eased into contact with a Marman ring in various alignments while data was collected. Precise robotic joint movements and a laser tracker were used to verify accuracy of the PSS proximity sensors, and the FTS was used to verify the accuracy of the FSS load cells.

To clarify once more in case it was not clear, the WVRTC system is the CBSS and the FSS/PSS sensor systems developed by the student were tested on the CBSS. The student's work with those sensor systems was to be used as a justification for integration into the actual CBS. Even though COTS sensors were employed and, in general, can only be used in ground based testing, the applications explored useful sensing techniques, proof-of-concept testing, and provided insight for technology development. Furthermore, these systems addressed all areas of concern with the satellite capture phase and the docked state, with the exception of determining the radial forces created by the palm tool jaws on the Marman ring. The binary side flappers NASA currently uses cannot provide near the information to teleoperators that the PSS and FSS can.

1.6 Tools & Technologies Used

A list of all tools and technologies used to carry out the student's research is given below. It is intended to foreshadow the technology throughout this document. Further details of each item and how they were applied to the FSS or PSS will be discussed in the remaining chapters.

- SolidWorks 2016-17 for modeling all sensor fixtures
- Stratasys uPrint SE Plus 3D printer for building sensor fixtures.
- Omega LCL-40 full bridge thin beam load cells for force measurements
- Four-Input PhidgetBridge ID 1046_0 for sampling Omega force transducers over the Universal Serial Bus (USB)
- Omega LCM-CL1 mounting kits for attaching load cells inside RSS modules
- Shimadzu Autograph AGS-X 10 kN stress analyzer for precise load cell calibration
- VL6180X time of flight (TOF) laser range sensors for proximity sensing over I2C
- STMicroelectronics STM32 Nucleo-64 development board with an on-board STM32L053R8 MCU for sampling range sensors and relaying data over USB
- ARM Mbed online compiler for programming the Nucleo development board
- Custom CBSS mechanical platform and replica GSFC CBS palm tools for testbed
- Windows 7 operating system (OS), Visual Studio 2015, Windows Presentation Foundation (WPF), and C# programming language for creating a GUI
- Ubuntu 16.04.1 OS, GCC 5.4.0 compiler, C++ programming language, and a standard text editor for creating a lightweight logging software
- Leica Absolute Tracker AT901-MR with reflectors for configuring CBSS posts and validating the PSS
- Leica T-MAC TMC30-B for tracking robotic arm Cartesian position and validating the PSS
- Full Satellite Aft Bulkhead (FSAB) on the YASKAWA Motoman-MH250 II robotic arm for providing a target feature with a Marman ring
- YASKAWA Motoman-SIA50D robotic arm equipped with a JR3 160M50A4-180-DF Force-Torque sensor (FTS) for mounting the CBSS and validating the FSS and PSS
- MATLAB R2017a for post processing test results

1.7 Thesis Organization

The remainder of this document is divided into five chapters. An overview of each chapter is given below.

Chapter 2 / Literature Review – This chapter focuses on previous research that has led up to the current project. It also gives some important background information/operating principles of the hardware featured in this research.

Chapter 3 / CBSS Design and Concepts – This chapter covers the design of each CBSS sensor system (the PSS and FSS) and the concepts behind how they work. Sensor system attachments to the palm tools are showcased, the exact placements of those systems on the CBSS are given, and the mathematical calculations that govern them are discussed.

Chapter 4 / CBSS Software Tools and Experimental Overview – This chapter covers all the CBSS software used and the lab test equipment for validating the prototype. Test plans and procedures intended to validate the operations of the FSS and PSS subsystems are also discussed.

Chapter 5 / Results – This chapter covers how experimental data was processed. It also contains an analysis for every CBSS test performed over the duration of this research.

Chapter 6 / Conclusion – This chapter provides a restatement of goals, summary of findings, and how the work can be extended.

CHAPTER 2 | LITERATURE REVIEW

This chapter will focus on previous research that has led up to the current project and give some important background information/operating principles on the hardware featured in this research.

2.1 Previous Work

Work with NASA GSFC, the SSPD, and WVU/WVRTC has been ongoing since 2011 with respect to the CBS and other tool grippers. The WVRTC has focused on retrofitting sensor systems to custom grappling tools created by NASA and other industrial partners that were intended to latch onto a satellite's Marman ring. In previous years, the tools were referred to as gripper tools rather than palm tools. Research focused on the initial capture phase of a servicing mission where a robotic arm extends out to capture a satellite and berth it to a servicer. The end effector/gripper tool on the robotic arm is used to grab a Marman ring on a satellite to capture the asset. The migration from one gripper tool to three palm tools and the WVRTC CBSS started with the student's research project in 2015. However, the CBSS mechanical platform that provided the testbed for the student's research was fashioned in 2014/15. The next few sections provide a recap of previous work and sensing approaches.

2.1.1 WVRTC Research from 2011

The first system, shown in Figure 2.1, featured two embedded planar beam load cells in a main housing that supported two independent force plates. This was designed to rest between the gripper NASA was using at the time. These plates provided two force measurements and a one dimensional reconstruction of the contact force location from Marman ring contact. A miniature camera was mounted in the main housing to visualize the ring during approach. Machine vision techniques were employed to interpret the relative depth to the ring and overall orientation. An infrared (IR) sensor was utilized to supplement machine vision by providing an additional distance measurement to the ring. Research later in the year focused on different load cell types and configurations under the force plates to allow two dimensional contact force reconstruction.

Figure 2.1: Proposed 2011 System

2.1.2 WVRTC Research from 2012

The next system featured multiple sensing approaches and is shown in Figure 2.2. The WVRTC active jaw gripper was fitted with IR sensors to detect when Marman rings break a plane indicating ring capture can begin. It also utilized two laser range finders to detect distance to a ring and the overall orientation on approach. The “Palm Rail System” was the first RSS iteration and housed two beam load cells in each rail. They detected initial contact with a ring and monitored transfer loads while it was captured. They allowed the magnitude of the resultant contact force to be sensed along with its two dimensional x-y-position. RSS prototypes are called rail sensing systems because they are side rail attachments for sensing force. Capacitive touch sensors were also included in this tool’s jaw for detecting clamping force.

Figure 2.2: 2012 Gripper and Sensor Systems

2.1.3 WVRTC Research from 2012-2015

Focus from 2012 to 2015 was dedicated to developing the RSS to match smaller form factors as real-estate on the NASA gripper tools grew smaller and smaller. Figure 2.3 shows this progression. The main idea was to scrap the beam load cells in the 2012 RSS and explore smaller options. Round load cells with a 9.6 mm (0.38 in) diameter were embedded into the RSS variations from the 2012/13 version onward. The culmination of the 2014/15 work resulted in the smallest RSS modules to date. Two of the small round load cells were embedded into the bottom structure of an RSS module. The load cells were preloaded through a top rail that was secured in place with one friction resistant shoulder screw. This RSS version experienced issues with the load cells not returning to zero after various directional forces were applied. Forces that did not contact the top rail in an orthogonal fashion twisted the top rail. This resulted in the preloaded force fluctuating due to tightening or loosening the screw. Rather than returning to zero force when unloaded, the load cells registered a value above or below the ideal preload based on how the rail was twisted. A three arm CBSS prototype was fashioned after the completion of the 2014/15 RSS and provided the platform for the sensor systems in this paper. The CBSS system featured three adjustable posts for mounting RSS modules that were oriented 120° apart from one another through a custom base and t-slotted aluminum extrusion bars.

Figure 2.3: Research from 2012 Onward

Figure 2.4 is provided to expand on the 2014/15 work. Pictures 1, 2, and 3 all show RSS prototypes being manufactured. The button load cells fit in the round cavities that can be seen in picture 2. Pictures 4-7 all pertain to the CBSS development. Picture 4 shows the center CBSS base plate being machined that holds the aluminum CBSS arms 120° apart. A completed post on the end of an arm, with RSS rails attached, can be seen in picture 5, while picture 7 shows the post base being machined. The arm posts allow sensor systems (in this case the 2014/15 RSS) to be adjusted vertically via set screws and horizontally along the arm lengths with t-slot hardware. Finally, picture 6 shows the entire CBSS system mounted to a robotic arm for testing. Force sensing remained largely untested in a global (three post, six RSS) configuration; only single post testing with two RSS modules was completed.

Figure 2.4: 2014/15 Work Expanded

2.2 Hardware Background – VL6180X & Nucleo

2.2.1 VL6180X Sensor

The VL6180X sensor was chosen for this research because it met the size requirements defined later in Section 3.2. These sensors are developed by STMicroelectronics and were the smallest time of flight sensors on the market with a footprint of 4.8 mm x 2.8 mm x 1 mm until they made a new version (the VL53L0X) at 4.4 mm x 2.4 mm x 1 mm [10]. These sensors are

based on a patented technology by STMicroelectronics that they call FlightSense [10]. Due to the proprietary nature of the product, not much information is given about FlightSense. They do, however, say that the VL6180X contains a Vertical Cavity Surface-Emitting Laser (VCSEL) as the emitter, a Single Photon Avalanche Diode (SPAD) array for sensing when reflected light from the emitter returns, and an Ambient Light Sensor (ALS) in an all-in-one package [10].

STMicroelectronics claims that despite the reflectance of the material, the VL6180X sensors are capable of measuring the true distance to an object where other IR sensors may struggle [11]. The reason that reflectance is not an issue for these sensors is because they are based on time of flight principles. Instead of estimating the proximity to a nearby object based on the amount of reflected light (which can be affected by surface color and reflective properties), the VL6180X measures the precise travel time of the light to an object and back to provide a distance measurement [11]. This is the principle idea of any time of flight system. Figure 2.5 shows the basics of FlightSense. Basically, the VCSEL sends photons towards a target and the SPAD array determines when photons from the partially reflected light returns [12]. The delay between the transmitted and received light pulse is measured and converted to a distance [12]. This distance is the photon travel time divided by two and multiplied by the speed of light. To give an idea of how fast this sensor operates, it is capable of measuring a 0.13 ns time delay at 20 mm and a 1.3 ns delay at 200 mm. To achieve time of flight in a sensor of this size is an impressive feat in engineering.

Figure 2.5: Time of Flight Concept [12]

Furthermore, the VL6180X sensor is capable of ranging up to 100 mm with a 1 mm resolution, but can go further depending on target reflectance and other external conditions such as ambient light and temperature [13]. The accuracy of the sensor is ± 10 mm [12]. It also works to reject ambient light except for the emitter's 850 nm wavelength light [13]. The sensor should be powered with 2.7-2.9 Vdc and operated in an environment from -10 °C to 60 °C [13]. It draws less than 1 μ A at standby and an average of 1.7 mA when performing a range measurement, which makes it a low power device requiring approximately 4.8 mW or less [13]. The sensor also has a 25° cone for emitting and viewing light [13]. This makes the sensor somewhat forgiving when it comes to detecting an object in front of it at varying orientations. Figure 2.6 shows the view cone and the illumination cone of the sensor. The ambient light sensor was not used in the CBSS application so it will not be discussed.

Figure 2.6: Conical Properties of VL6180X On-Board Sensors [13]

The diagram shown in Figure 2.7 provides an overview of what each sensor contains. Aside from the ambient light sensor, IR emitter, and proximity sensor already mentioned, the VL6180X has a built-in MCU along with random access memory (RAM) and non-volatile memory (NVM) to store default sensor configuration information [14].

Figure 2.7: Sensor Architecture [13]

Control, setup, and reading of the VL6180X sensor is facilitated over 400 kHz I2C by a host processor with the MCU in Figure 2.7 [13]. The host processor is the master and the VL6180X is the slave device. I2C communication with the sensor utilizes standard 7-bit addressing. 7-bit addressing is an 8-bit scheme where the address of the slave is in the first seven bits of a string, with the last one being a read or write bit (see Figure 2.8) [15]. If the read/write bit is 1, the master device will read from the slave designated by the address [15]. If the read/write bit is 0, the master will write to the slave device [15]. The VL6180X sensor has a default I2C address of 0x29 (hexadecimal), thus 0x52 is used when writing to the sensor and 0x53 is for reading from it [13].

Figure 2.8: I2C Addressing [15]

2.2.2 Mbed and the STM32 Nucleo-64 Development Board with STM32L053R8 MCU

The master processor selected to interface with the VL6180X sensors was the STM32 Nucleo-64 development board with an STM32L053R8 MCU. The MCU has a high-performance Arm Cortex-M0+ 32-bit RISC core operating at 32 MHz [16]. This development board is a breakout board that leverages the MCU's functionalities and provides a USB connection for powering, programming, and communicating with the device. This board was selected because it was part of a low power evaluation package that came with VL6180X sensors. Like most other general purpose MCU's, a large amount of peripherals are present. The most important leveraged for this research were the I2C communications, USB, and GPIO. The specific features of the development board and MCU will not be gone over in depth, but can be found at [17] [16]. Figure 2.9 shows the Nucleo hardware.

Figure 2.9: Nucleo L053R8 [17]

For programming on the development board, the Mbed online compiler was used; a link to Mbed was embedded in the Nucleo. Mbed also provided a direct API for interfacing with not only the selected MCU's peripherals, but all Arm Cortex-M MCUs [18]. After creating an account, one can select their hardware platform (in this case, the Nucleo L053R8) and begin developing code. Mbed has plenty code examples and the API allows access the Nucleo's peripherals through the device's pin names. Figure 2.10 shows a subset of the Nucleo's pins, including the ones used in this research. The pin names may also be referenced later so it was convenient to introduce them here. Flashing code to the Nucleo board involves dragging and dropping compiled binary code from Mbed onto the MCU from Windows. The MCU appears as a file system similar to a USB flash drive. Just find the device and copy code to it.

Figure 2.10: Nucleo Pinout 1 [19]

2.2.3 COTS VL6180X Hardware on a Nucleo L053R8

In a typical application, the wiring of the proximity sensor is given by Figure 2.11. The recommended general purpose input output (GPIO) pullup resistors are 47 k Ω , and the I2C pullup resistors should be attached on the host processor's side (Nucleo) [13]. The I2C pullups will vary depending on the application and must be sized to ensure proper operation.

Figure 2.11: Typical Sensor Setup [13]

The purchased sensors included VL6180X chips on small boards called satellite boards. Figure 2.12 shows the layout of one of these boards. It contains the sensor itself and some necessary electrical components as seen in Figure 2.11. Figure 2.12 shows that a regulator is present; this converted the Nucleo's input voltage from 3.3 V to 2.8 V to power the sensor. The serial clock (SCL), serial data (SDA), ground (GND), interrupt (INT), GPIO, and input voltage (AVDD) pins are all shown as well. Standard 2.54 mm header pins create connection points to the sensor. The size of the board is 25 mm x 9 mm, which is still a small sensor footprint.

Figure 2.12: Satellite Board Layout [20]

Figure 2.13: Satellite Board Schematic [20]

Figure 2.14: Nucleo VL6180X Expansion Board [21]

Figure 2.15 shows a schematic for the board connections in Figure 2.14. The pullups recommended in Figure 2.11 are applied to pins 1 and 3 for each sensor. This schematic refers to the GPIO pin on the satellite board as GPIO0_X, where X is L, R, or B based on the left, bottom, or right sensor in Figure 2.14. Any GPIO1 connections can be ignored. They are interrupts and are not used for the CBSS application.

Figure 2.15: VL6180X Expansion Board Satellite Connectors [21]

Figure 2.16 shows the GPIO expander on the board in Figure 2.14. This expander controls the segments of the digital display and provides the left, bottom, and right GPIO signals for the proximity sensors. One can see that the GPIO0_B, GPIO0_L, and GPIO0_R signals from Figure 2.15 come from this chip. Also, it is shown that the A0, A1, and A2 inputs are grounded. This indicates that the I2C address of the expander is 0x42 (0x84/0x85 in 7-bit addressing) [22].

Figure 2.16: GPIO Expander on Nucleo VL6180X Expansion Board [21]

2.3 Hardware Background – Load Cells & Phidgets

2.3.1 Load Cell Fundamentals

Mechanical strain is defined as the amount an object deforms from an applied force [23]. A strain gauge is a device that, when attached to an object, can measure strains based on the change in its electrical resistance [23]. Resistance is given by Equation (2.1), where R is the resistance, ρ is the resistivity, L is the length of the material, and Ar is the cross-sectional area [24].

$$R = \rho \frac{L}{Ar} \quad (2.1)$$

When a length of metal is stretched, it becomes thinner and elongated, which increases the resistance of the material [25]. If it is compressed without buckling, it will widen and become shorter [25]. Those physical changes decrease resistance. A common type is the metal foil strain gauge, consisting of thin metal strips on a nonconductive substrate that gets bonded to another object [25]. When the object is stressed by a force, the strain gauge will react accordingly by a change in resistance. Figure 2.17 shows an example of a bonded strain gauge. The resistance is measured between the two points indicated in the diagram. One can see that tension and compression forces will affect the gauge, but lateral forces have negligible influence.

Figure 2.17: Bonded Foil Strain Gauge object [25]

Typical strain gauges are mounted to rigid objects and are not allowed to flex much. Large forces that create large resistance changes in a strain gauge could permanently deform the object it is attached to or even damage the gauge itself [25]. Resistances of strain gauges usually range from 30 Ω to 3 k Ω when static and will only change by a small fraction under deformation [25].

To measure this phenomenon, a common circuit known as a Wheatstone bridge is applied. Figure 2.18 shows an example of this circuit with a strain gauge as one of its resistors. From an electronics standpoint, this circuit is just two parallel voltage dividers. Typically, R_2 is sized the same as the strain gauge under no load, and $R_1 = R_3$ [25]. With the strain gauge not stressed, the expected voltage (V) should be zero. If the gauge's resistance changes from force input, then V will react as a function of the strain. An inadequacy of this circuit is that temperature will affect the output voltage. Resistivity is a measure of how easy electrons can move through a given material and is influenced by temperature [24]. If the strain gauge in Figure 2.18 is unstressed but resistance is fluctuating due to temperature, it will cause an output voltage when it should be zero. This method is not very stable for sensing force and is not very popular.

Figure 2.18: Quarter Bridge Strain Gauge Circuit [25]

A better configuration is known as the half bridge and is shown in Figure 2.19. This configuration does two things: it compensates for temperature fluctuation and allows for larger response voltages for a given strain [25]. Suppose the strain gauges in Figure 2.19 have the same resistance, are not stressed, and $R_1 = R_3$. Temperature will still affect both strain gauges' resistances. However, the temperature will affect them both in the same manner and keep the right voltage divider proportional to left one, meaning $R_1/R_3 = R_{\text{Upper Gauge}}/R_{\text{Lower Gauge}}$. This significantly decreases the effects of temperature on the measured output voltage and keeps it closer to 0 V when the gauges are not stressed. Figure 2.19 also shows that bonding strain gauges in complimentary configurations is beneficial. If one gage is compressed from a strain and the other is tensioned by the same strain, the resistance differential will be larger, resulting in a larger output voltage. This is beneficial for making a strain gauge system more sensitive to both incremental loads and overall forces [25].

Figure 2.19: Half Bridge Circuit (left) and Physical Example (right) [25]

Finally, Figure 2.20 shows another configuration for using strain gauges that is even better than the half bridge implementation. This method requires four strain gauges where the left and right voltage dividers are proportional when the gauges are not stressed. The four gauges must also be exposed to the same strain, and they must be bonded to a rigid object such that the left voltage divider is complimentary to the right voltage divider. This means that the upper-left and lower-right gauges should be strained one way (for example in tension), and the other two should be strained the opposite way (in compression) for a given force. The previous methods' output voltages involved subtracting a changing voltage that was a function of the applied force from a fixed reference on the left voltage divider. In the full bridge configuration, both node voltages vary with force and are driven further apart. This creates an even larger voltage differential than the previous methods, thereby increasing the sensitivity to smaller loads and overall forces. Strain gauges that are packaged together in the configurations discussed that are bonded to mechanical units in strategic locations for measuring deformation are known as load cells [25].

Figure 2.20: The Full Bridge Strain Gauge Circuit [25]

2.3.2 Load Cells for this Research

LCL-40 full bridge thin beam load cells by Omega were selected for integration into the CBS palm tools and have a rated capacity of 177.9 N (40 lb) [26]. They were selected because they had the highest capacity in the thin beam load cell product line and were small enough that two could be placed end to end in some type of RSS fixture that would mount to the palm tool sides. The highest capacity force sensor was selected because typical contact force magnitudes when berthing a satellite to a servicer were not exactly known. Omega also sells mounting kits for these cells that allow them to be loaded properly and attached to custom applications through standard 10-32 fastening hardware. Figure 2.21 shows a few of these load cells between the LCM-CL1 mounting kits.

Figure 2.21: LCL-40 Load Cells and Mounting Kits

These sensors operate on 5 Vdc nominally but can go to a maximum of 12 V with a rated output of $2 \text{ mV/V} \pm 20\%$ [26]. This means the mV/V output signal at full load (177.9 N) can be expected to be between 1.6 and 2.4 mV/V. Furthermore, they have an output of 0.3 mV/V under no load conditions and a combined error of 0.25% of the full load [26]. Armed with this knowledge, one could produce a calibration line that would change a mV/V signal to a force in N by using data points (0.3 mV/V, 0 N) and (2 mV/V, 177.9 N). The line equation between these data points would provide a model for converting load cell responses to forces, but would be subject to the previously mentioned errors. An end user calibration with known weights provides a more accurate/repeatable model for force conversion than a model based on the manufacturer's specifications. This was performed for each load cell and is shown later in Section 3.1.2.

With signal to force models after an end user calibration, that 0.25% error at full load (177.9 N) would correspond to an accuracy of ± 0.44 N (± 0.1 lb) under ideal conditions. Because of hysteresis effects in the load cells and other factors, one could expect measured loads/forces to be correct within ± 0.44 N. Additionally, these load cells have an operating temperature of -54 to 93° C and can be loaded up to 266.9 N (60 lb) before properties of the cells are subject to permanent change [26].

2.3.3 Sampling Hardware

The selected sampling devices for the load cells were PhidgetBridge 4-Input 1046_0 boards because of their versatility (Figure 2.22). The student's familiarity with Phidgets brand devices from previous projects also factored into this decision. The student found from previous experiences that Phidgets brand devices are well documented online, have friendly APIs for many common programming languages, and plenty of code examples. That allows users to get started with the hardware with minimal setup time. This particular board was able to supply a 5 Vdc excitation voltage to four load cells and sample with a 24 bit resolution analog-to-digital converter (ADC), at a rate of 8-1000 ms via USB [27].

Figure 2.22: Load Cell Sampling Hardware [27]

Another feature available through the API is an adjustable gain. The Phidgets user guide recommends the highest possible gain be used that still encompasses the rated output of the connected load cells [28]. Table 2.1 provides the resolution and range at each gain setting. The LCL-40 Omega cells are rated for 2 mV/V, so a gain of 128 was used for all CBSS software packages discussed in Chapter 4. The ± 7.8125 mV/V range entirely covers the LCL-40 rated output.

Table 2.1: Gain vs Resolution and Range [3]

Gain	Resolution	Range
1	119 nV/V	± 1000 mV/V
8	14.9 nV/V	± 125 mV/V
16	7.45 nV/V	± 62.5 mV/V
32	3.72 nV/V	± 31.25 mV/V
64	1.86 nV/V	± 15.625 mV/V
128	0.93 nV/V	± 7.8125 mV/V

CHAPTER 3 | CBSS DESIGN AND CONCEPTS

The design of the FSS and PSS subsystems on the CBSS, which was the focal point of this research, will be covered here as well as the fundamental concepts of operation.

3.1 Design of the FSS

With WVRTC focus shifting to the CBS in 2015, it was sought after to retrofit similar style sensory systems from the previous research described in Chapter 2 to the three palm tools on the actual CBS. Figure 3.1 shows a CAD replica of the 2015/16 palm tool design that NASA was using on the CBS when this research began. This palm tool design was much different than the jaw grippers that functioned like a vice. Together, three palm tools clamp a Marman ring to the palms through pivoting tool jaws (shown in black) when all three tools are retracted inward on motorized linear slide mechanisms. The underside portion of the palm tools were built into those linear slides, so the usable real-estate for embedding sensors included the sides and the palm cavities. New RSS attachments had to be developed for this tool geometry. The new RSS attachments spanned the lengths of the palms and allowed for embedding larger load cells. The small round load cells employed to decrease the form factor in previous RSS versions from Chapter 2 were swapped out for beam type load cells like the ones featured in the 2012 RSS design in Figure 2.3. One RSS per side, meaning two RSS per palm, for a total of six RSS across the three palm tool CBSS prototype make up the FSS. Two beam cells from Section 2.3.2 were used for each RSS for a total of 12 load cells.

Figure 3.1: 2015/16 Replica of CBS Palm Tool

3.1.1 Phidgets Wiring Diagram

Three of the replica palm tools from Figure 3.1 were 3D printed for designing and fitting RSS devices to them. These three palm tools are referenced as palm tools A, B, and C throughout this paper. The geometry and locations will be discussed further as the FSS design is presented. It is, however, important to at least define them here. The 12 load cells across the three palm tools required three PhidgetBridge boards from Section 2.3.3 for gathering CBSS load cell response data. The wiring diagram is shown in Figure 3.2.

Figure 3.2: Phidgets Wiring Diagram

The specific serial number of the PhidgetBridge board that is on each palm tool is labeled in Figure 3.2. The color of each load cell wire corresponds to the actual load cell wire colors. Red is 5 Vdc, black is ground, and gray is the shield wire that is tied to ground. The green and white wires on the “+” and “-” inputs are the load cell signal lines. Numbers 0-3 between the “+” and “-” symbols designate the bridge input number. Load cells A1-A4, B1-B4, and C1-C4 connect to inputs 0-3 on PhidgetBridge #476983, #404917, and #410770 respectively. The labels for specific palm tool load cells are simply the concatenation of the palm tool letter with the load cell number, hence the A1-A4, B1-B4, and C1-C4 naming convention. Lastly, all three of these boards are connected via USB to a powered hub that connects to a computer running the CBSS software outlined in Chapter 4.

3.1.2 Load Cell Calibration on Shimadzu

To create accurate signal-to-force load cell conversion models, an end user calibration had to be performed. A Shimadzu Autograph AGS-X 10 kN stress analyzer was used to apply precision forces onto the load cells while they were wired to the PhidgetBridge boards. This machine has an accuracy of $\pm 0.5\%$ of the applied force [29]. The load cells were characterized from zero to their full loads at 177.9 N (40 lb) in 44.5 N (10 lb) increments and then back to zero at the same intervals. One could expect a maximum error of ± 0.89 N (± 0.2 lb) on the Shimadzu while calibrating. Part of the test setup can be seen in Figure 3.3.

Figure 3.3: Calibrating on Shimadzu

Not pictured in Figure 3.3 is the computer used to read the PhidgetBridge devices. Upon downloading the software libraries necessary to interact with the Phidget devices at [30], configuration software from the Phidget Control Panel can be run upon plugging a Phidgets brand device into a computer via USB. Figure 3.4 shows this monitoring software for the PhidgetBridge boards. The mV/V bridge values at every applied Shimadzu force were manually recorded for all 12 CBSS load cells.

Figure 3.4: PhidgetBridge Monitoring Program

The results of the calibration are shown in Table 3.1. Information in this table can be used to create force conversion models for each load cell. The convention the table follows for finding a particular load cell's data is to look where rows for a specific palm tool intersect the load cell columns. For instance, load cell A2's calibration data is located where the rows in "Palm Tool A" and the "Load Cell 2" column all intersect. A simple line equation between any two signal force pairs for a desired load cell forms a conversion model over that particular force range.

Table 3.1: Load Cell Responses to Known Forces

	Applied Force (lb)	Load Cell Response (mV/V)			
		Bridge Input 0 (Load Cell 1)	Bridge Input 1 (Load Cell 2)	Bridge Input 2 (Load Cell 3)	Bridge Input 3 (Load Cell 4)
Phidget Bridge #476983 (Palm Tool A)	0	0.009	0.023	0.11	-0.057
	10	0.56	0.58	0.66	0.50
	20	1.12	1.14	1.22	1.06
	30	1.68	1.70	1.76	1.62
	40	2.24	2.27	2.33	2.19
	30	1.69	1.72	1.78	1.64
	20	1.13	1.16	1.23	1.08
	10	0.58	0.59	0.68	0.52
	0	0.019	0.034	0.12	-0.047
Phidget Bridge #404917 (Palm Tool B)	0	-0.095	-0.16	-0.14	-0.003
	10	0.46	0.39	0.42	0.56
	20	1.02	0.95	0.98	1.13
	30	1.58	1.51	1.54	1.69
	40	2.14	2.08	2.10	2.27
	30	1.60	1.53	1.55	1.72
	20	1.04	0.97	0.99	1.15
	10	0.48	0.41	0.43	0.59
	0	-0.080	-0.14	-0.13	0.010
Phidget Bridge #410770 (Palm Tool C)	0	-0.17	0.030	-0.015	-0.077
	10	0.39	0.59	0.54	0.48
	20	0.95	1.16	1.10	1.04
	30	1.51	1.72	1.65	1.59
	40	2.08	2.29	2.22	2.15
	30	1.53	1.73	1.67	1.61
	20	0.98	1.17	1.12	1.05
	10	0.41	0.61	0.55	0.50
	0	-0.16	0.041	-0.007	-0.065

Equations (3.1), (3.2), and (3.3) walk one through building a calibration line equation. A load cell force (LCF) is equal to the slope between two known load cell response (LCR)-applied force pairs in units of (lb*V)/mV and multiplied by a measured LCR with the intercept, b , added. The slope, sl , for a model is given by the change in the applied force over the change in a load cell's response. Lastly, the intercept is found by plugging one of the two chosen data points into (3.1) and solving for b .

$$LCF = (sl * LCR) + b \quad (3.1)$$

$$sl = \frac{\Delta AF}{\Delta LCR} \quad (3.2)$$

$$b = AF - (sl * LCR) \quad (3.3)$$

If one wanted to find a line for converting mV/V signals from load cell C2 in Table 3.1 to lb forces over the rated load, the following steps would yield a model:

1. Select data points for cell C2: (0.030 mV/V, 0 lb) and (2.29 mV/V, 40 lb)
2. $sl = \frac{\Delta AF}{\Delta LCR} = \frac{40-0}{2.29-0.030} = \frac{40}{2.26} \approx 17.70 \frac{lb*V}{mV}$
3. $b = AF - (sl * LCR) = 0 - \left(17.70 \frac{lb*V}{mV} * 0.03 \frac{mV}{V}\right) \approx 0.53 lb$
4. Plug sl and b into Equation (3.1): $LCF = 17.7 \frac{lb*V}{mV} * LCR + 0.53 lb$
5. Model complete. If C2 reads 1.4 mV/V, then $LCF = 17.7 * 1.4 + 0.53 \approx 25.31 lb$

This example is considered a two point calibration model. More complicated models can be constructed by using more data points over the 0-40 lb range such as a three point, four point, or even a full five point calibration. The more points that are used, the more line equations are necessary. A five point calibration requires four lines over the four unique force intervals. The models for load cells then become piecewise functions when more than two points are used. The CBSS GUI software discussed in Chapter 4 employs a simple two point calibration for displaying force data. This is because it is presentation software for WVRTC purposes only. A complicated model was not required to showcase CBSS concepts and operations. However, the analysis of the test results in Chapter 5 utilizes a five point calibration model for accuracy.

References to Table 3.1 will be made in these chapters as the models employed are further discussed. Additionally, Table 3.1 contains data for unloading the force from the load cells as well. These can be used to see the effects of hysteresis pertaining to loading versus unloading on the cells.

3.1.3 RSS Design

The RSS fixtures that housed the load cells were designed in SolidWorks 2016-17. The student's past experience with robotics leveraged SolidWorks to create custom parts for 3D printing. RSS fixtures were printed on a uPrint SE Plus 3D printer by Stratasys. This printer, shown in Figure 3.5, allows the user to build parts within a 203 x 203 x 152 mm (8 x 8 x 6 in) volume with a layer thickness of 0.254 mm (.010 in) or .330 mm (.013 in) [31]. With a resolution of 0.254 mm, parts could be fabricated with precision.

Figure 3.5: 3D Printer [31]

The design of the new RSS, shown in Figure 3.6, features two beam cells in the Omega mounting kits. The load cell assemblies are inserted in the bottom housing and covered by a top contact rail. All the main parts are held together with four flathead machine screws.

Figure 3.6: RSS Exploded View

Figure 3.7 shows the insides of the RSS after full assembly. It can be seen that the flathead screws hold everything in place and are countersunk in the top rail so they do not interfere when interacting with a Marman ring. The top rail is “floating” and the gap between it and the bottom load cell housing is 0.51 mm. A cantilevered arm on the back side allows the load cell housing to be screwed to a side of a palm tool. The large round opening is for mounting to the pivot rod that holds the jaw in place. A longer pivot rod, spacer bearing, and a clip would hold the RSS. Since the jaw was not used, the RSS was fastened to a palm’s side behind the round opening.

Figure 3.7: Internals of RSS Assembly

Figure 3.8 shows the exact placement of the two RSS attachments on a single palm tool. The cantilevered arm mentioned before not only provides an attachment point but also holds an RSS in an orthogonal position with respect to the back of the palm. The red box's corners represent the center locations of the four load cells on the tool. Resultant forces from contact should ideally reside somewhere within this boundary. This box also represents where the Marman ring should be resting when the capture drive motors are engaged.

Figure 3.8: RSS Placement on CBS Palm Tool

It should also be noted that the top rail of any given RSS rests just above the profile of the palm surface to establish elevated contact points (see Figure 3.9). The distance above the palm is one millimeter and should still allow a ring to be captured in the jaws as the palm tools retract inward during the capture phase. It is believed the rail height will not affect that process, but testing of ring clampdown would need performed for verification. This, however, was beyond the scope of this research. There were no capture drive mechanisms in place for testing clampdown. The palm tool jaws were printed, but they were not used.

Figure 3.9: RSS Height above Palm Surface

3.1.4 The FSS and Fundamental Concepts

Figure 3.10 gives the spatial layout of the FSS on the CBSS. Palm tools A, B, and C are held 120° apart from one another through the CBSS arms and are labeled in the diagram. Also, the names for every RSS are shown along with the load cells that are embedded in them. This relationship is shown with a curly bracket. RSS names are on the outsides of the brackets, and the two load cells they contain are on the inside. Load cells are still referenced as A1-A4, B1-B4, and C1-C4.

Figure 3.10: FSS Naming Convention on the CBSS

The overall idea of the FSS is to analyze the individual forces on each load cell embedded in the RSS attachments and provide useful state information. The FSS works to determine the magnitudes and locations of the resultant force on each palm tool and the entire (global) three palm tool CBSS system. Cartesian force locations are with respect to the origin at the centroid of the tool. Figure 3.11 shows the forces relative to the CBSS as well as the Cartesian axes at the centroid. The 12 individual load cell forces are labeled near the RSS attachments and are at known x-y locations. The palm tool resultant forces (F_A , F_B , and F_C) are the forces on palm tools A, B, and C, respectively. These forces are local to each palm tool, so they are called the CBSS local forces. They are indicated by the red dots in the red boxes where all local forces should theoretically reside. Each local force is found by analyzing the four load cells on each palm. The resultant force of the entire system, labeled F_G (designated by the light blue dot), is called the CBSS global force. This force is found by using all 12 CBSS load cell forces. All global forces should theoretically reside within the light blue boundary.

Figure 3.11: CBSS Forces

One last piece of necessary information is the physical length from the center of the CBSS to a palm tool. Figure 3.12 shows the distance from the CBSS center to the midpoints of two RSS attachments. This distance is necessary to interact with the Marman ring on the test equipment in Section 4.4.1.

Figure 3.12: Length to Palm Tool

With the physical dimensions given by Figure 3.8 and Figure 3.12, the x-y locations of the load cell forces with respect to the origin in Figure 3.11 can be found. Geometrical concepts and trigonometric functions can be used to find the x-y coordinates of every load cell, which will be important for calculating local and global force information.

The CBSS load cells sense z-axis forces. The load cells cannot determine the angular position of a force vector, and are only capable of providing force magnitudes that are assumed to be orthogonal to an RSS rail. Hibbeler calls a system such as the CBSS a parallel force system because the forces are all parallel to the z-axis [32]. In a parallel force system, the sum of the individual z-axis forces is the magnitude of the resultant force, and the individual forces multiplied by their x-y locations yield torque (moments) about the axes of a given coordinate system [32]. The x-y locations of the resultant force can be found by dividing the sum of the torques by the magnitude of the resultant force [32].

Equations (3.4), (3.5), and (3.6) calculate the magnitude and location of the local force on palm tool A with values from the load cells. Refer to Figure 3.11 to see the pertinent variables. Local force information on palm tools B and C are calculated in the same manner using the load cell forces and locations for those palms.

$$F_A = \sum_{i=1}^4 F_{Ai} \quad (3.4)$$

$$x_A = \frac{\sum_{i=1}^4 F_{Ai} * x_{Ai}}{F_A} \quad (3.5)$$

$$y_A = \frac{\sum_{i=1}^4 F_{Ai} * y_{Ai}}{F_A} \quad (3.6)$$

Lastly, Equations (3.7), (3.8), and (3.9) calculate the magnitude and location of the global CBSS force by using all 12 load cells. Again, refer to Figure 3.11 to see the relevant variables.

$$F_G = \sum_{i=1}^4 (F_{Ai} + F_{Bi} + F_{Ci}) \quad (3.7)$$

$$x_G = \frac{\sum_{i=1}^4 [(F_{Ai} * x_{Ai}) + (F_{Bi} * x_{Bi}) + (F_{Ci} * x_{Ci})]}{F_G} \quad (3.8)$$

$$y_G = \frac{\sum_{i=1}^4 [(F_{Ai} * y_{Ai}) + (F_{Bi} * y_{Bi}) + (F_{Ci} * y_{Ci})]}{F_G} \quad (3.9)$$

These equations govern CBSS operation regarding force sensing. The equations in this section are employed by the software discussed in Chapter 4, with emphasis on the global equations. Magnitude equations are used to tell when palm tool forces are over certain thresholds to determine if contact is made, and the global equations give the overall state of the CBSS. Global equations were important for testing to compare CBSS data to truth data generated by a force, torque sensor featured in the experimental setup. Since torque is output by the sensor, it is important to be able to calculate CBSS x and y torque as well. The global x-axis torque, T_X , and the global y-axis torque, T_Y , are given by Equations (3.10) and (3.11).

$$T_X = F_G * x_G \quad (3.10)$$

$$T_Y = F_G * y_G \quad (3.11)$$

3.2 Design of the PSS

With the FSS design solidified, it was desired to provide another sensory system to determine palm tool proximity to a Marman ring. The only real estate remaining on the palm tools for embedding sensors was the cavities shown in Figure 3.13. The cross sectional area of a cavity measures 1.52 cm x 3.36 cm with a depth of 1.22 cm at the end nearest the black jaw that curves downward to 0.59 cm at the front of the palm tool. Range sensors were needed that fit within this volume constraint that did not protrude above the profile of the palm. One range sensor per palm for a total of three sensors across the CBSS and external sampling hardware make up the PSS.

Figure 3.13: Palm Tool Cavities

3.2.1 Initial PSS Hardware Setup

In summary, three satellite boards (seen in Figure 2.12) plug into the expansion board in Figure 2.14 and together plug into the Nucleo L053R8 board in Figure 2.9. The satellite and expansion boards provide electronics for a stable sensor setup (like in Figure 2.11) without the need for additional electrical components. The only thing not accounted for is I2C pullup resistors that should be installed on the Nucleo. This issue and resolution will be touched upon later. Most of the satellite board signals including AVDD, GND, SCL, and SDA come directly from the Nucleo. When referencing Figure 2.10, those satellite signals are provided by the +3v3, GND, I2C1 SCL, and I2C1 SDA pins on the Nucleo board. The INT (interrupt) pins on each satellite board are not used. Finally, the GPIO pins on the satellite boards come from the Nucleo's GPIO expander in Figure 2.16. The GPIO expander is an I2C device just like the range

sensors, so it must be controlled by the master Nucleo device as well. The expander is located at the address 0x42 (0x84/0x85 in 7-bit addressing), and the sensors all boot up by default with address 0x29 (0x52/0x53 in 7-bit addressing). This concatenation of boards, shown in Figure 3.14, is the foundation of the PSS hardware. Some minor modifications were made as the system was applied to the CBSS, and those will be discussed as the PSS is further presented.

Figure 3.14: Foundation of PSS Hardware

3.2.2 Nucleo Code for PSS

With the hardware setup in Figure 3.14, the embedded code was developed for the PSS before integrating the sensors into the palm tools. However, looking forward to CBSS integration, the embedded code was written with the intent of using the left, bottom, and right sensors (indicated by Figure 3.14) for palm tools A, B, and C respectively. On a high level, Figure 3.15 shows the Nucleo code behind the PSS system.

Figure 3.15: PSS Nucleo Code

The first part of the embedded code defines the 400 kHz I2C frequency and creates the variables “range_l”, “range_b”, and “range_r” for the left, bottom, and right range sensors. The code then interfaces with the GPIO expander in Figure 2.16 to turn off each sensor. They are turned back on one at a time so that the default I2C slave addresses of 0x29 can be changed to unique values. This must happen because all of the sensors boot up to 0x29. If the addresses were left the same, all three slave devices would attempt to perform the Nucleo’s requests at the same time.

The next steps are to initialize/prepare the chips with the manufacturer’s recommended settings found at [33]. The code then clears any range interrupts in the sensors’ MCU registers to ensure they are all ready to begin ranging. Each sensor is then polled sequentially for a range, and the code waits until range finished interrupt signals on the MCUs are set. After the range measurements are completed, the range variables (“range_l”, “range_b”, and “range_c”) are updated with the new range values (in mm). Those ranges are then relayed over USB for another application program to read. The software described in Chapter 4 monitors the Nucleo serial port for range data.

This code was heavily borrowed from STMicroelectronic’s application note on basic ranging with the VL6180X sensors in Mbed at [33]. This application note provided the foundation for performing range measurements on a single sensor but did not offer a solution for using several sensors. The student researched device registers in the data sheets for the GPIO expander and the VL6180X sensor to enable such functionality. The full embedded code is provided in 0 and shows the specific values that were written to all slave device registers to enable a multiple sensor ranging application. For more information about specific device registers of the VL6180X and the GPIO expander, refer to [13] and [22].

3.2.3 VL6180X Placement in Palm Tools and on CBSS

With the embedded code completed, the sensors were then inserted into the palm tools. A simple snap-in fixture was designed for the palm tools that leveraged the open front cavities. The snap-in fixture, shown in Figure 3.16, included a 2x3 female socket and a VL6180X satellite board. The socket was press fit into the snap in fixture, allowing the sensor board to plug in through the socket.

Figure 3.16: PSS Snap-In Fixture and Components

In the same manner as the socket, the snap-in fixtures were press fit into the palm tools. Figure 3.17 shows an example of this. The picture also depicts the recessed distance from the RSS rail plane to the face of the VL6180X sensor. This distance is 8.74 mm, which serves as the offset to zero the sensors when a Marman ring evenly contacts the palm tools. The setup depicted is the complete sensory suite equipped to a single palm tool.

Figure 3.17: VL6180X Placement in Palm Tool and Recess

Figure 3.18 shows the spatial layout of the CBSS with the equipped proximity sensors. Three proximity sensors, the Nucleo development board, and stacked plug-in VL6180X expansion board make up the PSS subsystem of the CBSS. To further add to the naming convention established in Figure 3.10, the proximity sensors on palm tools A, B, and C will be referred to as proximity sensors A, B, and C. Proximity, range, and distance are all used interchangeably in this document. P_A , P_B , and P_C are used to define the measurements from each proximity sensor in mm. Not shown in the picture, is the wiring that connects the proximity sensors to the Nucleo. Each connecting wire from a VL6180X satellite board to the Nucleo was approximately one meter in length.

Figure 3.18: PSS Naming Convention on the CBSS

3.2.4 Sensor Issues, Fix, and Schematic

As soon as the I2C sensors were connected with long wires, they started freezing at random. Further analysis was needed with an oscilloscope to see what the issue was. The scoped clock and data signals of the Nucleo are shown in Figure 3.19. The top image shows the I2C clock and the bottom is the data signal. The clock looked okay except for some “ringing” on the line, but the data signal was not showing clearly defined voltage pulses. The application in Figure 2.11 did recommend that I2C pull-up resistors be added. In this case, none were present except for the MCU’s internal pull-up of 45 k Ω [34]. Stronger pull-ups (lower valued resistors) were needed to correct the signals.

Figure 3.19: I2C Clock and Data Lines without Pull-up Resistors

Referencing Figure 2.10, two $2.2\text{ k}\Omega$ pull-up resistors were added from +3v3 to I2C1 SCL and I2C1 SDA of the Nucleo board. Figure 3.20 shows the results. The clock line looks about the same, but the data line reveals clearly defined voltage pulses, allowing the sensors to return to normal operation.

Figure 3.20: I2C Clock and Data Lines with Pull-up Resistors

Figure 3.21 shows the final wiring diagram with the I2C issues fixed. The VL6180X expansion board gets plugged into the Nucleo. The left, bottom, and right expansion board connectors go to the proximity sensor satellite boards on palm tools A, B, and C through approximately one meter of cabling for each sensor. The wiring of this cabling is one to one with the sensors. The expansion board has pin 1 marked on all sensor connectors, which corresponds to the square pad INT pins on the satellite boards (Figure 2.12). Furthermore, the added resistors intended to stabilize the I2C signals that were degraded from cable capacitance are shown on the appropriate Nucleo terminals. Lastly, the Nucleo is wired via USB to the CBSS hub and connected to a computer running the control software.

Figure 3.21: Final PSS Wiring Diagram

3.2.5 PSS Fundamental Concepts

The idea behind the PSS is to allow robot operators, or even autonomous algorithms, to know the distance between every palm tool and an inbound object (the Marman ring). The PSS utilizes one proximity sensor per palm to report these distances. The three palm CBSS can use this information for pose estimation and calculating rotational angles about certain axes to correct alignment with a Marman ring. When the CBSS was tested, it was physically moved into contact with a stationary Marman ring so correction angles were calculated about the CBSS x-y axes. The assumptions made were that proximity measurements represent the distance to a ring from the exact center of a palm. The sensors were less than 2.23 cm (0.9 in) from center in any Cartesian direction, so this compensation was left out of the equations for simplicity.

For a y-axis correction, shown in Figure 3.22, P_B and P_C are the proximity measurements of interest. P_A is not required because the CBSS y-axis runs into the page at the origin of the x-z set of axes. Rotation about the y-axis will not change P_A but equalize P_B and P_C . For calculating θ_Y , the distance between two palm tool centers must be known. This can be found using dimension in Figure 3.12 and the fact that CBSS arms are 120° apart. Let that distance between palms be denoted by D . Under a new x-y coordinate system, indicated by $x'-y'$ in the figure, vectors $v1$ and $v2$ are given by: $v1 = \langle D, P_B \rangle - \langle 0, P_C \rangle = \langle D, P_B - P_C \rangle$ and $v2 = \langle D, 0 \rangle$.

Figure 3.22: Calculating Y-Axis Pose Correction

Taking the dot product of the two vectors and solving for θ_Y yields the following equation:

$$\theta_Y = \cos^{-1} \left(\frac{D}{\sqrt{D^2 + (P_B - P_C)^2}} \right) \quad (3.12)$$

Calculating the CBSS x-axis correction angle, shown in Figure 3.23, is more complex and requires all three proximity sensor measurements. The x-axis points out of the page at the origin of the y-z set of axes. First, the average of P_B and P_C is evaluated to represent the distance halfway between proximity sensors B and C to the Marman ring plane. The distance between a line through sensors B and C to sensor A (depicted as vector $v1$) is not the same distance as D in the last problem. Trigonometry can be employed to reveal that the distance is $D \sin(60^\circ)$. Referencing the x' - y' coordinate system, the vectors of interest become: $v1 = \langle D \sin 60^\circ, 0 \rangle$ and $v2 = \langle D \sin 60^\circ, P_A \rangle - \langle 0, \frac{P_B + P_C}{2} \rangle = \langle D \sin 60^\circ, \frac{2P_A - P_B - P_C}{2} \rangle$.

Figure 3.23: Calculating X-Axis Pose Correction

Taking the dot product of the two vectors and solving for θ_X yields the following equation:

$$\theta_X = \cos^{-1} \left(\frac{D \sin 60^\circ}{\sqrt{(D \sin 60^\circ)^2 + \left(\frac{2P_A - P_B - P_C}{2} \right)^2}} \right) \quad (3.13)$$

Equations (3.12) and (3.13) define the two rotations about the CBSS axes to align the contact plane of the three palms with a Marman ring. These equations are employed by the CBSS GUI in Chapter 4 to relay this information to a remote user. The only value needing resolved is D , which can be found with CAD or trigonometric methods. Given previous dimensions, D was 104.1 cm (40.98 in) in the CBSS prototype.

3.3 CBSS Hardware Setup

Figure 3.24 shows the actual CBSS prototype that was tested in this research. The student designed and printed the green RSS attachments as well as the proximity sensor snap-in fixtures. The Nucleo is mounted at the center of the CBSS, and the sensor cabling can be seen going to the palm tools. It is difficult to see, but the Phidgets 1046 devices that the load cells are connected to are all mounted directly below the palms close to the RSS attachments. The white USB cables are routed from the Phidgets 1046 devices to a USB hub. The Nucleo is also plugged into the same hub. Although the picture does not show this hub, it is inside the black structure supporting the center of the CBSS. Lastly, the USB hub connects to a computer running CBSS code for telemetry. Figure 3.2 and Figure 3.21 provide the wiring diagrams for both the FSS and PSS hardware setups.

Figure 3.24: Complete CBSS Hardware Setup

CHAPTER 4 | CBSS SOFTWARE TOOLS AND EXPERIMENTAL OVERVIEW

This chapter provides an overview of the developed CBSS software and relevant WVRTC robotic lab equipment. The lab equipment is briefly mentioned to provide insight into test setups and specifications for metrology devices such as accuracy, repeatability, etc. Additionally, procedures and test plans that experimentally validate the operations of the CBSS prototype are discussed.

4.1 CBSS Software Overview

Two software tools were developed for the CBSS prototype: a user friendly presentation GUI for monitoring purposes and a lightweight logger in C++ for experimental data collection. The GUI was created in Visual Studio 2015 with the C# programming language and WPF on a Windows 7 OS. C# is a type safe object oriented language targeted for the .NET Framework that can be used to create a variety of Windows applications [35]. The .NET Framework is made up of a common language runtime (CLR) and the .NET Framework library [36]. The CLR enforces language rules and manages code at execution time by providing services such as compilation, memory management, garbage collection, thread management, and other critical services [36]. The .NET class library is language independent and provides ways to perform common programming tasks such as string management, file input/output (IO), reading/writing to serial ports, and much more for any .NET programming language [36]. Lastly, WPF is a rendering engine and a subset of the .NET library that allows developers to create custom GUI applications to simplify user-software interaction [37]. WPF can feature 2D or 3D animation; forms that gather user input through drop down lists, text boxes, or buttons; and much more [37]. More information on WPF, the .NET library, and C# can be found at [37] [38] [39] [40] [41].

The C++ logger was developed on Ubuntu 16.04.1 OS with a standard text editor and the GCC 5.4.0 compiler. Ubuntu is an open source Linux OS maintained by a worldwide network of voluntary developers [42]. The GCC compiler is standardly packaged with the Ubuntu OS and provides compilers for common languages such as C, C++, Objective-C, Fortran, Ada, and Go [43]. The version of C++ that was flagged upon compilation was C++11 to enable those specific library and language features. Regarding code development, the default gedit GUI text editor packaged with Ubuntu was employed.

Both CBSS software tools were multithreaded by design because of the associated benefits of the programming paradigm. Multithreading allows for concurrent execution within a program if a computer's hardware allows for it [44]. Multithreading and concurrency on a system with a single computing core means that all thread execution is interleaved on that single core whereas multiple threads can execute on multiple cores if they are present [44]. The benefits of threading include increased responsiveness to a user because of parallel execution and easily shared resources among threads such as memory without programmers having to specifically implement sharing like in multiple process programming [44]. Additionally, the overhead a processor experiences when managing multiple threads is generally less than managing multiple processes [44]. This makes multithreading a popular choice among programmers. Since threads share memory, some caution should be taken to avoid overwriting variables that may be needed by other threads. Both CBSS software tools were designed so that variables in shared memory were not corrupted between threads.

4.2 CBSS GUI

4.2.1 Dependencies and Assumptions

There were three external libraries the GUI application required for one to properly compile and run the code on the Visual Studio Integrated Development Environment (IDE). The dependencies of the code were the Phidget21 libraries, OxyPlot.Core, and OxyPlot.WPF. The Phidget21 libraries allow developers to easily interact with any Phidgets product and can be downloaded at [30]. These libraries were used to get input from FSS load cells on the PhidgetBridge 4-Input boards. The OxyPlot libraries are under the Massachusetts Institute of

Technology (MIT) software license, but are open source .NET libraries that allow users to plot functions on a variety of platforms including WPF [45]. These libraries were used to plot range measurements over time in the GUI and were downloaded from within the Visual Studio IDE.

One assumption the GUI made was that the Nucleo board was plugged in via USB, flashed with the embedded code in 0, and outputting range sensor data over a specific virtual serial communications port. For the Nucleo to be recognized in Windows, drivers had to be downloaded at [46]. STSW-LINK009 was the USB driver, and STSW-LINK007 provided a firmware upgrade. The Nucleo board had to remain plugged in because the GUI was not programmed to recover from attaching or removing the device at runtime. The GUI was not user friendly regarding the Nucleo's serial port. No tools were implemented to allow users to find and connect to virtual serial communication ports during runtime. The board's port number was hard coded in the GUI and had to be changed within the code prior to execution to match the port Windows assigned the device. Another method used for matching the port involved using the Windows device manager to change the port number of the Nucleo board to match the GUI code. Unlike the Nucleo board, the PhidgetBridge boards could be plugged in and removed repeatedly via USB. The GUI, along with the Phidgets API, handled those events seamlessly and recovered to continue operation.

4.2.2 Operating the GUI

Upon running the GUI in Visual Studio or with the executable, one would see the interface shown in Figure 4.1. The radial grid graphic on the left is the main hub of the GUI and represents the contact zone of the CBSS. This contact zone graphic encompasses all of the CBSS load cell locations and is used for animating the approximate location of the global force when contact is made. The angular lines in the radial grid originate at the centroid of the CBSS and run to the edges of the contact zone. A line is provided for every angle from 0° to $\pm 180^\circ$, in 20° increments, and dotted lines represent the x-y Cartesian axes. Concentric circles intersect the angular lines to form sectors that are illuminated when the global force resides within their boundaries.

Figure 4.1: CBSS Software GUI

The colored polygon southeast of center in Figure 4.1 represents the global force at the time the image was captured. A robot operator or autonomous algorithm could obviously see that contact has been established on all three palms and that more force is on palm B. Based on this knowledge, robotic manipulators could articulate a captured satellite very slowly along a line of action through the origin that is parallel to a line linking palms A and C. The global force would then become centralized on the CBSS as the forces on palms A and C increase. A robot operator or autonomous algorithm could then cease articulation when the global force is within a specified tolerance of center thereby bringing the CBSS into even contact with a captured satellite. Maintaining even contact through force telemetry is a feature granted through inclusion of a system such as the FSS.

Force location colors in Figure 4.1 were chosen randomly and indicate one unit of spacing further away from the center of the CBSS. For example, the colored polygon is yellow-green, and forces residing in sectors between the next two concentric circles may be orange. The readouts to the right of the graphic under “Force Data” are used to display precise global force information to the user. Polar and Cartesian coordinates are shown in inches and degrees as well as the total force magnitude. A button and a dropdown list under those readouts allow the user to tare all the load cells at once and to display all force information in lb, kg, or N units.

In each of the three corners of the triangular shaped graphic, two gray rectangles are shown representing the RSS rails on a single palm tool. Each gray rectangle has two circles representing the two load cells in each RSS. Figure 4.2 shows palm tool A. Load cell circles are animated with color gradients from green to dark red and represent forces from zero to 177.93 N (40 lb), which was the capacity of the FSS load cells.

Figure 4.2: RSS Rails in GUI and Sensor Readouts

Text boxes under “Physical Measurements (in)” in the GUI give the dimensions of the CBSS prototype and are shown in Figure 4.3. The “CBSS Arm Length” is 60.1 cm (23.66 in) by default and represents the orthogonal distance from the center of the CBSS to a line connecting the midpoints of the two RSS rails on a palm. The default values for the center to center distance between two load cell circles on any given rail and the separation between two rails on any palm tool are 4.14 cm (1.63 in) and 17.48 cm (6.88 in) respectively. They are labelled “RSS Depth” and “RSS Width”. Also, the center circle in the radial grid represents a zone with a 10.16 cm (4 in) diameter. If nominal contact occurs, the location of the global force is at the centroid of the CBSS and the inner circle is colored green. With the default physical tool dimensions, the calculated spacing between concentric circles in the radial grid is 11 cm (4.33 in). The physical dimensions of the CBSS can be altered by the user should geometric locations change over time due to test setups.

Figure 4.3: CBSS Dimensions in GUI

Figure 4.4 provides a zoomed in view of the radial grid force graphic. The “Cell” readouts surrounding the CBSS contact rails report the forces on each load cell and correspond to the animated circles that are closest to them. “Prox” readouts are used to display CBSS proximity data, representing each palm’s distance to an inbound object. They display “No Data” if an incoming object is not present or if it is simply outside a palm sensor’s operational range. Proximity sensor readouts have tare buttons beneath them to allow the user to zero the sensors at contact. A “Tare All” button also exists under “Proximity Functions” on the left side. Additionally, a dropdown list is featured there for displaying proximity data in mm; cm; or in. units. The remaining readouts are used to calculate axis rotation angles base on proximity data to nominally align an inbound object with the CBSS. They represent angles that the CBSS must be rotated about its x- and y-axes for a nominal approach to a Marman ring. The angles are defined with respect to turning the CBSS because the CBSS was in motion during testing rather than the Marman ring. The arrows on the x- and y-axes near the correction angle readouts define the positive direction of rotation. At the time the image was captured, the CBSS had to be rotated 0.3° on its x-axis and -0.2° degrees on its y-axis based on palm tool A, B, and C’s proximity measurements of 0.0, 1.0, and 0.0 mm. The tool was in full contact with the ring but the global force showed that contact was not exactly even across all palm tools.

Figure 4.4: Close Up of GUI Force Graphic

The upper right quadrant of the GUI, shown in Figure 4.5, is for plotting the proximity sensor measurements from each palm. The OxyPlot libraries enable the GUI to perform this. The user can set the plot refresh rate by typing a value in ms and clicking the “Set Refresh Rate (ms)” button. The refresh rate is set to update the plot every 250 ms by default. The user can also set how many seconds of history is shown in the plot with the “Set Window (s)” button. The time window must be in seconds, and is 20 by default. The near real time data is continuously plotted on the right and history data is shifted left as new measurements are received.

Figure 4.5: Proximity Plotting in the GUI

PhidgetBridge board information is given under the main CBSS force graphic and is shown close up in Figure 4.6. This part of the GUI tells users if a palm tool’s PhidgetBridge board is connected and displays the number of active sensors on each board. A data rate slider allows the user to adjust the refresh rate of the sampling boards at runtime. By default, the PhidgetBridge boards are set to take force samples every 192 ms. The user can quickly change this to make the load cells more or less responsive.

Figure 4.6: PhidgetBridge Properties in the GUI

Data logging functionality is provided to the user in the bottom right quadrant of the GUI and is shown in Figure 4.7. The user can click on “Browse” to navigate the OS to select or create a directory for storing data log text files. If the user does not select or create a folder, the GUI creates a “LOGS” folder in the directory where the code is stored. Before initiating a data log with the “Log Data” button, users can set the sampling rate. By default, the GUI logs raw CBSS sensor data every 200 ms unless the user modifies this text to a custom value. New sampling rates are saved in the code when “Log Data” is pressed. Every log file generated will have a timestamp so stored logs are not overwritten.

Figure 4.7: GUI Logging

4.2.3 Software Flowcharts

As stated previously, the underlying CBSS GUI code was multithreaded. The WPF threading model is designed so developers do not necessarily have to develop code with multiple threads in mind. A background thread performs the majority of the work, and a separate thread checks an object known as Dispatcher for tasks that involve updating and rendering a GUI [47]. This allows GUI rendering to be separated from the background code to increase GUI responsiveness to a user. Most WPF classes are derived from DispatcherObject, which means they contain a reference to the Dispatcher [47]. This allows developers to queue GUI rendering tasks to Dispatcher for execution [47].

The background and GUI threads for the CBSS software are shown in Figure 4.8. The background thread initiates global variables, populates the GUI with default values, attaches events to the PhidgetBridge boards and the Nucleo serial port, opens hardware devices, and waits for code events to fire. As events fire, they are processed by handler functions. Any part of the code in the flowchart that requires the GUI to be rendered, which is the initial GUI population and the processing of incoming sensor data events, executes code on the Dispatcher. The specific flowchart steps in the background thread that require GUI rendering indicate this with “(add to

dispatcher)”. The GUI thread continuously waits for rendering tasks on the Dispatcher, executes them as they become available, and makes changes in the GUI accordingly at the request of the background thread. When the user finally exits, objects are torn down, events are stopped, memory is freed, and logs are closed.

Figure 4.8: Overview of GUI Code

The most important events that occur in the GUI are sensor data events and log file creation. The flowcharts for such event handlers are shown in Figure 4.9 – Figure 4.11. Figure 4.9 depicts the handler for processing load cell input from a PhidgetBridge board on a specific palm tool. Upon being called, the code determines what input was updated on the sampling board. Bridge inputs 0-3 correspond to load cells 1-4. Depending on which palm tool the board is on, that designates the letter prefix of the load cell. For instance, if palm tool B's PhidgetBridge device was sampled from, load cells 1-4 in the flowchart correspond to CBSS load cells B1-B4 to follow naming conventions. The load cell's mV/V measurement on a specific bridge input is then converted to a force based on each load cell's calibration line and saved to a specific global force variable in memory. The GUI employs a simple two point calibration using load cell data at no load (0 lb/N) and full load (40 lb/177.93 N) from Table 3.1 to yield 12 CBSS load cell lines that convert mV/V signals to forces. A more complicated five point calibration is explored later in Section 0.

Continuing on with the flowchart in Figure 4.9, the force variable and readout for the load cell that has been sampled from is updated in the GUI. The readout is updated with the new value minus a tare if one is present. The circles that represent them are rendered with a color gradient algorithm next. Both tasks are added to the Dispatcher so the GUI thread executes them concurrently. After those steps have been performed for the specific load that triggered the event, the global force information (magnitude and location) is updated based on the state of all individual force variables in the GUI's memory. The global force readouts are then updated and the radial grid force graphic is rendered. Once again both tasks are added to the Dispatcher so the event handler can exit and reestablish sequential execution.

Figure 4.9: Load Cell Data Event Handler

Figure 4.10 shows the handler for processing proximity sensor input from the Nucleo board. Upon being invoked, the handler checks if three distance values have been received through the port. Sometimes, upon initiation of the port, an incorrect number of sensor values are received and the handler exits immediately. Otherwise, the serial data is parsed and each GUI range variable is updated. If any sensor value is 255, the GUI proximity readouts are updated to show “No Data”. If the sensors show any other value, the readouts are updated with those values minus tares. These updates are added to the dispatcher and then pose correction angles are calculated. Proximity data must be present for sensors B and C to calculate a y-axis rotation, but the x-axis rotation angle cannot be resolved if a single sensor shows 255/“No Data”.

Figure 4.10: Nucleo Serial Port Data Event Handler

Figure 4.11 shows the event handler that logs CBSS data when an automatically resetting periodic timer expires. To begin logging, a user must press the “Log Data” button in the GUI (left flowchart). At this point in time, the background thread attaches an event to that periodic logging timer. Events are generated when the timer expires, which is at the desired sampling rate specified by the user in the GUI. A text file is then created with a timestamp and opened for writing. Additionally, a stopwatch is started to show the time elapsed when data is collected. This all occurs before any log event is raised.

When the actual handler (middle flowchart) is invoked, it simply writes all force and range GUI variables to the opened text file. A timestamp from the stopwatch, followed by the 12 load cell responses and three proximity values, are written to the opened file. All data is separated by commas when written and constitutes a single line in a log file. The sensor data log contains raw data, meaning no tares or filters are applied.

When a user terminates a log via the “End Log” button (right flowchart), the background thread stops the log timer, detaches the timer firing event, and resets the stopwatch. The opened text file is then closed and saved in the default or user specified directory. This entire process can be repeated at the discretion of the user anytime data must be collected.

Figure 4.11: Logging Event

4.3 CBSS C++ Data Logger

4.3.1 Dependencies and Assumptions

The only libraries this logger required for compilation and execution was the Phidget21 libraries. These libraries allowed the logger code to interface with the PhidgetBridge boards to collect CBSS load cell data and were downloaded and installed at [48]. Exactly like the GUI, this code required the Nucleo board to be plugged in via USB, flashed with the embedded code described in 0, and outputting range sensor data. This could be verified with “dmesg | grep tty” and “sudo cat /dev/tty<insert device name>” in a Ubuntu terminal. The first command shows attached devices and the second shows the output over the chosen device’s serial port. These commands were used to explore the Nucleo board. The embedded code was flashed to the Nucleo on a Windows machine where the USB driver was installed. The Nucleo was then plugged into Ubuntu, and the embedded code executed as intended. Exactly like the GUI, the PhidgetBridge boards could be removed and plugged back in without breaking the logger due to the Phidgets API. The Nucleo board had to remain plugged in though.

The logger’s sole purpose was to sample and log data. The code, provided in APPENDIX B, should have been used to collect CBSS data from every experiment described later in this chapter. However, the GUI logger was used in some cases because it was developed and operational before the C++ code was finished. The C++ logger was a terminal application, did not have any of the overhead associated with rendering graphics, and was very responsive. This logger was not as user friendly as the GUI and assumed the operator had some experience with coding. The code did not take user input, so constants at the beginning of the C++ file had to be edited to match the user’s needs. Constants of interest were PHIDGET_REFRESH_RATE, SAMPLE_TIME_US, and NUCLEO. By default, the Phidgets refresh rate was 192 (ms), the sampling time was 200000 (μ s), and the NUCLEO device string was “/dev/ttyACM0”. The first two constants can be changed if a user wishes, but the terminal commands in the former paragraph should be used to verify the device string of the Nucleo board. If it differs from the string in the code, the user must update NUCLEO to allow the code to sample from the proximity sensors.

4.3.2 *Compilation and Execution*

Upon following Section 4.3.1, installing the Phidget21 libraries, and saving the logger code, a user can compile the code with “g++ -std=c++11 -pthread DataLogger.cpp -o logger -lphidget21”. When Phidgets devices and a Nucleo are attached via USB, the logger must be executed with elevated permissions by using “sudo ./logger”. Logging begins as soon as the program loads and ends at the user’s discretion when any character is input through the keyboard. Generated log files are timestamped and saved in the same directory as the logger code.

4.3.3 *Software Flowchart*

The overview flowchart of the C++ code is shown in Figure 4.12. The logger was a multithreaded application containing three threads: the main thread, the logger thread, and the Nucleo thread. The main thread begins by defining global CBSS force and proximity variables, device constants, the sampling rate, and etc. Handles/pointers to the three CBSS PhidgetBridge boards are created, and events are attached to them. The boards are then opened, after which the logger and Nucleo threads are spawned. The Nucleo thread opens up the serial port and constantly reads it for data. As data becomes available, the proximity sensor variables are updated. The logger thread opens a file for writing, records a sample timestamp, and then records the global CBSS sensor variables every 200 ms. The main thread waits for user input and handles PhidgetBridge board events. When load cell data is available, the CBSS force sensor variables are updated. When a user strikes a keyboard character, the main thread sets Boolean variables “serialDone” and “logOver” to “true”. The other threads monitor these variables to know when to stop execution. They then terminate log files, close the Nucleo port, clean up, and exit when those state variables change. In the main thread, sequential execution is reestablished at the close of the other threads. The Phidgets board objects are then deleted, and the entire program exits.

Figure 4.12: C++ Logging Code Overview

4.4 Testing Overview

The test campaigns described later in this chapter were designed to show proof-of-concept of the prototype CBSS. These tests involved measuring the force that was applied/received on the CBSS in various contact configurations with a Marman ring. Additionally, measuring proximity of the CBSS to the ring in a nominal alignment scenario was featured. Force measurements from each palm tool were used to reconstruct the magnitude and location of the global force for comparison to truth data generated by WVRTC lab equipment. Proximity measurements were also validated with the equipment.

4.4.1 WVRTC Lab Equipment Used

The WVRTC Fairmont lab has a multitude of robotic equipment and metrology devices that were utilized for CBSS testing and validation. One robotic arm present is the Motoman-SIA50D. This industrial arm, shown in Figure 4.13, has seven degrees of freedom (DOF) and can support a 490.3 N (110.3 lb) payload [49]. It has a vertical reach of 2,597 mm (102.2 in), a horizontal reach of 1,630 mm (64.2 in), and a commanded position repeatability of ± 0.1 mm (± 0.004 in) [49]. With the kinematic models employed for this arm in the robot's control software, the accuracy is ± 2 mm (± 0.04 in) [50]. In general, the CBSS prototype was mounted on this arm and moved during testing to change orientation and applied forces to match desired test conditions. The high DOF and adequate reach allowed the CBSS to be manipulated to a variety of configurations effectively. This robotic arm will be referred to as the “Moto 50” henceforth.

Figure 4.13: Motoman-SIA50D [51]

The second robotic arm used was the Motoman-MH250 II (shown in Figure 4.14). This six DOF industrial arm can support a 2451.7 N (551.2 lb) payload, has a vertical reach of 3,490 mm (137.4 in), a horizontal reach of 2,710 mm (106.7 in), and a commanded position repeatability of ± 0.2 mm (± 0.008 in) [52]. Accuracy and repeatability were not specifications of interest for this arm because it remained static during testing. A mock-up section containing the Marman ring of the Geostationary Operational Environmental Satellite-12 (GOES-12) was mounted to this arm and positioned in the location required for testing. The GOES-12 mock-up is what has been referred to in this document as the FSAB, or Full Satellite Aft Bulkhead. Figure 4.15 shows the FSAB connected to the MH250. The Marman ring can be seen and was the target feature the CBSS interacted with. This robotic arm will be referred to as the “Moto 250” hereafter.

Figure 4.14: Motoman-MH250 II [53]

Figure 4.15: Motoman-MH250 II and FSAB [54]

A JR3 160M50A4-180-DF force-torque sensor (FTS), mounted on the distal end of the WVRTC's Moto 50, is where the CBSS was secured. The CBSS bolted directly to the top of the FTS. This sensor can be seen in Figure 4.16 and will be referred to as the “JR3” moving forward. The JR3 employs multiple strain gauges on internal load bearing elements to provide a six DOF sensor, meaning that it can determine both force and torque along the x-, y-, and z-axes [55]. The x-, y-, and z-coordinate system originates at the geometric center of the sensor [55]. The particular model JR3 employed in testing has an x-, y-, and z-axis force range of 1112, 1112, and 2224 N (250, 250, and 500 lb) as well as an x-, y-, and z-axis torque range of 181, 181, and 181 Nm (1600, 1600, 1600 in-lb) [56]. The accuracy of the force and torque measurements is ± 2.8 N (± 0.625 lb) and ± 17.8 Nm (± 4 in-lb), respectively [56]. The JR3 sensor provided a baseline for validating the FSS subsystem's performance.

Figure 4.16: JR3 Force Torque Sensor [55]

The remaining metrology devices featured in the experiments were the Leica Absolute Tracker AT901-MR and the Tracker-Machine control (T-MAC) TMC30-B. These devices will be referred to as the “Leica” and “T-MAC”. The Leica, when paired with the T-MAC, is a six DOF setup that allows the Leica to accurately calculate x-, y-, and z-position, as well as roll, pitch, and yaw orientation, up to a distance of 18 m [57]. The Leica is a laser tracker that can automatically lock onto and follow the T-MAC sensor to provide the aforementioned telemetry. The Leica is also impervious to ambient lighting conditions and tunes out all other light sources except for its own laser [57]. When the T-MAC was necessary (only featured in one experiment), it was mounted on the Moto 50 via an adapter plate fastened to the top of the JR3; the CBSS was then connected to the JR3 on top the adapter plate. The Leica was used to generate six DOF data

as a baseline for validating the PSS subsystem on the CBSS. Additionally, the Leica can lock onto spherical reflectors that can be positioned at custom measurement locations. Both the T-MAC and the reflectors were utilized either in setup stages or actual experimentation.

One noteworthy performance metric is the Leica's Maximum Permissible Error (MPE), which is defined as the extreme values of an error that are permitted for a given instrument [58]. The Leica T-MAC setup has the lowest guaranteed MPE when compared to any laser tracker on the market [58]. Accuracy is typically half of the maximum permissible error, so the Leica's position and orientation accuracies are $\pm 0.2 \mu\text{m} + 0.15 \mu\text{m/m}$ and $\pm 7.5 \mu\text{m} + 3 \mu\text{m/m}$ [58]. The Leica was around five meters away from the Moto 50 in the WVRTC lab, so those accuracies can be expected to be approximately $\pm 1 \mu\text{m}$ and $\pm 23 \mu\text{m}$. The repeatability is not specified, but the product brochure alluded to high repeatable. Figure 4.17 shows a Leica/T-MAC setup. This is from a conceptual video on Hexagon's webpage but captures what Leica tracking is all about [59]. The Leica is circled in the foreground of the image and the T-MAC is an attachment on the wrist of the orange robotic arm (the T-MAC is circled as well). The Leica is tracking this arm's position at 1000 Hz. The data box left of the robotic arm shows x-, y-, and z-position, as well as roll, pitch, and yaw orientation. This method was implemented in CBSS testing but pose data was all that was necessary.

Figure 4.17: Leica Tracking a T-MAC Sensor [59]

4.4.2 General Test Setup

The test setup, featuring most of the WVRTC lab equipment and the CBSS, can be seen in Figure 4.18. The Moto 250 cannot be seen but was supporting the FSAB above the CBSS. The JR3 was mounted to the end of Moto 50, and the CBSS was bolted directly to the JR3. The palm tools were adjusted to be on a common radius referenced from the center of the CBSS, allowing the RSS modules to contact the Marman ring properly. These adjustments yielded the dimension in Figure 3.12. The CBSS palms were then raised or lowered to establish coplanar RSS contact surfaces. The Leica and spherical reflectors were used to verify the common radius and coplanar contact surfaces, as well as note any discrepancies in the setup due to the limitations of the adjustable mechanisms on the CBSS. The Leica, reflectors, and Leica control software were then used to generate a plane connecting the contact surfaces of the three palms tools (the contact plane) and a plane for the Marman ring (the ring plane). Finally, the Moto 50's control software was used to position the CBSS such that the contact and ring planes were parallel but separated by 100 mm. This configuration served as the home position referenced by all tests. The T-MAC was only featured in a single, brief test and does not appear in any pictures due to a lack of footage. The particular setup in Figure 4.18 was used for all tests with the exception of the previously mentioned one, but Section 4.4.1 discusses how the T-MAC was attached.

Figure 4.18: Equipment Locations

4.4.3 Data Management

The CBSS software tools have the functionality to generate sensor logs that include sequential strings of timestamps, 12 load cell responses, and three proximity measurements. This data is generated by the FSS and PSS subsystems. By default, the PhidgetBridge hardware is configured to refresh load cells every 192 ms (5.21 Hz), while the Nucleo is setup to poll range sensors every 100 ms (10 Hz). In the experiments, CBSS data logging tools were set to record data every 200 ms (5 Hz). Data was continuously logged over the duration of any given test. Measurements from metrology lab equipment were sampled every 2 ms (500 Hz) and continuously logged with lab computers. The only exception was the test campaign in Section 4.5. When the experiments were performed, CBSS load cells were configured to refresh at a much quicker rate and CBSS software logging occurred every 100 ms (10 Hz).

4.4.4 Independent Testing Variables

The independent variables that were manipulated in the experiments were the alignment/contact configuration of the CBSS with respect to the FSAB Marman ring, the contact force (F) between the CBSS palm tools and the ring, the angular position (α) between the contact plane and the ring plane, and the z-axis distance (D_z) from the contact plane to the ring plane. Test scenarios where α was non-zero are considered misalignment contact scenarios.

4.5 Nominal Contact FSS Testing

This group of experiments, depicted in Figure 4.19, tested the FSS subsystem of the CBSS in nominal contact with the FSAB Marman ring. It can be seen that the contact plane is parallel to, and in uniform contact with, the Marman ring plane ($\alpha = 0^\circ$, $D_z = 0$). In this scenario, all RSS modules should measure an equivalent force, and the location of the calculated global force should be at the geometric center of the CBSS.

Figure 4.19: Nominal Contact (3 Palm Tools, 6 RSS, $\alpha=0^\circ$)

Each test began from the home position, discussed in Section 4.4.2, and was conducted per Table 4.1 specifications. The Moto 50 was used to translate the CBSS upwards from home into contact with the Marman ring where the table-defined loads were then applied. The contact force (F) was increased from 0 to 250 N, in 50 N increments, and then removed in 50 N increments, returning to 0 N at the home position. Each force condition was maintained by the Moto 50 for approximately 25 s before transitioning to the next condition. Testing these cases exposed the effects of hysteresis through loading and unloading in a sequentially manner. CBSS logging started and ended at 0 N to observe load cell drift from zero (if any) and to remove any sensor biases in data reduction. Table 4.1 was completed three times to analyze repeatability. The JR3 was continuously logged over all three experiments.

Table 4.1: Nominal Contact Test Matrix

Test #	Alignment	Angular Position (α)	Force (F)
X	Nominal Contact	0°	0 N (0.0 lb)
			50 N (11.2 lb)
			100 N (22.5 lb)
			150 N (33.7 lb)
			200 N (45.0 lb)
			250 N (56.2 lb)
			200 N (45.0 lb)
			150 N (33.7 lb)
			100 N (22.5 lb)
			50 N (11.2 lb)
			0 N (0.0 lb)

4.6 Misalignment to Nominal Contact FSS Testing

This campaign tested the FSS in an alignment correction procedure (Figure 4.20 illustrates this). The CBSS was rotated in a manner such that only palm tool A contacted the ring (left picture). This configuration allowed contact on both RSS rails on palm tool A. Four of the 12 load cells contributed to the global force calculations in this scenario. The CBSS was then driven into nominal contact (right picture) after the initial, misaligned contact occurred.

Figure 4.20: Misalignment (1 Palm Tool, 2 RSS, $\alpha=5^\circ$) to Nominal Contact (3 Palm Tools, 6 RSS, $\alpha=0^\circ$)

Starting from home, the Moto 50 was used to move the CBSS into nominal contact with a total force of 180 N. This position was saved as “nominal”. After returning to home, robot motion was performed to tilt the CBSS away from the Marman ring at 5° (a line connecting the two midpoints of palm A’s RSS rails served as the axis of rotation). The experiment was then carried out per Table 4.2 specifications with the Moto 50 maintaining each table condition for approximately 10 s. From the tilted position, the CBSS was moved vertically into contact with the Marman ring until a force (F) of 50 N was achieved. Upon meeting that condition, the CBSS was then steered into nominal contact by commanding the Moto 50 to the nominal position. Logging started and ended at 0 N to observe load cell drift from zero (if any) and to remove sensor biases in data reduction. Table 4.2 was completed 11 times to analyze repeatability, and the JR3 was continuously logged over all 11 experiments.

Table 4.2: Alignment Correction Test Matrix

Test #	Alignment	Angular Position (α)	Force (F)
X	Misaligned Contact (1 Palm Tool, 2 RSS)	5°	0 N (0.0 lb)
			50 N (11.2 lb)
	Nominal Contact	0°	180 N (40.5 lb)
			0 N (0.0 lb)

4.7 Nominal Alignment PSS Testing

Two experiments were performed to test the PSS in a nominal CBSS alignment with the FSAB Marman ring. Proximity sensors were used to track palm tool distances to the ring as the CBSS descended from the FSAB. The CBSS contact and Marman ring planes remained parallel through the descent, implying equal palm distances. One test utilized the Leica/T-MAC setup to provide a baseline for the PSS, while the other used the Moto 50 kinematic models.

4.7.1 Leica Tracking T-MAC

The CBSS was moved into nominal contact with the Marman ring, and D_z was increased from 0 to 190 mm based on Table 4.3. The Leica tracked the T-MAC to indicate when the table distances were achieved. PSS data was manually logged while observing the CBSS GUI. The minimum and maximum values for each proximity sensor at each table distance were noted. With Leica accuracy at $\pm 1 \mu\text{m}$, the positional values of the Moto 50, and the corresponding distances of travel, were practically the ideal values and not logged. Previous experimenting with the proximity sensors showed that they stop outputting data around 200 mm, which is why the distances in Table 4.3 were chosen. Only one test was performed, and it should be noted that the sensor recess in the palm (8.64 mm) was added to the table distances for the results. This testing exposed the accuracy of the VL6180X sensors; the manufacturer did not make it very clear.

Table 4.3: Proximity Characterization Test Matrix with Leica & T-MAC

Test #	Alignment	Angular Position (α)	z-Distance (D_z)
1	Nominal Alignment (start in contact, move away)	0°	0 mm (0.0 in)
			10 mm (0.4 in)
			20 mm (0.8 in)
			30 mm (1.2 in)
			40 mm (1.6 in)
			50 mm (2.0 in)
			100 mm (3.9 in)
			150 mm (5.9 in)
			160 mm (6.3 in)
			170 mm (6.7 in)
			180 mm (7.1 in)
			190 mm (7.5 in)

4.7.2 Moto 50 Robot Kinematics

A PSS test similar to the Leica/T-MAC one was performed again, but this time the Moto 50 kinematic models were used to provide a baseline for performance. The z-distance (D_z) was increased from 0 to 200 mm in 10 mm increments. The moto 50 was used to maintain each distance in Table 4.4 for 10 s before transitioning to the next distance. CBSS logging was done with software this time and was initiated at contact (0 mm) and ended after the 200 mm test condition. Data from logging at 0 mm allowed sensor biases to be removed during data reduction. Only one test was performed; it should be noted again that the sensor recess in the palm (8.64 mm) was added to the table distances when analyzing the results.

Table 4.4: Proximity Characterization Test Matrix with Moto 50 Kinematic Models

Test #	Alignment	Angular Position (α)	z Distance (D_z)
1	Nominal Alignment (start in contact and move away)	0°	0 mm (0.0 in)
			10 mm (0.4 in)
			20 mm (0.8 in)
			30 mm (1.2 in)
			40 mm (1.6 in)
			50 mm (2.0 in)
			60 mm (2.4 in)
			70 mm (2.8 in)
			80 mm (3.1 in)
			90 mm (3.5 in)
			100 mm (3.9 in)
			110 mm (4.3 in)
			120 mm (4.7 in)
			130 mm (5.1 in)
			140 mm (5.5 in)
			150 mm (5.9 in)
			160 mm (6.3 in)
			170 mm (6.7 in)
			180 mm (7.1 in)
			190 mm (7.5 in)
			200 mm (7.9 in)

4.8 FSS Test Plan

This test plan featured four contact configurations of the CBSS with the FSAB Marman ring. They include perfect “nominal” contact of the CBSS with the Marman ring, misalignment where two RSS rails across two palm tools contacted the ring, misalignment where two RSS rails on a single palm tool contacted the ring, and misalignment where a single RSS rail contacted the ring. All contact configurations required data collection at no load (0 N) before and after each test for analysis of sensor drifts and bias removals in data reduction. The JR3 was logged continuously for the entire test plan, and the CBSS software was used to log data for each configuration.

4.8.1 Nominal Contact

Procedures for this experiment were similar to the first nominal contact, with the test # now referring to a single force condition rather than the entire experiment. The criteria in Table 4.5 contained five tests at three unique force levels (20 N, 100 N, and 200 N) and two tests at no load for 17 total tests. Each force condition in the table was maintained by the Moto 50 for 10 s as the CBSS software logged data. The applied loads were also randomized to avoid sensor hysteresis.

Table 4.5: Nominal Contact Test Cases

Test #	Alignment	Angular Position (α)	Force (F)
0.1	Nominal Contact	0°	0 N (0.0 lb)
1			200 N (45.0 lb)
2			20 N (4.5 lb)
3			100 N (22.5 lb)
4			20 N (4.5 lb)
5			100 N (22.5 lb)
6			200 N (45.0 lb)
7			100 N (22.5 lb)
8			20 N (4.5 lb)
9			200 N (45.0 lb)
10			100 N (22.5 lb)
11			200 N (45.0 lb)
12			20 N (4.5 lb)
13			200 N (45.0 lb)
14			100 N (22.5 lb)
15			20 N (4.5 lb)
0.2			0 N (0.0 lb)

4.8.2 Misalignment One – Two Palm, Two Rail Contact

Starting from the home position, the Moto 50 was used to tilt the CBSS away from the FSAB Marman ring such that only palm tools A and B would contact the ring. This configuration (shown in Figure 4.21) allowed contact to occur on RSS A1 and RSS B2, therefore only four load cells contributed to the global force calculations. A line linking the midpoints of the two contacted RSS rails served as the axis of rotation. Test cases were evaluated where the contact and ring planes intersected at 5° at the axis of rotation, but the test plan was written to include 10° and 15° as well. The reason those angles were selected is because the misalignment between the Marman ring and real CBS can be as large as fifteen degrees in the berthing phase [4].

Figure 4.21: Misalignment One (2 Palm Tools, 2 RSS, $\alpha=5^\circ$)

For misalignment one, three tests were performed at three unique force levels (20 N, 100 N, and 200 N) for $\alpha = 5^\circ$; the other rotations of this misalignment still need evaluated. However, the test matrix for all misalignment one test cases is shown in Table 4.6. The table proposes 27 force tests and two no load tests, for 29 tests in total. The first section of the table was completed with test 0.4 following test 24. Each force condition was maintained by the Moto 50 for 10 s as the CBSS software logged data. Again, the applied loads were randomized to combat hysteresis.

Table 4.6: Misalignment One Test Cases

Test #	Alignment	Angular Position (α)	Force (F)
0.3	Misalignment One – Two Palm, Two Rail (RSS) Contact	5°	0 N (0.0 lb)
16			200 N (45.0 lb)
17			20 N (4.5 lb)
18			100 N (22.5 lb)
19			20 N (4.5 lb)
20			200 N (45.0 lb)
21			100 N (22.5 lb)
22			200 N (45.0 lb)
23			20 N (4.5 lb)
24			100 N (22.5 lb)
25		10°	200 N (45.0 lb)
26			20 N (4.5 lb)
27			100 N (22.5 lb)
28			20 N (4.5 lb)
29			200 N (45.0 lb)
30			100 N (22.5 lb)
31			200 N (45.0 lb)
32			20 N (4.5 lb)
33			100 N (22.5 lb)
34		15°	200 N (45.0 lb)
35			20 N (4.5 lb)
36			100 N (22.5 lb)
37			20 N (4.5 lb)
38			200 N (45.0 lb)
39			100 N (22.5 lb)
40			200 N (45.0 lb)
41			20 N (4.5 lb)
42			100 N (22.5 lb)
0.4			0 N (0 lb)

4.8.3 Misalignment Two – Single Palm, Two Rail Contact

Starting from the home position, the Moto 50 was used to tilt the CBSS away from the FSAB Marman ring so only palm tool C would contact the ring. The depiction of misalignment two is shown in Figure 4.22. This configuration allowed contact to occur on RSS C1 and RSS C2, therefore only four load cells contributed to the global force calculations. A line linking the midpoints of the two contacted RSS rails served as the axis of rotation. Once again, test cases were evaluated where the contact and ring planes intersected at 5° at the axis of rotation, but the test plan was written to include 10° and 15° misalignments.

Figure 4.22: Misalignment Two (1 Palm Tool, 2 RSS, $\alpha=10^\circ$)

For misalignment two, three tests at three unique force levels (20 N, 100 N, and 200 N) with $\alpha = 5^\circ$ were performed in a randomized sequence; the other rotations of this misalignment still need evaluated. The test matrix encompassing all misalignment one test cases is shown in Table 4.7. The table proposes 27 force tests and two no load tests, for 29 tests in total. The first section of the table was completed with test 0.6 following test 51. Each force condition was maintained by the Moto 50 for 10 s as the CBSS software logged data.

Table 4.7: Misalignment Two Test Cases

Test #	Alignment	Angular Position (α)	Force (F)
0.5	Misalignment Two – Single Palm, Two Rail (RSS) Contact	5°	0 N (0.0 lb)
43			200 N (45.0 lb)
44			20 N (4.5 lb)
45			100 N (22.5 lb)
46			20 N (4.5 lb)
47			200 N (45.0 lb)
48			100 N (22.5 lb)
49			200 N (45.0 lb)
50			100 N (22.5 lb)
51			20 N (4.5 lb)
52		10°	200 N (45.0 lb)
53			20 N (4.5 lb)
54			100 N (22.5 lb)
55			20 N (4.5 lb)
56			200 N (45.0 lb)
57			100 N (22.5 lb)
58			200 N (45.0 lb)
59			100 N (22.5 lb)
60			20 N (4.5 lb)
61		15°	200 N (45.0 lb)
62			20 N (4.5 lb)
63			100 N (22.5 lb)
64			20 N (4.5 lb)
65			200 N (45.0 lb)
66			100 N (22.5 lb)
67			200 N (45.0 lb)
68			100 N (22.5 lb)
69			20 N (4.5 lb)
0.6			0 N (0 lb)

4.8.4 Misalignment Three – Single Palm, Single Rail Contact

From the home position, the Moto 50 was used to tilt the CBSS away from the Marman ring so that a single RSS on a single palm tool (RSS C2) contacted the ring. Two load cells contributed to the global force calculations in this scenario. An example of this configuration is depicted in Figure 4.23. A line connecting the midpoint of RSS C2 and the geometric center of the CBSS served as the axis of rotation. A planar tilt of 5° was the only rotation considered.

Figure 4.23: Misalignment Three (1 Palm Tool, 1 RSS, $\alpha=5^\circ$)

Three tests at three unique force levels (20 N, 100 N, and 200 N) were performed in a randomized sequence. The test matrix for this final misalignment is shown in Table 4.8. The table proposes nine force tests and two no load tests, for 11 tests in total. Each force condition in the table was maintained by the Moto 50 for 10 s as the CBSS software logged data.

Table 4.8: Misalignment Three Test Cases

Test #	Alignment	Angular Position (α)	Force (F)
0.7	Misalignment Three – Single Palm, Single Rail (RSS) Contact	5°	0 N (0.0 lb)
70			200 N (45.0 lb)
71			20 N (4.5 lb)
72			100 N (22.5 lb)
73			20 N (4.5 lb)
74			200 N (45.0 lb)
75			100 N (22.5 lb)
76			200 N (45.0 lb)
77			20 N (4.5 lb)
78			100 N (22.5 lb)
0.8			0 N (0.0 lb)

Considering nominal contact and the three misalignment contact configurations, 78 test cases and eight no load test cases were defined in this document, for 86 total tests. Although a subset of the 86 tests was performed, the test plan as a whole was proposed to complete proof of concept testing for the FSS prototype system.

4.9 Performance Evaluation

FSS force data from all 12 load cells were used to calculate the magnitude of the global force and the x-y torques on the CBSS prototype. This data was compared to truth data measured by the JR3 sensor on the Moto 50. When proximity tests were analyzed, CBSS proximity data was compared to truth data generated by either the Leica/T-MAC or the Moto 50 kinematics models. It is ideal if CBSS force, torque, and proximity data fall within the lab equipment's uncertainty (accuracy) limits. To evaluate CBSS subsystem performance, a standard percent error with respect to the truth data was calculated; Equation (4.1) shows this calculation. Furthermore, the average error of PE was also evaluated as a performance metric.

$$Percent\ Error = 100 * \left| \frac{(Truth\ Data - CBSS\ Data)}{Truth\ Data} \right| \quad (4.1)$$

The standard deviation for the percent errors was also calculated to see the precision of the error over the experiment. To simplify the nomenclature for defining the standard deviation, a given percent error vector is denoted by PE , while $PE(i)$ denotes an error at a certain index in that vector. AE indicates the average of the PE vector, and S represents the size of PE . The standard deviation used in data reduction is given by Equation (4.2).

$$Standard\ Deviation\ of\ the\ Error = \sqrt{\frac{\sum_{i=1}^S |PE(i) - AE|^2}{S - 1}} \quad (4.2)$$

Additionally, findings from multiple (repeated) trials were averaged to define the expected errors for a given experiment. Multiple trials were also analyzed to determine if repeatable results were obtained. The method chosen for showing repeatability was standard deviation, as it is an indicator of upper and lower limits referenced from the average containing approximately

68% of all errors within a given set of experimental trials [60]. For 95% confidence in results, meaning an interval containing 95% of the data of interest (errors or repeatability), one would multiply the standard deviation by two when defining tolerances of results [60]. Average errors and repeatability values in the results were reported at a 68% confidence level, by default. To define the repeatability equation, let the average error for every repeated trial form a vector denoted as trial error (TE); let $TE(i)$ indicate the error of a specific trial in vector TE . Let the average error among the trials be denoted by AT and the number of trials by n . With the nomenclature defined, Equation (4.3) shows how repeatability was calculated in data reduction.

$$Repeatability = \sqrt{\frac{\sum_{i=1}^N |TE(i) - AT|^2}{n - 1}} \quad (4.3)$$

CHAPTER 5 | RESULTS

5.1 Post Processing and Data Conditioning

The software suite used for processing experimental data was MATLAB R2017a [61]. Figure 5.1 shows the high level transformations of experimental data leading up to an analysis. CBSS sensor data and validation data from the lab equipment in Section 4.4.1 was loaded into MATLAB, conditioned, time aligned, and analyzed. The conditioning process varied for each CBSS sensor system but primarily involved interpolation, average filtering, and sensor tares. Average filtering and downsampling was applied to the validation data in the robot log to ensure equal sampling rates among the datasets. The bulk of this chapter (from Section 5.2 onward) includes analysis details for each test campaign but does not describe data conditioning. Ensuing subsections describe the conditioning process for each dataset in Figure 5.1.

Figure 5.1: Data Preparation for Analysis

5.1.1 CBSS Data – Force Conditioning

CBSS software in Chapter 4 had built-in loggers to capture raw sensor data from the FSS and PSS. CBSS data logs were standard text files containing sample timestamps, responses from 12 load cells, and distances from three range sensors. MATLAB code was written to split this data into distinct timestamp, load cell, and range sensor matrices. Load cell matrices were fed into five-point calibration models that were derived from Table 3.1 for converting load cell responses to forces in lbs. The table's applied force-load cell response pairs increasing from zero to 40 lbs (in 10 lb increments) were used to construct each piecewise model in Figure 5.2.

Figure 5.2: Load Cell Calibration Lines

Recall the load cell calibration line equations (3.1), (3.2), and (3.3). Applying said equations to the previously defined set of data points from Table 3.1 resulted in the line equations in Table 5.1. Those line equations make up the models for each CBSS load cell that converted mV/V responses to lb forces. The convention followed for locating a single load cell model was to observe where the rows for a particular palm tool and the column for the desired load cell intersect, resulting in a four row by one column area of the table. For example, to locate the model for load cell B2, the intersection of the rows in “Palm B” and the column denoted “(Load Cell 2)” is where one would look. The four rows for each model labeled “1” to “4” in the “Line” column correspond to the numeric labels for a particular group of line segments in Figure 5.2. Line equations in rows labeled with a “1” were used when a load cell’s response to a force was less than its mV/V threshold for a 10 lb force. Line equations in rows labeled with a “2” were used when a load cell’s response was between its mV/V thresholds for a 10 and 20 lb force. Line equations in rows labeled with a “3” were used when a load cell’s response was between its mV/V thresholds for a 20 and 30 lb force. Finally, line equations in rows labeled with a “4” were used when a load cell’s response to a force was greater than its mV/V threshold for a 30 lb force. After load cell matrices were transformed using Table 5.1 equations, the resulting lb forces were multiplied by 4.44822163 to convert them to Newton (N) forces for comparison to the JR3 data recorded in N.

Table 5.1: Piecewise MATLAB Load Cell Models

	Line	LCF =			
		(Load Cell 1)	(Load Cell 2)	(Load Cell 2)	(Load Cell 3)
Palm A	1	18.089*LCR-0.165	18.032*LCR-0.413	18.354*LCR-2.055	18.033*LCR+1.028
	2	17.892*LCR-0.054	17.775*LCR-0.265	17.905*LCR-1.760	17.711*LCR+1.188
	3	17.944*LCR-0.112	17.934*LCR-0.446	18.229*LCR-2.154	18.016*LCR+0.864
	4	17.707*LCR+0.285	17.602*LCR+0.119	17.814*LCR-1.422	17.500*LCR+1.699
Palm B	1	17.887*LCR+1.693	18.047*LCR+2.895	17.835*LCR+2.524	17.624*LCR+0.058
	2	17.847*LCR+1.712	17.829*LCR+2.981	17.829*LCR+2.526	17.561*LCR+0.094
	3	18.051*LCR+1.503	18.123*LCR+2.701	17.983*LCR+2.375	17.821*LCR-0.201
	4	17.680*LCR+2.089	17.520*LCR+3.609	17.721*LCR+2.779	17.440*LCR+0.445
Palm C	1	17.843*LCR+3.075	17.773*LCR-0.538	18.028*LCR+0.265	17.951*LCR+1.380
	2	17.659*LCR+3.147	17.568*LCR-0.417	17.812*LCR+0.381	17.892*LCR+1.409
	3	17.907*LCR+2.910	17.955*LCR-0.867	18.088*LCR+0.078	18.103*LCR+1.189
	4	17.579*LCR+3.406	17.551*LCR-0.171	17.659*LCR+0.786	17.750*LCR+1.751

The next step performed was interpolation to a constant sampling rate. Figure 5.3 shows a few interpolation techniques that were considered, including PCHIP, spline, and linear methods. The MATLAB function used for interpolating CBSS data was “interp1”. This function is a general purpose interpolation function that allows a user to select from multiple interpolation methods consisting of: nearest, next, previous, linear, spline, PCHIP, or cubic [62]. MATLAB documentation states that the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) is a shape-preserving method that requires at least four neighboring points and is computationally more expensive than linear but less than spline [62]. Based on this insight, PCHIP was selected; however, the choice between methods did not matter much. On average, a change in interpolation methods resulted in less than or near a 0.01% change in experimental errors.

Figure 5.3: Interpolation Techniques

The only reason data was interpolated was because the timestamps of the logged data drifted from theoretical timestamp values given a set sampling rate. This deviation from the expected timestamps varied depending on the logging software used. It is common knowledge that running a GUI application is more resource intensive than running a command line/terminal application with limited user output. The CBSS GUI had more inherent overhead (graphics rendering, multiple firing events, etc.) than the C++ logger, so timestamp drift was more noticeable. More time was required for logging events in the GUI to be completed when compared to the C++ logger, which had negligible time drift, but interpolation was used regardless.

After load cell force matrices were interpolated to a uniform sampling rate, the MATLAB code applied a one second moving average filter with the “smoothdata” function. The function was very useful for operating on multi-dimensional matrices and had many different smoothing methods with options for ignoring not a number (NaN) values in the data [63]. A moving average calculates the mean of every data sample that falls within a fixed size, sliding window that is translated across an entire set of data one sample at a time. It was decided this technique would help reduce any noise in the load cell force data so underlying trends could be observed more clearly. The window size for each filter was equal to the CBSS sampling rate for a given test, which was 5 or 10 Hz (meaning a window 5 or 10 samples wide). The final step in the force conditioning process was to tare every load cell to zero. Averages of the forces at no load conditions at the beginning of every test were evaluated and subtracted from each CBSS load cell in the load cell force matrices.

5.1.2 CBSS Data – Proximity Conditioning

Following force conditioning, CBSS proximity data was prepared for analysis. Range sensor matrices were ran through a separate process, beginning with a NaN injection. In the MATLAB code, values where the range sensors reported 255 were replaced with NaNs. A value of 255 meant a range sensor did not converge on a measurement within an allotted time, indicating there was not an object within range. Having NaNs as placeholders for these “no data” conditions were convenient when sending range sensor matrices through the desired MATLAB functions.

The remaining steps were nearly identical to the steps for the load cell force data, including interpolation to a constant sampling rate, one second moving average filtration, and the removal of tare values. However, the average filter was configured with the ignore NaN option enabled. The average was calculated based on the amount of actual numbers in the averaging window, not including NaNs. Tare values were evaluated before test runs to ensure the proximity sensors registered zero distance when in contact with the Marman ring. These tares were removed from range sensor matrices to conclude proximity conditioning.

5.1.3 Validation Data – JR3 Conditioning

Most experiments involved testing the performance of the FSS, with the exception of a few tests that focused on the PSS. JR3 data was often the only data required from lab metrology equipment after experiments were concluded. JR3 data was extracted from the robot log mentioned in Figure 5.1, saved as a matrix, and then sent through conditioning. The MATLAB code ran JR3 force matrices through a one second average filter and downsampled the data to match the CBSS sampling rate. Since the JR3 hardware enabled logging at 500 Hz, the averaging filter had a window size of 500 samples and the downsampling process kept every 100th or 50th sample to match the CBSS sampling rates of 5 Hz or 10 Hz. It was better to eliminate data from JR3 matrices rather than make assumptions and interpolate a large number of values between data points to upsample CBSS data.

Finally, a JR3 coordinate rotation was performed. Figure 5.4 shows an example of such a rotation on a generic point in the x-y plane at a given angle ϕ [64]. The picture shows a point (P) at location (x, y) and a new point (P') at location (x', y') after a rotation about the origin.

Figure 5.4: Coordinate Rotation [64]

The x-y Cartesian coordinate systems of the CBSS and JR3 were significantly misaligned when mounted on the Moto 50. However, a simple rotation matrix rectified this issue. The MATLAB code first calculated x- and y-coordinates for a vector of global forces in a JR3 data matrix from a given test. The x- and y-coordinate vectors were the result of dividing x- and y-torque vectors in the JR3 matrix by the global force vector. The MATLAB code then leveraged a common transformation matrix given by Equation (5.1) [64] to rotate the coordinate vectors about the JR3's origin at the geometric center of the device. Through experimentation with rotating coordinates, the optimal angle of rotation that minimized error between JR3 and CBSS data was approximately 240°. New x- and y-coordinate vectors, resulting from the rotation, were then multiplied by the JR3 global force vector to generate new JR3 x- and y-torque vectors. It should also be noted that the JR3 was zeroed with gravity compensation enabled before data was logged in an experiment. No tare values needed evaluated and removed when conditioning JR3 data in MATLAB.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (5.1)$$

5.1.4 Validation Data – Moto 50 Cartesian Position Conditioning

This data was only leveraged in a single PSS test intended to verify the performance of the CBSS range sensors against the Moto 50's approximate Cartesian position, based on robot kinematic models within the robot's control software. Moto 50 Cartesian data was extracted from the robot log mentioned in Figure 5.1, saved as a matrix, and then sent through conditioning. The MATLAB code ran the data through a one second average filter and then downsampled it to match the CBSS sampling rate. The parameters for average filtering and downsampling remained the same as those defined in Section 5.1.3. The differences between preparing JR3 data versus the Moto 50's Cartesian data were that a tare was required and no rotations were necessary. The Cartesian data contained x-, y-, and z-coordinate information for the Moto 50 robotic arm. Given the vertical nature of testing, only z-axis Moto 50 coordinates were utilized. An average of the z-coordinates at the beginning of the test during ring contact was removed from the dataset to zero the Moto 50's z-axis position.

5.1.5 Data Alignment

The final step to prepare for any given post-experimental analysis was to time-align CBSS data with the validation data from lab equipment. The main function the MATLAB code leveraged was “alignsignals”, which estimates the delay between two input signals and returns the aligned versions of them [65]. Conditioned CBSS and validation datasets from the previous subsections were input into the “alignsignals” function and shifted versions of them were returned. After alignment, a timestamp vector common to both datasets was generated based on a given experiment’s sampling rate.

Further investigation into “alignsignals” revealed that the function relies on “finddelay”, which relies on “xcorr” to compute the cross-correlation between two input signals [65] [66]. Signal delay is based on the location where the cross-correlation has the largest absolute value [66]. Every entry in a cross-correlation vector is the sum of the multiplication between one of the signals and a shifted version of the other one [67]. That sum is computed every time the sliding signal shifts right by one sample [67]. Figure 5.5 provides an illustration of a cross-correlation for the signals shown in the top-left plot. The red pulse is considered the “early signal” and lags the blue pulse by one second. To find signal delay, the red pulse is shifted across the blue pulse one sample at a time as the sum of the two multiplied signals is found. The top-right set of plots show a few cases in this shifting process, while the bottom plot shows a vector of those computed sums at a given time lag. The bottom plot shows that the highest correlation between signals occurs at one second. If the red signal is delayed one second, it will be aligned with the blue signal. Finding signal delay is how the CBSS and validation data from lab equipment was aligned for analyses.

Figure 5.5: Finding Signal Delay

5.2 Nominal Contact FSS Testing

A series of three tests were performed to gain insight into the performance of the system's force tracking capabilities in a nominal contact scenario. Refer to Section 4.4.2 – Section 4.4.4 for the general test setup, data management, and independent variables; refer to Section 4.5 for the experimental procedure. Regarding the test setup, the palm tools were mounted such that the centers of the palms were seated on a common radius of 601 ± 5 mm. Palm tool contact surfaces (RSS top rails) were measured by the Leica to be within 10 mm of coplanar. The CBSS prototype did not have systems in place for fine adjustments to tighten those tolerances. The lack of fine tuning affected nominal contact and was a source of error in the results.

Figure 5.6 is a plot that captures the fundamental ideas behind the FSS regarding contact evaluation. The blue, red, and green signals represent the local forces on palm tools A, B, and C over the duration of the test. The black signal is the global force, which is the sum of the local forces. It can be seen that palm tools B and C make contact at 50 seconds, while tool A contacts later around 95 seconds. This shows that nominal contact was not exactly achieved. The CBSS contact plane was not parallel with the FSAB Marman ring plane despite efforts to orient them as such. Palm tool A only registered a force because the CBSS arms and other 3D printed components flex noticeably under load. F_A , F_B , and F_C would have been equal over the duration of all three tests had nominal contact been truly achieved.

Figure 5.6: FSS Fundamental Concepts (June 2016, Test #1)

Figure 5.7 shows how well the FSS tracked the global force and axis torques when compared to data from the JR3 in test one. Plots one, two, and three show the global force, x-axis torque, and y-axis torque, respectively. The tracking of the global force magnitude and axis torques were very good despite the inadequacies of the CBSS mentioned at the beginning of this section. The 601 mm common radius was used for every palm tool when processing results, so that was a source of error. 3D printed composite structures and prototype flexing were also unquantifiable errors. However, visual inspection shows that the difference between the FSS and JR3 in the global force, x-axis torque, and y-axis torque was approximately 0-7 N, 0-2 Nm, and 0-6 Nm.

Figure 5.7: FSS to JR3 Force-Torque Comparison (June 2016, Test #1)

The main metric for quantifying FSS performance was percentage error against the JR3 data. Figure 5.8 shows the error over the entire test for the global force and calculated torques. Data at zero force and the transition points where the amount of force applied to the Marman ring changed produced large spikes in percent error. These large spikes lessened the ability to evaluate the overall performance of the FSS and made it nearly impossible to visualize the trends. The data at no load conditions and the transition points had to be removed to properly analyze the overall error.

Figure 5.8: FSS Percent Errors without Data Deletion (June 2016, Test #1)

Figure 5.9 depicts the selection process used to eliminate problematic data in all three tests. The boxes in the plots indicate the selected data from nine distinct regions for the global force and x-y tool torque. The centers of each region were found and 125 samples were extracted from both the CBSS and JR3 datasets for comparison. These 125 samples consisted of 62 samples left of center, the center value, and 62 samples right of center. The nine regions defined here will be referenced in a later plot (Figure 5.14) when the average error at each region is analyzed.

Figure 5.9: Data Selection (June 2016, Test #1)

Figure 5.10 shows the data stitched together. This extraction process was performed for all three tests before analyzing and reporting FSS performance. However, the data was no longer in the continuous time domain, so it was referred to by the sample number. The first 125 samples had an applied force of about 50 N, samples 126-250 had an applied force of about 100 N, and so on up to 250 N. The applied force was then removed in 50 N increments until the test was complete. Sample numbers are also referenced in later plots (Figure 5.11 and Figure 5.13).

Figure 5.10: Stitched Data (June 2016, Test #1)

Figure 5.11 shows FSS percent errors against the JR3 for test one with means and standard deviations overlaid. Plots one, two, and three represent the error in the global force, x-axis torque, and y-axis torque, respectively. The red dotted lines show the mean errors, and the light red shaded regions constrained by upper (UB) and lower (LB) bounds indicate plus or minus single standard deviations from the means. One standard deviation contains approximately 68% of the data, whereas two would contain 95%. The means, standard deviations, UBs, LBs, minimum errors, and maximum errors are clearly labeled in the plots. Regarding trends, errors in the global force, x-torque, and y-torque increased with load. This was expected due to 3D printed components and the inherent flex when loaded; the CBSS arms flexed as well.

Figure 5.11: FSS Percent Errors with Data Deletion (June 2016, Test #1)

The plots for the remaining tests are located in APPENDIX C for reference. Table 5.2 contains the results for all three tests in this campaign and some useful statistics. The average trial for all the tests is highlighted in yellow. It shows that the average FSS error in the global force magnitude was $2.16\% \pm 0.95\%$ (ave $\pm$ std), meaning the majority (68%) of all errors, on average, were between 1.21% and 3.11%. Average minimum and maximum errors were also reported and were 0.18% and 3.19%. The standard deviation of all the tests was also calculated to show repeatability of the FSS. The standard deviation row is highlighted in light red and was computed for all column categories for the sake of completeness. The important table cells to review include where the average and standard deviation columns intersect the standard deviation row. If more identical tests were performed, one would expect (with 68% confidence) the global force magnitude error to be repeatable within $\pm 0.31\%$ of the average. The standard deviation would be expected to be repeatable within $\pm 0.18\%$ of the average.

Regarding torque, the average trial shows that the x-axis torque error was $10.48\% \pm 6.79\%$. The majority of all errors were between 3.69% and 17.27%, with minimum and maximum errors of 1.07% and 22.08%. If more tests were performed, one would expect the x-axis torque error and standard deviation to be repeatable within $\pm 1.92\%$ and $\pm 0.29\%$ of the average. The average y-axis torque error was $9.73\% \pm 3.14\%$. The majority of all errors were between 6.59% and 12.87% with minimum and maximum errors of 4.36% and 14.58%. With more tests, one would expect the y-axis torque error and standard deviation to be repeatable within $\pm 0.4\%$ and $\pm 0.37\%$ of the average. Just as the force in the previous paragraph, the torque error analysis was based on the 68% confidence level. 68% confidence has been emphasized here, but it will be assumed in future analyses without being explicitly stated. Furthermore, if one requires errors and repeatability at a 95% confidence level, simply double the standard deviations.

Table 5.2: June 2016 Overall FSS Error Analysis

	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
	Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
Test 1	1.85	0.74	0.09	2.71	12.65	6.89	0.73	22.56	9.57	3.27	3.50	14.70
Test 2	2.18	1.06	0.05	3.36	9.74	7.02	1.42	22.32	9.44	2.72	5.12	13.62
Test 3	2.46	1.06	0.41	3.49	9.03	6.47	1.05	21.35	10.18	3.42	4.47	15.41
Ave	2.16	0.95	0.18	3.19	10.48	6.79	1.07	22.08	9.73	3.14	4.36	14.58
Std	0.31	0.18	0.20	0.42	1.92	0.29	0.34	0.64	0.40	0.37	0.81	0.90

Another useful data visualization tool was the video playback of the entire test. Figure 5.12 shows a single frame from the video. The main graphic displayed local and global force locations on a polar grid relative to the CBSS and palm tool contact zones. The CBSS contact zone was the dashed, triangular shape and was the geometrical perimeter of the load cells designated by the white circles. Theoretically, the global force should have always resided in this boundary. The dashed boxes represented the palm tool contact zones where the local forces were supposed to exist. The centroid of the CBSS contact zone and the center of the polar grid were coincident. The spacing between concentric circles in the polar grid was 12.7 cm (5 in). The CBSS x- and y-axes were the horizontal and vertical lines of the polar grid. The CBSS global and local forces were plotted with circles, and the JR3 global force was a diamond. Force magnitudes from zero to 300 N were represented through a color gradient from white to dark red. Lastly, the plot at the bottom showed the global force of the JR3 and the CBSS, as well as the current timestamp of the data. It should be noted that typical CBSS x- and y-coordinate differences in the global force, compared to the JR3, were less than approximately 4 cm (1.57 in).

Figure 5.12: Video Playback Tool (June 2016, Test #1)

Figure 5.13 was generated to support the previous claim. The plots show that the maximum difference in global force x- and y-coordinates, compared to the JR3, was about 2.5 cm (0.98 in) and 4 cm (1.57 in), respectively. The differences in the x-coordinates were low ($< \sim 1$ cm) until the end of the test. The differences in the y-coordinates were greater (~ 3 cm or less), except at the beginning of the test. These plots were shown to strengthen the idea that although there are larger percentage errors in the torque comparisons, they sometimes only correspond to a small distance error.

Figure 5.13: Differences in FSS and JR3 Global Force Locations (June 2016, Test #1)

Recall Figure 5.9 where data was selected out of nine distinct regions. Figure 5.14 shows average errors and standard deviations in each of those regions. The blue signals represent region error and the shaded, red areas around them show standard deviation. Errors over the whole test deviated much more from the mean than what was observed in a specific region at a specific applied force. The repeatability of the error in each region was exceptional. One can barely see the red, shaded areas around the blue signals. Another noteworthy quality is the approximate

symmetry of the top plot. Hysteresis was largely ignored because the effects observed when loading and unloading were negligible. Once again, error increasing with load was fully expected. Flexing in the 3D printed RSS contact rails, 3D printed palm tools, and CBSS arms reduced the ability of the contact rails to effectively couple the applied force to the load cells embedded in the RSS modules.

Figure 5.14: Average FSS Percent Errors at each Applied Force (June 2016, Test #1)

Because the standard deviations in the previous plots could barely be observed, Table 5.3 and Table 5.4 were provided. The table is very large and will not be covered in-depth. However, the standard deviation columns should be noted. At most applied forces, the standard deviations of the force and torque errors were incredibly low. This indicates that the FSS load cells displayed very high precision for the applied forces. These results were impressive and speak volumes about the fidelity of COTS Omega load cells. The average trial and experimental repeatability for each metric at every applied force is also shown.

Table 5.3: June 2016 FSS Error Analysis for each Applied Force

		% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
(~50 N)	Test1	0.38	0.14	0.09	0.66	0.90	0.09	0.73	1.06	12.43	0.06	12.35	12.58
	Test2	0.55	0.10	0.35	0.78	1.56	0.07	1.42	1.65	13.46	0.06	13.37	13.62
	Test3	0.59	0.10	0.41	0.81	1.69	0.06	1.59	1.78	13.90	0.08	13.80	14.05
	Ave	0.51	0.11	0.29	0.75	1.38	0.08	1.24	1.50	13.26	0.07	13.17	13.42
	Std	0.11	0.02	0.17	0.08	0.43	0.02	0.45	0.39	0.75	0.01	0.74	0.76
(~100 N)	Test1	1.90	0.08	1.70	2.02	5.25	0.07	5.12	5.39	6.90	0.02	6.84	6.93
	Test2	2.55	0.03	2.50	2.61	5.54	0.06	5.42	5.62	6.96	0.02	6.90	6.99
	Test3	2.63	0.02	2.59	2.66	5.32	0.06	5.21	5.43	7.58	0.03	7.52	7.63
	Ave	2.36	0.04	2.26	2.43	5.37	0.06	5.25	5.48	7.15	0.02	7.09	7.18
	Std	0.40	0.03	0.49	0.35	0.15	0.01	0.15	0.13	0.38	0.005	0.38	0.39
(~150 N)	Test1	2.42	0.03	2.37	2.50	6.59	0.19	6.18	6.82	8.77	0.05	8.69	8.83
	Test2	3.34	0.02	3.29	3.36	7.01	0.15	6.74	7.23	8.55	0.02	8.50	8.58
	Test3	3.38	0.02	3.34	3.41	6.60	0.07	6.47	6.70	9.11	0.02	9.06	9.14
	Ave	3.05	0.02	3.00	3.09	6.74	0.14	6.46	6.92	8.81	0.03	8.75	8.85
	Std	0.54	0.01	0.55	0.51	0.24	0.06	0.28	0.28	0.28	0.02	0.28	0.28
(~200 N)	Test1	2.35	0.01	2.33	2.37	11.45	0.27	10.90	11.83	9.60	0.04	9.52	9.66
	Test2	2.85	0.03	2.79	2.91	5.90	0.21	5.39	6.19	9.40	0.02	9.35	9.44
	Test3	3.27	0.02	3.23	3.31	5.64	0.14	5.34	5.93	10.23	0.03	10.17	10.28
	Ave	2.82	0.02	2.78	2.86	7.66	0.21	7.21	7.98	9.74	0.03	9.68	9.79
	Std	0.46	0.01	0.45	0.47	3.28	0.07	3.19	3.33	0.43	0.01	0.43	0.44
(~250 N)	Test1	2.65	0.04	2.59	2.71	20.04	0.36	19.29	20.75	12.85	0.08	12.69	12.95
	Test2	3.27	0.04	3.18	3.32	3.31	0.60	2.31	4.49	12.83	0.06	12.68	12.92
	Test3	3.47	0.01	3.44	3.49	1.72	0.51	1.05	2.58	13.74	0.06	13.61	13.84
	Ave	3.13	0.03	3.07	3.17	8.36	0.49	7.55	9.27	13.14	0.06	13.00	13.24
	Std	0.43	0.02	0.44	0.41	10.15	0.12	10.19	9.98	0.52	0.01	0.53	0.52

Table 5.4: June 2016 FSS Error Analysis for each Applied Force (continued)

		% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
(~200 N)	Test1	2.22	0.01	2.20	2.25	22.16	0.21	21.77	22.56	14.63	0.04	14.57	14.70
	Test2	2.76	0.01	2.74	2.79	21.51	0.06	21.37	21.64	12.06	0.03	11.99	12.11
	Test3	3.23	0.01	3.20	3.25	20.99	0.17	20.74	21.35	15.32	0.04	15.25	15.41
	Ave	2.74	0.01	2.72	2.76	21.55	0.15	21.30	21.85	14.00	0.04	13.94	14.07
	Std	0.51	0.0005	0.50	0.50	0.59	0.08	0.52	0.63	1.72	0.01	1.72	1.74
(~150 N)	Test1	2.24	0.01	2.20	2.26	17.95	0.08	17.72	18.09	10.42	0.02	10.39	10.45
	Test2	2.12	0.02	2.10	2.16	22.21	0.06	22.07	22.32	9.65	0.06	9.52	9.73
	Test3	2.82	0.02	2.78	2.86	18.47	0.03	18.40	18.54	10.60	0.04	10.51	10.66
	Ave	2.39	0.02	2.36	2.43	19.54	0.06	19.40	19.65	10.22	0.04	10.14	10.28
	Std	0.37	0.004	0.37	0.38	2.32	0.03	2.34	2.32	0.50	0.02	0.54	0.49
(~100 N)	Test1	1.74	0.02	1.70	1.79	17.31	0.03	17.25	17.38	6.93	0.02	6.86	6.97
	Test2	2.01	0.03	1.94	2.06	10.70	0.05	10.57	10.76	6.82	0.05	6.72	6.89
	Test3	2.10	0.02	2.06	2.14	11.49	0.04	11.37	11.56	6.61	0.05	6.49	6.67
	Ave	1.95	0.02	1.90	2.00	13.17	0.04	13.06	13.24	6.78	0.04	6.69	6.84
	Std	0.19	0.005	0.19	0.19	3.61	0.01	3.65	3.61	0.16	0.02	0.19	0.15
(~50 N)	Test1	0.72	0.06	0.61	0.89	12.17	0.05	12.01	12.24	3.55	0.02	3.50	3.60
	Test2	0.18	0.04	0.05	0.26	9.92	0.04	9.78	9.99	5.19	0.03	5.12	5.24
	Test3	0.68	0.11	0.43	0.87	9.35	0.08	9.16	9.46	4.55	0.04	4.47	4.64
	Ave	0.53	0.07	0.36	0.67	10.48	0.06	10.32	10.56	4.43	0.03	4.36	4.49
	Std	0.30	0.03	0.29	0.36	1.49	0.02	1.50	1.48	0.82	0.01	0.81	0.83

5.3 Misalignment to Nominal Contact FSS Testing

11 tests were performed to evaluate the performance of the FSS during an alignment correction procedure, as seen in Figure 5.15. The Moto 50 was used to correct a five degree misalignment, with ring contact occurring on one palm tool, to nominal, 3-palm contact. Refer to Section 4.4.2 – Section 4.4.4 for the generic setup/overview and Section 4.6 for the procedure. The palm tools were again within 10 mm of coplanar and within 5 mm of a common radius. However, this time robot operators experimented with CBSS x-y rotations while contacting the Marman ring and monitored when nominal contact was achieved. That position was recorded and then repeated during testing to ensure the CBSS contacted the ring nominally in the second half of the test procedure. A subsequent set of plots (Figure 5.16) shows that nominal contact was much better in this test campaign.

Figure 5.15: Alignment Correction Maneuver

Figure 5.16 includes the local and global CBSS force magnitudes, along with palm tool proximities to the Marman ring. The range sensors were operational for this test campaign, but were not analyzed in this phase. The PSS provided a redundant method for approximating palm tool ring contact. As seen in the plots, the load cells registered forces near the time the range sensors displayed zero millimeters. The PSS also extended the functionality of the CBSS to calculate x- and y- axis rotation angles required to correct the alignment to the Marman ring based on proximity measurements P_A , P_B , and P_C before contact. Ideally, remote operators or

autonomous algorithms would use proximity information to align a satellite with a servicing vehicle during the berthing phase. In Figure 5.16, the range sensors showed that palm A made contact around 21 seconds, after which the load cells began showing a force of about 50 N. The other two palms contacted around 36 seconds, and the load cells began showing forces around 63 N. Also note that local forces F_A , F_B , and F_C were much closer to being equal in the second half of this test, with the total global force in the top plot being near 180 N. Nominal contact was better achieved throughout this test campaign.

Figure 5.16: CBSS Fundamental Concepts (August 2016, Test #1)

Figure 5.17 shows how well the FSS tracked the global force and torque when compared to data from the JR3 sensor in the first test. These plots follow previously established conventions regarding the information they show and how it is conveyed. Once again, the FSS did very well at monitoring force and torque given the limitations of the prototype system. Differences in the global force between 21 to 47 seconds visually appeared to be 5-10 N, while differences in the x- and y-axis torques in the same time frame appeared to be 0-2 Nm and 2-8 Nm.

Figure 5.17: FSS to JR3 Force-Torque Comparison (August 2016, Test #1)

Figure 5.18 shows the data selection process for this group of experiments. The boxes indicate the selected data in two distinct regions of the test for each metric of interest. The regions are labeled with “1” and “2”. Region one consisted of initial contact on palm tool A and the corrective procedure to reach nominal contact. The second region was where the CBSS indicated 180 N, nominal contact. The duration of region one was about 16 seconds, and region two was about 10 seconds. This varied from test to test but most had comparable time frames, with the exception of tests nine and ten. The regions in those tests were five to ten seconds longer. The centers of each region for every test were found and 80% of the data was extracted. This resulted in approximately 120 and 70 samples being pulled from regions one and two on average; tests nine and ten had much more samples extracted.

Figure 5.18: Data Selection (August 2016, Test #1)

Figure 5.19 shows FSS percentage error against the JR3 and references the sample number based on data extracted in Figure 5.18. These plots follow the same conventions established in Figure 5.11. The means, standard deviations, UBs, LBs, minimum errors, and maximum errors are clearly indicated in the plots. The errors in the global force were higher in misalignment than nominal contact. This was expected because the FSS load cells were intended to be loaded on the z-axis. For misalignment in general, there will be some components of the applied force that lie on axes that the FSS load cells are insensitive to detecting, resulting in higher error. Surprisingly, the errors in the torques were lower in misalignment, but this will be further investigated later.

Figure 5.19: FSS Percent Errors with Data Deletion (August 2016, Test #1)

The plots for the rest of the tests are located in APPENDIX D for reference. Table 5.5 contains the results for all 11 tests in this campaign. The average trial for all tests is highlighted in yellow. It shows that the average percent error in the global force magnitude was $6.84\% \pm 3.14\%$. Errors, on average, were between 3.7% and 9.98% with minimum and maximum errors of 2.94% and 10.34%. The repeatability for all the tests (highlighted in light red) shows that if more identical tests were performed, one could expect the global force magnitude error and standard deviation to be within $\pm 0.39\%$ and $\pm 0.33\%$ of the average.

The average trial also shows that one can expect the x-axis torque error to be $19.92\% \pm 12.46\%$. Errors, on average, were between 7.46% and 32.38% with minimum and maximum errors of 1.95% and 36.24%. If more tests were performed, one could expect the x-axis torque error and standard deviation to be within $\pm 3.01\%$ and $\pm 2.13\%$ of the average. The average y-axis torque error was $19.17\% \pm 3.17\%$. Errors, on average, were between 16% and 22.34% with minimum and maximum errors of 7.12% and 21.65%. With more tests, one could expect the y-axis torque error and standard deviation to be within $\pm 0.42\%$ and $\pm 0.85\%$ of the average.

Table 5.5: August 2016 Overall FSS Error Analysis

	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
	Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
Test 1	6.58	2.79	1.62	9.86	22.78	17.52	0.24	45.83	18.93	4.00	4.38	22.38
Test 2	7.18	2.84	3.85	10.40	23.97	11.53	5.37	40.46	18.40	2.84	6.92	20.28
Test 3	6.02	2.50	2.95	9.15	21.25	14.81	0.06	41.19	19.59	4.82	5.58	24.75
Test 4	7.09	3.22	3.27	10.66	21.25	11.81	1.62	37.38	18.98	3.26	6.47	21.21
Test 5	6.76	3.01	3.11	10.12	20.70	11.12	3.39	35.41	18.99	3.04	7.02	21.11
Test 6	6.57	3.15	2.99	10.30	21.37	10.76	2.95	34.34	19.18	3.41	6.06	21.57
Test 7	6.73	3.22	3.04	10.54	20.99	11.08	3.87	35.30	18.98	3.12	6.34	21.27
Test 8	6.70	3.30	2.95	10.44	19.03	11.30	0.00	33.54	19.21	3.34	7.40	21.97
Test 9	7.16	3.48	2.89	10.51	15.19	13.27	1.13	31.97	19.96	1.40	12.15	20.90
Test 10	7.38	3.48	2.86	10.75	14.15	13.28	0.02	31.38	19.58	2.46	8.77	21.24
Test 11	7.10	3.54	2.85	10.99	18.48	10.58	2.81	31.80	19.10	3.23	7.27	21.49
Ave	6.84	3.14	2.94	10.34	19.92	12.46	1.95	36.24	19.17	3.17	7.12	21.65
Std	0.39	0.33	0.52	0.50	3.01	2.13	1.84	4.58	0.42	0.85	2.01	1.16

Figure 5.20 was created to show differences in the x- and y-coordinates of the global force between the FSS and JR3 systems. These plots were shown to further explore the large percentage errors in the x- and y-axis torques. There were some very large standard deviations in the average x-axis torque errors, with maximum values reaching around 46% error throughout the tests. It should be noted that larger errors exist in the second region of the tests because the CBSS was in nominal contact with the Marman ring. Low global torque values were expected because of individual offsetting torques at different sensor locations. In nominal contact, the location of the global force should be close to the centroid of the tool near x- and y-coordinates (0, 0). When dealing with low values, percentage error can be really high. For example, the percent error between two and one is 50%. The percent error is huge but the difference is one. That is the case with the CBSS in these tests at nominal contact. It can be seen in the plots below that the maximum difference in the global force's x- and y-coordinates was about 2 cm (0.79 in) and 7 cm (2.76 in), respectively. The difference in the x-coordinates was overall very low (less than ~1.5 cm) despite having an error greater than 40% from sample 120 onward. The same can be said for the y-axis coordinates. The differences from sample 120 onward were less than approximately 1 cm. Together, the x- and y-coordinates were very close to (0, 0).

Figure 5.20: Differences in FSS and JR3 Global Force Locations (August 2016, Test #1)

Quantifying overall error for entire tests can be insightful, but looking at unique conditions is equally important. In these tests the unique regions were the initial 50 N contact with the ring where the alignment correction occurred and the 180 N nominal contact that followed. Figure 5.21 is a set of average and standard deviation boxplots for both of those regions for the FSS force and torque errors, with overlays of the underlying data samples. The averages are shown with red lines, and the standard deviations are the light purple boxes. The data samples were randomly scattered over the x-axes to avoid having sets of vertical points. It was desired to show force and torque errors from both test regions within the same plots on the same y-axis scales, but the averages were too far apart to see both boxplots.

Figure 5.21: Average FSS Percent Errors for both Test Regions (August 2016, Test #1)

The sample overlays revealed trends that mean and standard deviation analyses alone do not always show. For instance, the averages are skewed up or down and the standard deviations are larger in the left plots because of the alignment correction. Percentage error fluctuated more as the Moto 50 steered the CBSS into nominal contact with the Marman ring. Until that event though, the error was pretty steady and consistent. Note the tightly grouped data points that form lines above or below the average lines in the plots to the left. Visual inspection of those plots show the mean $\pm$ standard deviation pairs for the average FSS error in the global force, x-torque, and y-torque to be approximately $8.3\% \pm 2\%$, $10\% \pm 8\%$, and $17\% \pm 4\%$, respectively. If this test region only contained the five degree off-nominal ring contact without a corrective maneuver, the means would be near 10%, 5%, and 19% all with low deviations. The previous test campaign indicated that errors in the load cells at a given applied force were very repeatable, often under 0.1%. Because of these results, low standard deviations were assumed in general and in this misalignment scenario. The FSS was tested again in this particular off-nominal configuration alone in later tests, so the low deviation assumption made here was verified in the FSS test plan results (Section 5.5).

Regarding the right plots depicting the nominal contact region of the test in Figure 5.21, the FSS performed as expected based on the previous experiments. The average FSS global force, x-torque, and y-torque errors appeared to be $3.77\% \pm 0.02\%$, $43.4\% \pm 0.7\%$, and $22.15\% \pm 0.1\%$, respectively. The average force error of 3.77% with that extremely low standard deviation was comparable to what was observed before. The errors in x-y torque were higher this time though. However, as stated earlier, nominal contact was more closely achieved in these tests. The observed x- and y-torques were quantitatively much smaller due to equal offsetting torques on the CBSS, resulting in high percent errors. Percent error can be very large even with a miniscule difference in data when dealing with small numbers.

Table 5.6 provides a breakdown for all the tests in both the initial contact and nominal contact regions of the data. The average trial shows that the FSS error in the global force, x-torque, and y-torque during initial contact with the alignment correction was $9.03\% \pm 1.76\%$, $11.47\% \pm 7.71\%$, and $17.97\% \pm 3.43\%$. The repeatability of those average errors was $\pm 0.65\%$, $\pm 3.33\%$, and $\pm 0.8\%$, respectively. Additionally, the repeatability of the average standard deviations was ± 0.3 , ± 0.88 , and ± 0.64 . The average trial also shows that the average FSS error in

the global force, x-torque, and y-torque during nominal contact was $3.28\% \pm 0.06\%$, $33.68\% \pm 1.04\%$, and $21.14\% \pm 0.15\%$. The repeatability of those average errors was $\pm 0.32\%$, $\pm 4.41\%$, and $\pm 1.38\%$, respectively. Additionally, the repeatability of the average standard deviations was $\pm 0.03\%$, $\pm 0.2\%$, and $\pm 0.07\%$. One can see that the standard deviation columns are all very low in nominal contact, indicating how precise the load cells were.

Table 5.6: August 2016 FSS Error Analysis for both Test Regions

		% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
Initial Contact with Correction	Test 1	8.33	2.16	1.62	9.86	9.96	8.27	0.24	29.63	16.93	3.93	4.38	19.32
	Test 2	9.14	1.62	4.14	10.40	15.93	6.21	5.37	29.96	18.14	3.57	6.92	20.28
	Test 3	7.61	1.77	2.95	9.15	11.00	7.96	0.06	29.00	16.68	3.73	5.58	19.05
	Test 4	9.29	1.91	3.99	10.66	13.53	7.90	1.62	29.76	17.84	3.67	6.47	20.09
	Test 5	8.90	1.61	4.28	10.12	13.31	7.42	3.39	29.14	17.90	3.44	7.02	19.99
	Test 6	8.89	1.85	3.72	10.30	14.22	8.32	2.95	30.72	17.72	3.79	6.06	20.12
	Test 7	8.99	1.95	3.31	10.54	13.38	7.13	3.87	30.06	17.94	3.61	6.34	20.22
	Test 8	9.04	1.92	4.42	10.44	11.85	8.65	0.00	29.91	17.88	3.68	7.40	20.16
	Test 9	9.97	1.02	6.22	10.51	5.03	6.48	1.13	25.62	19.56	1.70	12.15	20.28
	Test 10	9.70	1.63	4.56	10.75	6.18	9.09	0.02	28.47	18.97	2.86	8.77	20.42
	Test 11	9.52	1.95	4.69	10.99	11.79	7.43	2.81	28.81	18.15	3.74	7.27	20.53
	Ave	9.03	1.76	3.99	10.34	11.47	7.71	1.95	29.19	17.97	3.43	7.12	20.04
	Std	0.65	0.30	1.15	0.50	3.33	0.88	1.84	1.35	0.80	0.64	2.01	0.45
Nominal Contact	Test 1	3.77	0.02	3.74	3.81	43.33	0.75	42.19	45.83	22.16	0.09	22.00	22.38
	Test 2	3.91	0.02	3.85	3.95	37.39	1.16	36.08	40.46	18.86	0.07	18.73	19.07
	Test 3	3.34	0.05	3.20	3.39	38.63	1.18	37.12	41.19	24.56	0.12	24.36	24.75
	Test 4	3.43	0.07	3.27	3.53	34.12	1.36	32.56	37.38	20.92	0.08	20.77	21.21
	Test 5	3.29	0.09	3.11	3.39	32.68	1.21	31.27	35.41	20.77	0.11	20.56	21.11
	Test 6	3.18	0.08	2.99	3.27	31.74	1.10	30.39	34.34	21.33	0.10	21.14	21.57
	Test 7	3.18	0.08	3.04	3.29	32.94	1.00	31.81	35.30	20.63	0.25	20.31	21.27
	Test 8	3.03	0.03	2.95	3.08	30.32	1.13	28.87	33.54	21.33	0.19	21.12	21.97
	Test 9	3.08	0.08	2.89	3.19	29.99	1.01	28.57	31.97	20.55	0.15	20.35	20.90
	Test 10	2.93	0.05	2.86	3.04	29.39	0.87	28.38	31.38	20.74	0.23	20.38	21.24
	Test 11	2.94	0.06	2.85	3.03	29.93	0.69	28.77	31.80	20.75	0.23	20.38	21.49
	Ave	3.28	0.06	3.16	3.36	33.68	1.04	32.37	36.24	21.14	0.15	20.92	21.54
	Std	0.32	0.03	0.34	0.30	4.41	0.20	4.40	4.58	1.38	0.07	1.39	1.35

5.4 Nominal Alignment PSS Testing

Two experiments were performed to validate the range sensors of the PSS subsystem. These tests were both performed in nominal alignment with the FSAB Marman ring, meaning equal palm distances to the ring when contact is not established. The CBSS began in nominal contact with the ring and was lowered via the Moto 50, while pausing at various distances for data collection at a given proximity. Refer to Section 4.4.2 – Section 4.4.4 for details on the setup and Section 4.7 for the procedure. The T-MAC was attached to the Moto 50 and tracked by the Leica in the first test for PSS range sensor validation, while the second test relied on kinematic models in the Moto 50 robot control software.

5.4.1 Leica Tracking T-Mac

Table 5.7 contains manually collected proximity data from each range sensor for each palm tool. Data was obtained from observing the minimum and maximum values at every Leica distance and then averaging the two. The actual procedure in Table 4.3 referenced test points at 0 mm, 10 mm, and so on, while this table does not. The sensor recess distance, referenced from the RSS contact rails, was added to convey true sensor distance.

Table 5.7: Manually Collected Proximity Data (October 2016, Test #1)

Region	Leica (mm)	P _A (mm)			P _B (mm)			P _C (mm)		
		Min	Max	Ave	Min	Max	Ave	Min	Max	Ave
1	8.64	8.64	9.64	9.14	8.64	9.64	9.14	8.64	8.64	8.64
2	18.64	20	21	20.50	22	24	23.00	17	18	17.50
3	28.64	27	29	28.00	31	33	32.00	25	27	26.00
4	38.64	36	40	38.00	42	43	42.50	32	34	33.00
5	48.64	46	49	47.50	50	52	51.00	40	42	41.00
6	58.64	56	58	57.00	59	61	60.00	50	52	51.00
7	108.64	104	107	105.50	106	109	107.50	101	104	102.50
8	158.64	153	157	155.00	157	159	158.00	148	150	149.00
9	168.64	165	166	165.50	165	167	166.00	157	160	158.50
10	178.64	174	176	175.00	176	178	177.00	165	167	166.00
11	188.64	187	189	188.00	184	187	185.50	175	177	176.00
12	198.64	-	-	-	195	-	-	187	188	187.50

The averaged data from Table 5.7 was compared to the Leica data in Figure 5.22. It can be seen that all the palm proximities P_A , P_B , and P_C were very close to the Leica measurements. P_C showed the greatest deviation, but agreed within 12 mm. The region numbers in the plot correspond to region numbers in Table 5.7. Also note that P_A and P_B data did not exist near 200 mm. The sensors were not converging on any measurements at that distance.

Figure 5.22: PSS to Leica Comparison (October 2016, Test #1)

Table 5.8 provides the minimum, maximum, and average calculated percent errors with the standard deviations for each palm sensor.

Table 5.8: PSS Percent Errors at each Leica Distance (October 2016, Test #1)

Region	% Error (P_A)				% Error (P_B)				% Error (P_C)			
	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std
1	0.00	11.58	5.79	8.19	0.00	11.58	5.79	8.19	0.00	0.00	0.00	0.00
2	7.32	12.69	10.00	3.79	18.05	28.78	23.42	7.59	8.78	3.41	6.10	3.79
3	5.71	1.27	2.22	3.14	8.26	15.24	11.75	4.94	12.70	5.71	9.21	4.94
4	6.82	3.53	1.65	2.33	8.71	11.30	10.00	1.83	17.18	12.00	14.59	3.66
5	5.42	0.75	2.34	3.30	2.80	6.92	4.86	2.91	17.76	13.64	15.70	2.91
6	4.50	1.08	2.79	2.41	0.62	4.03	2.33	2.41	14.73	11.32	13.02	2.41
7	4.27	1.51	2.89	1.95	2.43	0.34	1.05	1.48	7.03	4.27	5.65	1.95
8	3.55	1.03	2.29	1.78	1.03	0.23	0.40	0.57	6.70	5.44	6.07	0.89
9	2.16	1.56	1.86	0.42	2.16	0.97	1.56	0.84	6.90	5.12	6.01	1.26
10	2.60	1.48	2.04	0.79	1.48	0.36	0.92	0.79	7.63	6.51	7.07	0.79
11	0.87	0.19	0.34	0.48	2.46	0.87	1.66	1.12	7.23	6.17	6.70	0.75
12	-	-	-	-	1.83	-	-	-	5.86	5.35	5.61	0.36

Figure 5.23 is a group of average percent error plots for each PSS range sensor. The region axis in the plot corresponds to the region numbers in both Table 5.8 and Table 5.7. Blue signals show the average sensor errors, and the shaded areas around them show the standard deviations at each Leica distance. The red dashed lines and shaded areas around them show the average sensor errors and standard deviations for the entire test. On average, each sensor had a low error, which is clearly labeled on the plots. All sensor performance was comparable, with larger deviations and errors closer to the Marman ring and smaller errors and deviations further away.

Figure 5.23: Average PSS Percent Errors at each Leica Distance (October 2016, Test #1)

5.4.2 Moto 50 Robot Kinematics

This test featured tracking CBSS proximity to the Marman in 10 mm increments out to 200 mm. Figure 5.24 shows how well the CBSS proximity sensors matched the Moto 50 data. It can be seen that all palm proximities (P_A , P_B , and P_C) were close to the Moto 50 data. As observed in the Leica test, P_C deviated more but was within 10 mm. Also, palm tool proximity sensors A and B stopped converging on distances around 200 mm, while sensor C remained online.

Figure 5.24: PSS to Robot Arm Comparison (October 2016, Test #2)

Figure 5.25 shows the selection process for extracting samples. The Moto 50 held each test distance for approximately 10 seconds. 26 samples (5.2 s of data) were taken from each boxed region, except at contact. This area was shorter, so 11 samples were extracted. A later plot references the 20 regions labeled in this plot.

Figure 5.25: Data Selection (October 2016, Test #2)

Figure 5.26 provides percent error plots for each proximity sensor, compared to the Moto 50 data, with calculated averages and standard deviations for the entire test overlaid. The red dashed lines and shaded red areas show those metrics for each sensor, and the blue signals represent the real time error. Mean errors, standard deviations, UBs, LBs, minimum errors, and maximum errors are clearly indicated in the plots. Each sensor had an impressively low average error again and followed the trends described in the previous PSS test. These trends are presented more clearly in the next set of plots.

Figure 5.26: PSS Percent Errors with Data Deletion (October 2016, Test #2)

Figure 5.27 is a group of plots showing the average error in each proximity sensor at every Moto 50 distance. The region number axis corresponds to the 20 regions in Figure 5.25, which indicate the distance conditions in the next table (Table 5.9). The plots show that the region errors are consistently near the calculated averages from Figure 5.26. The errors were not significantly greater at low proximities when compared to higher ones, but the deviations differed. The sensors revealed comparable performance analogous to the last test, with larger deviations when the CBSS was close to the Marman ring and smaller deviations further away.

Figure 5.27: Average PSS Percent Errors at each Arm Distance (October 2016, Test #2)

Table 5.9 provides the data the plots were built on. This table contains an in-depth analysis of the PSS percent error against every arm distance. The table shows the average percent errors and the standard deviations from those averages, along with the minimum and maximum percent errors for each proximity sensor.

Table 5.9: PSS Error Analysis at each Arm Distance (October 2016, Test #2)

Region	Moto 50 Distance (mm)	% Error (P _A)				% Error (P _B)				% Error (P _C)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
1	8.6	3.34	2.34	0.41	6.45	4.40	2.67	0.06	9.24	5.00	3.42	0.22	9.96
2	18.4	1.84	1.33	0.01	5.35	3.04	2.97	0.02	8.87	5.27	2.99	0.22	11.27
3	28.4	3.62	1.86	0.49	7.42	6.33	1.66	3.99	9.27	4.25	1.76	0.56	6.68
4	38.4	5.47	0.99	3.19	6.74	6.45	0.81	5.15	7.97	8.15	1.20	6.15	10.86
5	48.4	3.20	1.54	0.81	5.78	5.03	1.03	2.91	6.95	7.43	1.39	5.75	9.94
6	58.4	6.14	0.53	5.19	6.98	3.54	0.75	2.07	4.70	8.23	0.76	6.43	9.31
7	68.4	5.12	0.86	3.52	6.45	3.51	0.70	2.11	4.75	7.62	1.01	5.93	9.47
8	78.4	5.32	0.51	4.24	6.09	5.18	0.52	4.10	6.19	5.39	1.49	3.28	8.70
9	88.4	5.42	0.95	3.68	7.04	4.82	0.54	4.07	5.87	3.92	1.39	1.55	6.20
10	98.4	4.77	0.86	3.44	6.24	4.10	0.50	3.08	4.91	3.79	0.65	2.82	5.25
11	108.4	5.33	0.70	4.00	6.73	3.27	0.36	2.56	3.92	6.25	0.32	5.78	6.83
12	118.4	3.08	1.18	1.84	5.34	2.46	0.74	1.15	3.41	6.21	0.58	5.38	7.24
13	128.4	3.44	0.71	2.11	4.38	2.50	0.39	1.74	3.06	6.17	0.34	5.81	6.90
14	138.4	2.98	0.97	1.57	5.22	3.26	0.42	2.62	4.02	5.23	0.57	4.23	6.35
15	148.4	3.93	0.41	3.25	4.93	2.93	0.42	2.39	3.71	5.89	0.78	4.12	6.76
16	158.4	3.82	0.36	3.06	4.42	2.30	0.34	1.94	3.08	5.56	0.23	5.22	5.98
17	168.4	4.02	0.67	2.69	4.97	2.60	0.32	2.02	3.18	5.64	0.64	4.67	6.74
18	178.4	2.75	0.67	1.34	3.70	3.43	0.34	2.62	3.92	5.57	0.42	4.71	6.13
19	188.4	2.33	0.39	1.57	2.75	1.44	0.28	0.84	1.90	5.60	0.74	3.87	6.18
20	198.4	-	-	-	-	-	-	-	-	3.21	0.26	2.61	3.63

5.5 FSS Test Plan, Round One Testing

Four experiments were performed from the FSS test plan that was defined in Section 4.8. The test plan was designed to fully characterize the FSS, but only a subset of those tests could be completed. Section 4.8 indicates the tests that were actually completed, and this subset of tests was dubbed “Round One Testing”. Ensuing plots and analyses in this section follow the same conventions established in the first two FSS test campaigns. FSS Global force, x-axis torque, and y-axis torque are all still compared to the JR3, and errors are shown with averages, standard deviations, UBs, LBs, minimum errors, maximum errors, the same color schemes, etc. For that reason, this will not be stated for every plot. If there is any confusion about how plots convey information, refer to Section 5.2 and Section 5.3 for definitions. Only major findings or trends will be discussed.

Experimental setups were also the same as previous FSS tests. Refer to Section 4.4.2 – Section 4.4.4 for the generic setup/overview and Section 4.8 for detailed procedures and test matrices. Palm tools were again within 10 mm of coplanar and within 5 mm of a common radius. Figure 5.28 provides a reminder of the contact scenarios from the test plan. The four conditions experimentally tested were nominal contact, a two rail contact across two palm tools at a 5° misalignment, a two rail contact on a single palm tool at a 5° misalignment, and single rail contact on one palm tool at a 5° misalignment.

Figure 5.28: Contact Scenarios

5.5.1 Nominal Contact

Figure 5.29 shows the FSS response to the nominal contact tests in Table 4.5. It can be seen that palm tools A, B, and C made contact around 49 seconds. F_A and F_B were nearly identical; F_C showed about a 30 N difference, revealing minor misalignment. Since each palm tool force was similar, near nominal contact was achieved. F_G was the global force, which was the sum of the three local forces.

Figure 5.29: FSS Verification of Nominal Contact

Visual inspection of Figure 5.30 shows the differences in the global force to be approximately 0-5 N. Differences in the x- and y-torques both appear to be 0-2 Nm. The plots indicate the FSS performed well in this scenario like in previous test campaigns.

Figure 5.30: FSS to JR3 Comparison (October 2016, Nominal Contact)

Figure 5.31 shows where data was extracted. Each Table 4.5 force was maintained for 10-15 seconds, and the centers of the data for each test case, disregarding any abnormal spikes, were found. 80% of the data, based on the shortest usable time period, was extracted for the global force and x-y torque datasets. This resulted in 9.6 seconds (48 samples) of usable CBSS and JR3 data from each boxed region. Regions labeled “1” through “15” correspond to tests “1” to “15” in Table 4.5.

Figure 5.31: Data Selection (October 2016, Nominal Contact)

Figure 5.32 does not reveal anything out of the ordinary. Errors trends were pretty typical based on previous test campaigns. The y-axis torque was alarmingly high in some tests though for reasons stated in previous nominal contact testing. This is further explored in the next plots.

Figure 5.32: FSS Percent Errors with Data Deletion (October 2016, Nominal Contact)

High torque errors were expected because of overall low resultant torques in nominal contact and the nature of percent error at small values. Figure 5.33 shows the differences between the CBSS and JR3 global force coordinates. It can be seen in the plots that the maximum difference in the global force's x- and y-coordinates was about 3.5 cm (1.38 in) and 5.5 cm (2.17 in). The differences in the y-coordinates, where percent torque errors were larger than 100%, were less than 2 cm.

Figure 5.33: Differences in FSS and JR3 Global Force Locations (October 2016, Nominal Contact)

Figure 5.34 shows if outliers are ignored, the average y-torque error is actually $16.97\% \pm 14.42\%$ over the duration of the experiment. This is much better than before.

Figure 5.34: FSS Y-Axis Torque Error Ignoring Outliers (October 2016, Nominal Contact)

Figure 5.35 shows the average errors and standard deviations for every boxed region of data extracted in Figure 5.31. Regions 2, 4, 8, 12, and 15 have an applied force of 20 N; regions 3, 5, 7, 10, and 14 have an applied force of 100 N; and regions 1, 6, 9, 11, and 13 have an applied force of 200 N. As in all other FSS testing, standard deviations were very low and can barely be visualized. The errors at each applied force were comparable too, with the exception of the y-torque at 100 N. However, this was already explored.

Figure 5.35: Average FSS Percent Error per Region (October 2016, Nominal Contact)

Table 5.10 provides the error analysis for every test in Table 4.5. The test column denotes the particular test in the plan. They are out of order in this table to organize the data by the 20 N, 100 N, and 200 N force levels; it was easiest to show repeatability when organized this way. Table 5.10's "Std" column confirms the low standard deviations seen before, indicating load cell precision. Regarding repeatability, the table shows that the global force and x-torque errors were repeatable within a few percent of the average, with the y-torque showing larger deviations because of the percent error caveat with small values.

Table 5.10: Test Plan Error Analysis for Nominal Contact

	Test	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
20 N	2	4.24	0.37	3.33	5.03	1.41	0.68	0.05	2.81	12.80	0.40	11.65	13.52
	4	7.70	0.36	6.97	8.38	2.58	0.32	1.92	3.24	24.78	0.39	24.08	25.76
	8	10.26	0.34	9.67	10.87	3.22	0.50	2.14	4.31	37.52	0.30	37.02	38.16
	12	9.69	0.41	8.94	10.69	1.73	0.52	0.89	2.75	34.36	0.57	33.40	35.39
	15	13.32	0.20	12.84	13.71	4.19	0.50	3.26	5.15	47.26	0.32	46.72	48.02
	Ave	9.04	0.34	8.35	9.74	2.63	0.50	1.65	3.65	31.34	0.40	30.57	32.17
	Std	3.36	0.08	3.51	3.24	1.13	0.13	1.23	1.05	13.11	0.11	13.33	13.10
100 N	3	0.43	0.06	0.30	0.54	5.29	0.32	4.69	5.87	10.48	0.88	8.41	12.03
	5	0.62	0.08	0.44	0.77	0.45	0.27	0.03	1.04	4.94	0.86	3.20	6.49
	7	3.23	0.06	3.10	3.35	5.88	0.23	5.40	6.40	178.08	7.69	167.99	194.91
	10	3.51	0.06	3.43	3.63	6.44	0.17	6.11	6.86	308.21	18.07	265.99	346.36
	14	3.05	0.07	2.89	3.19	5.92	0.16	5.54	6.28	237.39	11.12	216.38	258.02
	Ave	2.17	0.07	2.03	2.30	4.80	0.23	4.36	5.29	147.82	7.73	132.39	163.56
	Std	1.51	0.01	1.53	1.51	2.46	0.07	2.47	2.40	135.96	7.29	120.66	150.79
200 N	1	2.39	0.03	2.34	2.47	5.76	0.12	5.52	6.03	3.35	0.13	3.01	3.60
	6	1.40	0.03	1.31	1.44	7.88	0.08	7.71	8.05	7.01	0.12	6.84	7.21
	9	2.14	0.03	2.08	2.21	8.16	0.08	8.03	8.32	5.60	0.17	5.35	5.93
	11	1.57	0.03	1.50	1.63	8.23	0.08	8.00	8.37	7.63	0.15	7.34	7.89
	13	1.06	0.04	0.93	1.15	7.88	0.07	7.72	8.00	8.13	0.27	7.71	8.74
	Ave	1.71	0.03	1.63	1.78	7.58	0.09	7.40	7.75	6.34	0.17	6.05	6.67
	Std	0.55	0.006	0.57	0.55	1.03	0.02	1.06	0.98	1.92	0.06	1.92	2.00

5.5.2 Misalignment One – Two Palm, Two Rail Contact

Figure 5.36 shows the FSS forces during the first misalignment scenario in Table 4.6. This misalignment configuration allowed the CBSS to only contact palm tools A and B. The plot shows that F_A and F_B was nonzero, while F_C was nonexistent, indicating that two rail contact across two palm tools did occur. F_G was the global force, and was the sum of the other three plot forces.

Figure 5.36: FSS Verification of Misalignment One

Figure 5.37 shows that the FSS did not do as well monitoring the global force and x-y torques for this particular configuration. Differences in the global force visually appear to be 2-30 N, while differences in the x- and y-torque appear to be 2-20 Nm and 2-8 Nm, respectively. This configuration marked the worst case of FSS performance, compared to the JR3.

Figure 5.37: FSS to JR3 Comparison (October 2016, Misalignment One)

The data selection process remained unchanged from the one described in Figure 5.31. 10.4 seconds (52 samples) of usable CBSS and JR3 data was extracted from each boxed region in Figure 5.38. Regions labeled “1” through “9” correspond to tests “16” to “24” in Table 4.6.

Figure 5.38: Data Selection (October 2016, Misalignment One)

Figure 5.39 shows that the FSS errors in the global force, x-axis torque, and y-axis torque were, on average, higher than other tests already presented or even those yet to be discussed. This two rail contact across two palm tools pushed sideways on the load cells more so than any other configuration due to the shape of the Marman ring and the large distance between contact points. Recalling that the FSS load cells are designed to be loaded on the z-axis and are insensitive to lateral forces, this explains why errors were much higher here.

Figure 5.39: FSS Percent Errors with Data Deletion (October 2016, Misalignment One)

Figure 5.40 shows average errors and standard deviations for every boxed region of data extracted in Figure 5.38. Referencing Table 4.6, plot regions 2, 4, and 8 correspond to tests 17, 19, and 23 and have an applied force of 20 N; regions 3, 6, and 9 correspond to tests 18, 21, and 24 and have an applied force of 100 N; and regions 1, 5, and 7 correspond to tests 16, 20, and 22 and have an applied force of 200 N. Region errors at identical applied forces were larger than normal but were consistent. Following the same trends previously observed, the standard deviations were very low, meaning the load cells still displayed high precision in this scenario.

Figure 5.40: Average FSS Percent Error per Region (October 2016, Misalignment One)

Table 5.11 provides the error analysis for every test performed in Table 4.6. The standard deviation columns numerically verify how low they actually were. Regarding repeatability values for each metric of interest, they were a little higher this time around but often within 3%-8%. The repeatability of the errors at larger forces was much lower than the 20 N and 100 N cases. Note the locations where the “Ave” columns intersect the “Std” row at 200 N. The repeatability of the average errors here were a fraction of one percent.

Table 5.11: Test Plan Error Analysis for Misalignment One

	Test	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
20 N	17	12.14	0.39	11.44	13.08	26.59	0.22	26.11	26.99	55.23	0.17	54.88	55.53
	19	9.72	0.75	8.56	11.93	26.72	0.45	25.47	27.38	48.30	0.17	48.07	49.06
	23	15.37	0.83	13.99	18.20	19.20	1.04	16.74	20.48	58.40	0.97	57.18	60.38
	Ave	12.41	0.66	11.33	14.40	24.17	0.57	22.77	24.95	53.97	0.44	53.38	54.99
	Std	2.83	0.24	2.72	3.34	4.30	0.42	5.23	3.88	5.17	0.46	4.73	5.68
100 N	18	21.56	0.16	21.34	21.94	39.25	0.19	38.99	39.72	27.61	0.07	27.51	27.75
	21	12.14	0.11	11.94	12.36	24.53	0.12	24.23	24.72	19.72	0.06	19.57	19.86
	24	19.84	0.17	19.60	20.28	36.86	0.16	36.65	37.26	29.96	0.11	29.85	30.27
	Ave	17.85	0.15	17.62	18.19	33.55	0.16	33.29	33.90	25.76	0.08	25.64	25.96
	Std	5.02	0.03	5.00	5.12	7.90	0.04	7.93	8.04	5.37	0.03	5.38	5.43
200 N	16	14.02	0.08	13.93	14.28	30.53	0.13	30.36	30.83	11.29	0.06	11.16	11.41
	20	14.03	0.04	13.95	14.13	30.78	0.13	30.58	31.03	10.75	0.08	10.54	10.87
	22	13.96	0.04	13.86	14.03	30.77	0.07	30.68	30.91	10.61	0.03	10.53	10.67
	Ave	14.01	0.05	13.91	14.15	30.69	0.11	30.54	30.92	10.88	0.06	10.75	10.98
	Std	0.04	0.02	0.05	0.13	0.14	0.04	0.16	0.10	0.36	0.02	0.36	0.38

5.5.3 Misalignment Two – Single Palm, Two Rail Contact

Figure 5.41 provides a verification of misalignment two, and shows that only palm tool C registered a force. F_A and F_B were both at 0 N, with F_B covering F_A , thus red was only seen. F_C shows nonzero force, and since F_G is the sum of the local forces, $F_C = F_G$. Because F_G was plotted first, F_C covered it, and green was the only color seen in the plot. Regardless of the visible colors, MATLAB did plot every local force and the global force, indicated by the legend. These forces provided evidence that single palm contact did occur, but distinguishing two-rail contact from single-rail contact was impossible by looking at these forces alone. This particular case is a two rail contact, but one would have to look at the individual load cell forces to tell a difference.

Figure 5.41 FSS Verification of Misalignment Two

Figure 5.42 shows that the differences in the global force, x-torque, and y-torque visually appear to be 0-5 N, 2-6 Nm, and 2-6 Nm, respectively. FSS performance was much better in this misalignment than the previous one.

Figure 5.42: FSS to JR3 Comparison (October 2016, Misalignment Two)

The data selection process remained the same. 9.2 seconds (46 samples) of usable CBSS and JR3 data was extracted from each boxed region in Figure 5.43. Regions labeled “1” through “9” correspond to tests “43” to “51” in Table 4.7.

Figure 5.43: Data Selection (October 2016, Misalignment Two)

Figure 5.44 shows the average errors were pretty low for every metric of interest, indicating good performance for this misalignment. The error in the force was very small ($< \sim 2\%$), and the x- and y-torques were typically under 20%. No performance expectations were ever specifically stated because this entire project was proof of concept of COTS systems. Experimentation was more exploratory in nature. However, the student considered results desirable when the FSS global force error was less than 5%, and the torque errors were below 20%.

Figure 5.44: FSS Percent Errors with Data Deletion (October 2016, Misalignment Two)

Figure 5.45 shows average errors and standard deviations for every boxed region of data extracted in Figure 5.43. Referencing Table 4.7, plot regions 2, 4, and 9 correspond to tests 44, 46, and 51 and have an applied force of 20 N; regions 3, 6, and 8 correspond to tests 45, 48, and 50 and have an applied force of 100 N; and regions 1, 5, and 7 correspond to tests 43, 47 and 49 and have an applied force of 200 N. Force errors were low, and torque errors were only slightly high at 20 N. Of course, the standard deviations were low too. The red shaded areas indicating the deviations were not visible, except in the top plot because of scale. Visual inspection also revealed that repeatability was very good. Errors from repeated contact were comparable.

Figure 5.45: Average FSS Percent Error per Region (October 2016, Misalignment Two)

Table 5.12 provides the error analysis for every test performed in Table 4.7 and shows the low standard deviations. Regarding the repeatability values for each metric, they were outstanding, whether the averages errors were large or miniscule. Repeatability values were under 0.5% with the exception of a few. The x- and y-torque repeatability values at 100 N were 1.57% and 1.17%, but that was still acceptable.

Table 5.12: Test Plan Error Analysis for Misalignment Two

	Test	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
20 N	44	0.91	0.21	0.50	1.30	22.94	0.10	22.75	23.10	28.53	0.08	28.36	28.72
	46	1.41	0.32	0.84	2.27	22.96	0.11	22.73	23.17	28.62	0.13	28.26	28.85
	51	0.53	0.29	0.03	1.09	23.46	0.10	23.26	23.66	29.61	0.10	29.40	29.81
	Ave	0.95	0.27	0.46	1.55	23.12	0.10	22.91	23.31	28.92	0.11	28.67	29.13
	Std	0.36	0.04	0.33	0.51	0.24	0.004	0.24	0.25	0.49	0.02	0.51	0.49
100 N	45	0.74	0.06	0.59	0.88	7.59	0.03	7.50	7.66	13.32	0.04	13.20	13.41
	48	1.47	0.05	1.37	1.64	4.72	0.03	4.67	4.78	11.09	0.04	11.02	11.16
	50	1.92	0.06	1.79	2.06	3.95	0.02	3.88	4.00	10.65	0.07	10.49	10.77
	Ave	1.37	0.06	1.25	1.53	5.42	0.03	5.35	5.48	11.69	0.05	11.57	11.78
	Std	0.49	0.003	0.50	0.49	1.57	0.004	1.56	1.57	1.17	0.01	1.17	1.16
200 N	43	1.85	0.03	1.80	1.91	5.06	0.07	4.94	5.21	10.39	0.04	10.33	10.50
	47	1.53	0.03	1.47	1.59	4.76	0.03	4.72	4.81	10.40	0.03	10.34	10.47
	49	1.19	0.03	1.14	1.25	3.98	0.01	3.95	4.00	9.87	0.02	9.83	9.91
	Ave	1.52	0.03	1.47	1.58	4.60	0.04	4.53	4.67	10.22	0.03	10.17	10.29
	Std	0.27	0.001	0.27	0.27	0.46	0.03	0.42	0.50	0.25	0.008	0.24	0.27

5.5.4 Misalignment Three – Single Palm, Single Rail Contact

Figure 5.46 provides a verification of misalignment three and shows that only palm tool C registered a force. This plot is nearly identical to Figure 5.41; however, it shows the single-rail contact forces from Table 4.8 in the test plan. The forces in the plot provided evidence that single palm contact did occur, but individual load cell forces would need analyzed to tell a difference between single-rail contact and two-rail contact.

Figure 5.46 FSS Verification of Misalignment Three

Figure 5.47 shows that the differences in the global force, x-torque, and y-torque were 0-1 N, 0-2 Nm, and 6-12 Nm, respectively. The FSS y-axis torque was furthest from the JR3 in this scenario and must be characteristic of this misalignment. More testing is required to know if the y-axis torque is always off like this. All other plot categories were highly accurate when compared to the JR3.

Figure 5.47: FSS to JR3 Comparison (October 2016, Misalignment Three)

The data selection process remained the same. 11.2 seconds (56 samples) of usable CBSS and JR3 data was extracted from each boxed region in Figure 5.48. Regions labeled “1” through “9” correspond to tests “70” to “78” in Table 4.8.

Figure 5.48: Data Selection (October 2016, Misalignment Three)

Figure 5.49 shows that the errors for the global force and x-axis torque were very low (less than ~3%), with averages being even lower near ~1%. However, the y-torque error was fairly high (around 44%) at larger applied forces. More testing would be required to know if this is just a characteristic of this misalignment.

Figure 5.49: FSS Percent Errors with Data Deletion (October 2016, Misalignment Three)

Figure 5.50 shows average errors and standard deviations for every boxed region of data extracted in Figure 5.48. Referencing Table 4.8, plot regions 2, 4, and 8 correspond to tests 71, 73, and 77 and have an applied force of 20 N; regions 3, 6, and 9 correspond to tests 72, 75, and 78 and have an applied force of 100 N; and regions 1, 5, and 7 correspond to tests 70, 74 and 76 and have an applied force of 200 N. Once again, the standard deviations for every error for each metric were all very low. The errors at each force level appeared repeatable as well, for the most part.

Figure 5.50 Average FSS Percent Error per Region (October 2016, Misalignment Three)

Table 5.12 provides the error analysis for every test performed in Table 4.8 and shows the low standard deviations in the “Std” columns, indicating the precision of the load cells. Repeatability values were under 0.5% with the exception of a few. The x- and y-torque repeatability values at 100 N were 1.55% and 1.33%, but that was still acceptable.

Table 5.13: Test Plan Error Analysis for Misalignment Three

	Test	% Error (F_G)				% Error (T_X)				% Error (T_Y)			
		Ave	Std	Min	Max	Ave	Std	Min	Max	Ave	Std	Min	Max
20 N	71	1.97	0.40	1.30	2.77	1.99	0.15	1.72	2.27	43.81	0.18	43.48	44.22
	73	2.12	0.39	1.37	2.84	2.16	0.15	1.68	2.47	43.24	0.13	42.78	43.47
	77	1.49	0.38	0.75	2.28	1.32	0.14	1.01	1.57	43.64	0.14	43.36	43.89
	Ave	1.86	0.39	1.14	2.63	1.82	0.14	1.47	2.10	43.56	0.15	43.21	43.86
	Std	0.33	0.007	0.34	0.31	0.44	0.009	0.40	0.47	0.30	0.02	0.37	0.37
100 N	72	0.50	0.13	0.29	0.73	0.06	0.04	0.003	0.16	20.68	0.03	20.61	20.74
	75	1.42	0.07	1.30	1.60	2.96	0.05	2.82	3.09	18.14	0.03	18.08	18.21
	78	0.73	0.06	0.58	0.82	0.57	0.05	0.46	0.67	20.08	0.03	20.02	20.14
	Ave	0.88	0.09	0.72	1.05	1.20	0.05	1.10	1.31	19.64	0.03	19.57	19.70
	Std	0.48	0.04	0.52	0.48	1.55	0.006	1.51	1.56	1.33	0.003	1.32	1.32
200 N	70	0.19	0.04	0.10	0.34	0.61	0.09	0.04	0.68	13.96	0.07	13.91	14.42
	74	0.18	0.03	0.10	0.26	0.72	0.03	0.67	0.78	13.94	0.02	13.91	14.00
	76	0.28	0.03	0.20	0.33	0.73	0.02	0.69	0.76	13.81	0.03	13.76	13.86
	Ave	0.22	0.03	0.14	0.31	0.68	0.04	0.47	0.74	13.90	0.04	13.86	14.09
	Std	0.06	0.008	0.06	0.05	0.06	0.04	0.37	0.05	0.08	0.03	0.08	0.29

5.6 Load Cell Drift from Zero Force

Another part of the test plan was to determine if the load cells returned to 0 N after experiments were performed. 10 seconds of load cell data before and after every experiment was analyzed. The forces over those 10 second intervals were averaged for each load cell to yield the typical force readings. Average post-experimental load cell forces were subtracted from their average initial forces to provide the drift of each load cell. Table 5.14 shows the resulting data. Columns A1 to C4 denote the 12 load cells in the palm tools. The total drift was the sum of the absolute values of individual load cell drifts and was featured in the rightmost column; averages are highlighted in yellow where applicable. The total FSS drift was negligible ($< \sim 1$ N), with the exception of the experiment featuring a two-rail contact across two palm tools. This test had a 3.64 N drift and the poorest overall results among all the experiments.

Table 5.14: Load Cell Zero Drift and Total Drift for all FSS Tests

		Load Cell Drift from Zero (N)												Total Drift (N)
		A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4	
June 2016		-0.02	-0.04	0.12	0.18	0.11	-0.01	-0.06	0.00	0.04	-0.08	-0.04	0.14	0.84
		-0.02	-0.02	0.02	0.04	0.04	0.00	-0.01	-0.01	0.02	-0.03	-0.01	0.02	0.24
		-0.02	-0.01	0.02	0.02	0.04	0.00	0.00	-0.01	0.02	-0.02	-0.01	0.02	0.19
		-0.02	-0.02	0.05	0.08	0.06	0.00	-0.02	-0.01	0.02	-0.04	-0.02	0.06	0.42
August 2016		0.02	-0.01	0.42	0.39	0.03	0.01	-0.02	-0.01	0.02	0.09	0.00	0.01	1.03
		0.00	-0.01	0.04	0.02	0.02	0.00	-0.01	-0.01	0.02	0.00	0.00	0.01	0.14
		0.02	-0.02	0.26	0.57	0.02	0.00	-0.02	-0.01	0.03	-0.01	0.00	0.01	0.97
		0.00	-0.02	0.04	0.01	0.02	0.00	-0.02	-0.01	0.02	-0.01	0.00	0.01	0.16
		0.00	-0.01	0.02	-0.01	0.02	0.00	-0.02	-0.01	0.02	-0.03	0.00	0.01	0.15
		0.00	-0.01	0.03	0.01	0.01	0.00	-0.01	-0.01	0.01	-0.01	0.00	0.01	0.11
		0.00	-0.01	0.03	0.00	0.02	0.00	-0.01	0.00	0.02	-0.02	0.00	0.01	0.12
		0.00	-0.01	0.02	0.00	0.01	0.00	-0.01	0.00	0.00	-0.01	0.00	0.01	0.07
		0.00	-0.02	0.03	0.00	0.02	0.00	-0.02	-0.02	0.03	-0.03	0.00	0.01	0.18
		0.00	-0.01	0.02	-0.01	0.01	0.00	-0.02	-0.01	0.01	-0.02	0.00	0.01	0.12
		0.00	-0.02	0.03	-0.01	0.02	0.00	-0.02	-0.01	0.02	-0.02	0.00	0.01	0.16
		0.00	-0.01	0.09	0.09	0.02	0.00	-0.02	-0.01	0.02	-0.01	0.00	0.01	0.29
October 2016		-0.01	-0.01	0.00	0.01	0.02	0.01	0.00	0.01	0.02	-0.03	0.00	0.00	0.11
		0.76	0.72	0.00	-0.05	-0.01	-0.01	0.97	1.12	0.00	0.00	0.00	0.00	3.64
		-0.03	-0.02	0.00	-0.01	0.00	0.00	-0.07	-0.13	-0.02	0.21	0.04	0.02	0.55
		0.00	-0.01	0.00	0.00	0.00	0.00	-0.03	-0.04	0.00	-0.01	0.15	-0.02	0.26

CHAPTER 6 | CONCLUSION

6.1 Summary

The goal of this research was to investigate solutions applicable to the CBS for proximity and force sensing capabilities. NASA presently uses binary switches that are triggered by “flapper” attachments connected to CBS palm tool sides. The flappers only allow the presence of a Marman ring to be detected very close to the CBS palm tools (≤ 18 mm), which is enough to capture a ring, but no further information is gained from the system. This project completed the design and implementation of two sensory systems, dubbed the pose sensing system (PSS) and force sensing system (FSS). This work extended the Marman ring detection distance to approximately 200 mm and allowed ring pose to be calculated. This work also allowed ring-to-CBS contact forces to be monitored, which provided knowledge of initial ring contact, ring clampdown/capture, and forces in the “docked” state during servicing operations. GUI and logging software was also developed and PSS/FSS performance was evaluated.

The PSS consisted of small form factor time of flight range sensors secured to attachments designed to be friction fitted into the open palm cavities on the undersides of each tool. The sensor system determined each palm tool’s proximity to a Marman. Testing revealed, on average, the palm tool proximity sensors were in error of the true distance by 5% at near nominal approaches. Tests also showed that the maximum sensing distance was near 200 mm.

The FSS consisted of thin beam load cells, sensor housings, and cover rails designed as side attachments for each palm tool. The cover rails rested just above the top palm surfaces of each palm tool and allowed determination of initial contact with a Marman ring. The system was used to track the magnitude and location of resultant forces on each palm tool, in addition to the magnitude and location of the global force on the whole WVRTC CBSS. Although the scope of this research did not involve testing the clampdown of the Marman ring in the palm tool jaws, the FSS is capable of determining when that occurs and how much force is applied. Transfer loads that may occur as robotic arms service a satellite docked with the real CBS can also be monitored with this system.

The FSS was fully functional long before the PSS, so it was tested more rigorously. Testing of various contact configurations (with different contact forces) revealed, on average, that the errors of the true global force were $\leq 15\%$. Additionally the FSS was in error of the true x-y torques by $\leq 30\%$. The two-palm tool, two-rail contact configuration was the test that yielded a 15% global force error and 30% torque errors. That configuration may have pushed the load cells sideways more so than the other tests. Deviating from the desired loading axis of the contacted force sensors likely contributed to the larger errors. The force and torque errors, not including that specific test, were typically under 5% and 20%, respectively. It should also be noted that hysteresis effects and load cell drift were determined negligible over the experiments.

The results of this research show how effective systems like the PSS and FSS can be and reinforce the need for inclusion on the real CBS. It is reasonable to say that data generated by these systems can provide teleoperators, or even autonomous algorithms, with invaluable state information that would decrease the likelihood of catastrophic failures or damaging components during docking. The range sensors could be used to orient two spacecraft at an optimal alignment when in close proximity and provide the necessary information to slow down the docking process as the Marman ring is drawn closer to the CBS, thereby reducing high-impact, initial contact forces. The force sensors would then provide knowledge of absolute contact to trigger the capture phase in addition to verifying Marman ring clampdown and monitoring the properties of sustained contact while docked.

6.2 Future Work

If the experiments were to be carried out again it would be beneficial to completely remove the adjustable height posts in the CBSS design. They only add a source of error to the results. Height adjustability was facilitated through set screws that held the palm tool mounting plates to the CBSS posts through friction. It was nearly impossible to get the CBSS ring contact points coplanar, and applying heavy loads could potentially cause the set screws to slip. The palm tools could be mounted directly to the aluminum t-slotted extrusion bars to increase the chance of coplanar contact surfaces among the RSS modules. A relatively thick aluminum plate faced on a CNC mill to keep low tolerances is all that is necessary and could be attached to the CBSS arms

directly with some t-slot hardware. The palm tools could then be attached to this plate for testing. Metal RSS modules and palm tools could also be machined. All of this would eliminate unquantifiable errors in the palm tools due to 3D printed parts. However, it was observed during testing that the aluminum extrusion bars (CBSS arms) flexed noticeably with load. The CBSS could be mounted flat on a wall or to a table. The FSAB Marman ring could be driven into the CBSS with a robotic arm equipped with a JR3 force-torque sensor. Another potential improvement to the test method may be to mount each palm tool on a JR3 sensor on a table top. Assuming the three-palm tool, three-JR3 method could be configured into a precise CBSS orientation on the table top, the FSAB could be driven into that setup, alternatively. This test configuration would allow not only global CBSS force-torque characteristics to be validated but also the local palm tool forces and torques.

Another note regarding the experiments is to always use the lightweight C++ logger that has minimal overhead because it is consistent with sampling the data at a constant rate. Regarding post-processing of any experimental results, the effects of hysteresis could be reduced by using average calibration load lines instead of the method used in Section 5.1.1. Data exists for every load cell for increasing the applied force from 0 lb to the rated capacity in 10 lb increments and then for removing the applied force in 10 lb increments. Calibration lines for loading and unloading could be found and averaged together for a better MATLAB model. This is not an imminent issue but delivering better data is always something to strive for. Another post-processing improvement to consider is better JR3 force models. FSS Omega load cells have one DOF, but the JR3 is a six-DOF sensor, yielding x-, y-, and z-force as well as x-, y-, and z-torque. The only JR3 data used to validate FSS load cells was the z-force, x-torque, and y-torque. In the various contact scenarios, perhaps higher complexity JR3 force models utilizing all six degrees of freedom would produce more accurate experimental results.

The range sensors must be tested more rigorously to determine experimental repeatability and other characteristics. The sensors are supposed to be resistant to ambient light while determining proximity. Testing them in variable lighting conditions would be an interesting group of tests. In space, the sensors may be exposed to direct sunlight at times, and it would be beneficial to know if they are even viable in extreme lighting conditions. The sensors also have a

25° field of view, so various CBSS-to-Marman ring angular misalignments should be experimented with to determine if or when the sensors become ineffective.

The FSS test plan also needs completed to collect data on contact configurations with larger misalignment angles in order to wrap up proof of concept testing. After that phase of testing is over, it is essential to test on real CBS equipment or on a precision machined replica. Sensor system plastic components must be swapped with precision metal counterparts. The full CBS mechanical platform should be replicated to enable Marman ring clampdown. The sensor systems could then be characterized on a rigid device under various approach and contact scenarios for every phase of the docking procedure. Testing on a precision replica would allow the sensor systems to truly be characterized with utmost confidence, removing current sources of error such as prototype flexing.

With most of the proof of concept testing finished, it is now time to push the sensor systems to flight ready solutions. Extensive research on proper cabling, protecting the sensors, shielding electronics, and surviving space conditions must be performed, as well as exploring solutions for embedding sampling hardware inside CBS components. It would also be beneficial to develop a custom electronics board that combines the hardware for sampling the force sensors with a microcontroller for sampling the range sensors. Concatenating a bunch of evaluation boards is fine for prototyping, but a more refined, compact solution should be found.

A recent look at the sensor documentation revealed that there is more content added now than what was available when the microcontroller code was developed. It would be beneficial to revisit some code solutions regarding the sensor API provided by the manufacturer. The algorithm developed for this research was enough to read the sensors, but the manufacturer's API facilitates easier configuration of the sensor. It has been discussed that range sensors would need protected with a radiation safe cover glass. The API has functionality to calibrate the sensors with a cover glass. Applying the API would decrease the time required to integrate sensors in a custom application.

REFERENCES

- [1] NASA Goddard, "Restore-L," [Online]. Available: <https://sspd.gsfc.nasa.gov/restore-l.html>. [Accessed 28 November 2017].
- [2] "On-Orbit Satellite Servicing Study," Project Report, NASA Goddard Space Flight Center, 2010.
- [3] R. Ticker, "Restore-L Mission Information, Package for NASA Solicitation #NNH15HEOMD001," NASA Goddard Space Flight Center, 2010.
- [4] J. B. Battin, "DESIGN OF A FORCE SENSING SYSTEM TO ASSIST ROBOTIC SPACE SERVICING AND EXPLORATION OPERATIONS," M.S. Thesis, West Virginia University, 2015.
- [5] J. Cook, V. Aksamentov, T. Hoffman and W. Bruner, "ISS Interface Mechanisms and their Heritage," [Online]. Available: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20110010964.pdf>. [Accessed 2017 28 November].
- [6] "Marman Clamp System Design Guidelines," Guideline No. GD-ED-2214, NASA Goddard Space Flight Center, 2000.
- [7] West Virginia Robotic Technology Center, "WVRTC," [Online]. Available: <http://wvrtc.com/>. [Accessed 28 November 2017].
- [8] A. M. Long, M. G. Richards and D. E. Hastings, "On-Orbit Servicing: A New Value Proposition for Satellite Design and Operation," *Journal of Spacecraft and Rockets*, vol. 44, no. 4, 2007.
- [9] NP-2016-2-392-GSFC, "Restore-L, Proving Satellite Servicing," NASA Goddard Space Flight Center, 2016.
- [10] STMicroelectronics, "Proximity Sensors," [Online]. Available: <http://www.st.com/en/imaging-and-photonics-solutions/proximity-sensors.html?querycriteria=productId=SC1934>. [Accessed 26 November 2017].
- [11] STMicroelectronics, "VL6180X," [Online]. Available: http://www.st.com/content/st_com/en/products/imaging-and-photonics-solutions/proximity-sensors/vl6180x.html. [Accessed 26 November 2017].
- [12] STMicroelectronics, "Time of Flight: Principles, Challenges, and Performance," [Online]. Available: http://www.st.com/content/dam/technology-tour-2017/session-1_track-4_time-of-flight-technology.pdf. [Accessed 28 November 2017].

- [13] STMicroelectronics, "Datasheet: VL6180X Proximity and ambient light sensing (ALS) module," [Online]. Available:
<http://www.st.com/content/ccc/resource/technical/document/datasheet/c4/11/28/86/e6/26/44/b3/DM00112632.pdf/files/DM00112632.pdf/jcr:content/translations/en.DM00112632.pdf>.
[Accessed 26 November 2017].
- [14] STMicroelectronics, "UM1983 User manual: VL6180X proximity, gesture and ambient light sensing (ALS) module," [Online]. Available:
http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/ba/47/75/5e/d2/80/47/7e/DM00247018/files/DM00247018.pdf/jcr:content/translations/en.DM00247018.pdf. [Accessed 26 November 2017].
- [15] H.-W. Huang, HCS12/9S12: An Introduction to Software and Hardware Interfacing, 2nd ed., Clifton Park, New York: Delmar, Cengage Learning, 2010, pp. 530-541.
- [16] STMicroelectronics, "STM32L053R8: Ultra-low-power ARM Cortex-M0+ MCU with 64 Kbytes Flash, 32 MHz CPU, USB, LCD," [Online]. Available:
http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l0-series/stm32l0x3/stm32l053r8.html. [Accessed 26 November 2017].
- [17] STMicroelectronics, "NUCLEO-L053R8: STM32 Nucleo-64 development board with STM32L053R8 MCU, supports Arduino and ST morpho connectivity," [Online]. Available:
http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-l053r8.html.
[Accessed 26 November 2017].
- [18] Arm Limited, "Mbed OS," [Online]. Available: <https://www.mbed.com/en/platform/mbed-os/>. [Accessed 26 November 2017].
- [19] Arm Limited/MBED, "NUCLEO-L053R8," [Online]. Available:
<https://os.mbed.com/platforms/ST-Nucleo-L053R8/>. [Accessed 26 November 2017].
- [20] STMicroelectronics, "Data Brief: VL6180X-SATEL Satellite boards based on VL6180X proximity, gesture and ambient light sensor," [Online]. Available:
http://www.st.com/content/ccc/resource/technical/document/data_brief/dc/9a/25/6c/bd/12/4a/08/DM00150183.pdf/files/DM00150183.pdf/jcr:content/translations/en.DM00150183.pdf.
[Accessed 26 November 2017].
- [21] STMicroelectronics, "UM1852 User Manual: X-NUCLEO-6180XA1 proximity and ambient light sensor expansion board based on VL6180X for STM32 Nucleo," [Online]. Available:
http://www.st.com/content/ccc/resource/technical/document/user_manual/5a/d3/5b/bd/14/82/46/1f/DM00154104.pdf/files/DM00154104.pdf/jcr:content/translations/en.DM00154104.pdf.
[Accessed 26 November 2017].

- [22] STMicroelectronics, "Datasheet: STMPE1600 16-bit port expander with ultra-low power consumption Xpander Logic™," [Online]. Available:
<http://www.st.com/content/ccc/resource/technical/document/datasheet/group1/5c/a5/55/85/a8/6b/41/3a/CD00260203/files/CD00260203.pdf/jcr:content/translations/en.CD00260203.pdf>. [Accessed 27 November 2017].
- [23] National Instruments , "Strain Gauge Measurement – A Tutorial," [Online]. Available:
http://elektron.pol.lublin.pl/elekp/ap_notes/NI_AN078_Strain_Gauge_Meas.pdf. [Accessed 28 November 2017].
- [24] W. H. J. Hayt, J. E. Kemmerly and S. M. Durbin, Engineering Circuit Analysis, 7th ed., New York, New York: McGraw-Hill, 2007, p. 25.
- [25] T. R. Kuphaldt, Lessons in Electric Circuits, 5th ed., vol. I – DC, 2006, pp. 321-328.
- [26] Omega Engineering, "Specifications: Full-Bridge Thin-Beam Load Cells," [Online]. Available: <https://www.omega.com/pressure/pdf/LCL.pdf>. [Accessed 24 November 2017].
- [27] Phidgets, Inc., "PhidgetBridge 4-Input: Specifications," [Online]. Available:
<https://www.phidgets.com/?tier=3&catid=98&pcid=78&prodid=35>. [Accessed 24 November 2017].
- [28] Phidgets, Inc., "PhidgetBridge 4-Input: User Guide," [Online]. Available:
https://www.phidgets.com/?tier=3&catid=98&pcid=78&prodid=35#Technical_Details. [Accessed 24 November 2017].
- [29] Shimadzu Corporation, "Shimadzu AGS-X Series Precision Universal / Tensile Tester," [Online]. Available: http://www.shimadzu.com/an/test/universal/ags-x/ags-x_6.html. [Accessed 24 November 2017].
- [30] "Legacy Support: OS - Windows," Phidgets Inc., [Online]. Available:
https://www.phidgets.com/docs21/OS_-_Windows#Quick_Downloads. [Accessed 19 November 2017].
- [31] Stratasys Ltd., "uPrint SE Plus: Specifications," [Online]. Available:
<http://www.stratasys.com/3d-printers/uprint-se-plus>. [Accessed 24 November 2017].
- [32] R. Hibbeler, Engineering Mechanics: Statics, 12 ed., Upper Saddle River, New Jersey: Pearson Prentice Hall, 2010, pp. 117-183.
- [33] STMicroelectronics, "AN4545 Application Note: VL6180X basic ranging application note," [Online]. Available:
http://www.st.com/content/ccc/resource/technical/document/application_note/d5/bb/ec/94/7d/1e/40/a0/DM00122600.pdf/files/DM00122600.pdf/jcr:content/translations/en.DM00122600.pdf. [Accessed 27 November 2017].
- [34] STMicroelectronics, "Datasheet: STM32L053R8," [Online]. Available:
<http://www.st.com/content/ccc/resource/technical/document/datasheet/8a/f4/9d/d7/61/1b/46/b4/DM00105960.pdf/files/DM00105960.pdf/jcr:content/translations/en.DM00105960.pdf>. [Accessed 27 November 2017].

- [35] "Introduction to the C# Language and the .NET Framework," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. [Accessed 19 November 2017].
- [36] "Overview of the .NET Framework," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview>. [Accessed 11/19/17 November 2017].
- [37] "Introduction to WPF," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf>. [Accessed 19 November 2017].
- [38] ".NET Framework Class Library," Microsoft, [Online]. Available: [https://msdn.microsoft.com/en-us/library/gg145045\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/gg145045(v=vs.110).aspx). [Accessed 19 November 2017].
- [39] "C# 6.0 draft Language Specification," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/index>. [Accessed 19 November 2017].
- [40] "C# Reference," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index>. [Accessed 19 November 2017].
- [41] "C# Programming Guide," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index>. [Accessed 19 November 2017].
- [42] "Ubuntu for Desktops," Canonical Ltd., [Online]. Available: <https://www.ubuntu.com/desktop>. [Accessed 19 November 2017].
- [43] "GCC, the GNU Compiler Collection," Free Software Foundation, Inc., [Online]. Available: <https://gcc.gnu.org/>. [Accessed 19 November 2017].
- [44] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 8th ed., Hoboken, New Jersey: John Wiley & Sons, Inc., 2009, pp. 153-182.
- [45] "OxyPlot," MIT, [Online]. Available: <http://www.oxyplot.org/>. [Accessed 19 November 2017].
- [46] STMicroelectronics, "NUCLEO-L053R8," [Online]. Available: <http://www.st.com/en/evaluation-tools/nucleo-l053r8.html#sw-tools-scroll>. [Accessed 23 November 2017].
- [47] "Threading Model," Microsoft, [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/threading-model>. [Accessed 20 November 2017].
- [48] "Legacy Support: OS - Linux," Phidgets Inc, [Online]. Available: https://www.phidgets.com/docs21/OS_-_Linux#Quick_Downloads. [Accessed 20 November 2017].
- [49] Yaskawa America, Inc. Motoman Robotics Division, "Product Specifications: SIA50D," [Online]. Available: <https://cdn2.hubspot.net/hubfs/366775/Robots/SIA50D.pdf>. [Accessed 21 November 2017].

- [50] P. Lewis, Interviewee, *WVRTC Robot Operator*. [Interview]. 15 November 2017.
- [51] "SIA50D 50kg Payload 7-Axis Robot," Yaskawa America, Inc. Motoman Robotics Division, [Online]. Available: <https://www.motoman.com/industrial-robots/sia50d>. [Accessed 21 November 2017].
- [52] Yaskawa America, Inc. Motoman Robotics Division, "Product Specifications: MH215 II, MH250 II," [Online]. Available: https://www.motoman.com/hubfs/PDFs/MH215II_MH250II.pdf. [Accessed 21 November 2017].
- [53] Yaskawa America, Inc. Motoman Robotics Division, "MH250 II: High-Speed, Material Handling Robot," [Online]. Available: <https://www.motoman.com/industrial-robots/mh250-ii>. [Accessed 21 November 2017].
- [54] WVRTC: West Virginia Robotic Technology Center, "WVRTC Talks Zero-G Testing with WVU Students," [Online]. Available: http://wvrtc.com/news/wvrtc-talks-zero-g-testing-with-wvu-students/attachment/dsc_0588/. [Accessed 21 November 2017].
- [55] "JR3 Multi-Axis Load Cell Technologies: Products," JR3, Inc., [Online]. Available: <http://www.jr3.com/products.html>. [Accessed 21 November 2017].
- [56] C. C. Panther, "CBSS Test Plan Development," West Virginia Robotic Technology Center, Internal Report, 2016.
- [57] Leica Geosystems, subsidiary of Hexagon Manufacturing Intelligence, "Leica Absolute Tracker AT901: Product Brochure," [Online]. Available: http://www.measure.com.au/tiny_mce_gallery/files/leica_absolute_tracker_at901_brochure_en.pdf. [Accessed 21 November 2017].
- [58] Leica Geosystems, subsidiary of Hexagon Manufacturing Intelligence, "Leica Absolute Tracker: ASME B89.4.19-2006 Specifications," [Online]. Available: http://www.leica-geosystems.no/downloads123/m1/metrology/AT901/brochures-datasheet/Leica%20Absolute%20Tracker_ASME%20Specifications_en.pdf. [Accessed 21 November 2017].
- [59] "Leica T-Mac: 6 DoF tracking device for automated applications," Hexagon Manufacturing Intelligence, [Online]. Available: <http://www.hexagonmi.com/products/laser-tracker-systems/leica-tmac>. [Accessed 21 November 2017].
- [60] W. Navidi, *Statistics for Engineers and Scientists*, 3rd ed., New York, New York: McGraw-Hill, 2011.
- [61] "MATLAB," MathWorks, [Online]. Available: <https://www.mathworks.com/products/matlab.html>. [Accessed 30 October 2017].
- [62] "Documentation: interp1," MathWorks, [Online]. Available: <https://www.mathworks.com/help/matlab/ref/interp1.html>. [Accessed 2 November 2017].

- [63] "Documentation: smoothdata," MathWorks, [Online]. Available:
https://www.mathworks.com/help/matlab/ref/smoothdata.html#inputarg_window.
[Accessed 1 November 2017].
- [64] B. Kolman and D. R. Hill, Elementary Linear Algebra with Applications, 9th ed., Upper Saddle River, New Jersey: Pearson Education Inc, 2008, pp. 61-62.
- [65] "Documentation: alignsignals," MathWorks, [Online]. Available:
https://www.mathworks.com/help/signal/ref/alignsignals.html?s_tid=doc_ta. [Accessed 3 November 2017].
- [66] "Documentation: finddelay," MathWorks, [Online]. Available:
<https://www.mathworks.com/help/signal/ref/finddelay.html>. [Accessed 3 November 2017].
- [67] "Documentation: xcorr," MathWorks, [Online]. Available:
<https://www.mathworks.com/help/signal/ref/xcorr.html>. [Accessed 3 November 2017].

APPENDIX A | CBSS Nucleo Code

```
////////////////////////////////////
// Beginning of code
////////////////////////////////////

#include "mbed.h"

Serial ser(SERIAL_TX, SERIAL_RX); // set-up serial to pc
I2C i2c(I2C_SDA, I2C_SCL); // Set up I2C on the Nucleo L053R8
#define def_addr (0x52) // I2C address of VL6180X shifted by 1 bit
                        //(0x29 << 1) so the R/W command can be added

//new addresses of sensors that default to 0x29
#define tof_top (0x54) // 2A << 1
#define tof_left (0x56) // 2B << 1
#define tof_bottom (0x58) // 2C << 1
#define tof_right (0x5A) // 2D << 1

//address of the GPIO expander
#define stmpe1600 (0x84) // 0x42 << 1

////////////////////////////////////
// Split 16-bit register address into two bytes and write
// the address + data via I2C
////////////////////////////////////
void WriteByte(wchar_t reg,char data,char addr)
{
    char data_write[3];
    wait_ms(1);
    data_write[0] = (reg >> 8) & 0xFF; // MSB of register address
    data_write[1] = reg & 0xFF; // LSB of register address
    data_write[2] = data & 0xFF;
    i2c.write(addr, data_write, 3);
    wait_ms(1);
}
```

```

////////////////////////////////////
// Split 16-bit register address into two bytes and write
// the address + 2 data bytes via I2C
////////////////////////////////////
void Write2Bytes(wchar_t reg,char data,char addr)
{
    char data_write[4];
    wait_ms(1);
    data_write[0] = (reg >> 8) & 0xFF; // MSB of register address
    data_write[1] = reg & 0xFF; // LSB of register address
    data_write[2] = (data >> 8) & 0xFF; // MSB of data
    data_write[3] = data & 0xFF; //LSB of data
    i2c.write(addr, data_write, 4);
    wait_ms(1);
}

////////////////////////////////////
// Split 16-bit register address into two bytes and write
// required register address to VL6180X and read the data back
////////////////////////////////////
char ReadByte(wchar_t reg, char addr)
{
    char data_write[2];
    char data_read[1];
    wait_ms(1);
    data_write[0] = (reg >> 8) & 0xFF; // MSB of register address
    data_write[1] = reg & 0xFF; // LSB of register address
    i2c.write(addr, data_write, 2);
    i2c.read(addr, data_read, 1);
    wait_ms(1);
    return data_read[0];
}

```

```

////////////////////////////////////
// load settings
////////////////////////////////////
int VL6180X_Init(char addr)
{
    char reset;
    reset = ReadByte(0x016, addr);
    if (reset==1) // check to see has it be Initialised already
    {
        //////////////////////////////////
        // Add manufacturer settings
        //////////////////////////////////
        // Mandatory : private registers
        WriteByte(0x0207, 0x01, addr);
        WriteByte(0x0208, 0x01, addr);
        WriteByte(0x0096, 0x00, addr);
        WriteByte(0x0097, 0xfd, addr);
        WriteByte(0x00e3, 0x00, addr);
        WriteByte(0x00e4, 0x04, addr);
        WriteByte(0x00e5, 0x02, addr);
        WriteByte(0x00e6, 0x01, addr);
        WriteByte(0x00e7, 0x03, addr);
        WriteByte(0x00f5, 0x02, addr);
        WriteByte(0x00d9, 0x05, addr);
        WriteByte(0x00db, 0xce, addr);
        WriteByte(0x00dc, 0x03, addr);
        WriteByte(0x00dd, 0xf8, addr);
        WriteByte(0x009f, 0x00, addr);
        WriteByte(0x00a3, 0x3c, addr);
        WriteByte(0x00b7, 0x00, addr);
        WriteByte(0x00bb, 0x3c, addr);
        WriteByte(0x00b2, 0x09, addr);
        WriteByte(0x00ca, 0x09, addr);
        WriteByte(0x0198, 0x01, addr);
        WriteByte(0x01b0, 0x17, addr);
        WriteByte(0x01ad, 0x00, addr);
        WriteByte(0x00ff, 0x05, addr);
        WriteByte(0x0100, 0x05, addr);
        WriteByte(0x0199, 0x05, addr);
        WriteByte(0x01a6, 0x1b, addr);
    }
}

```

```

WriteByte(0x01ac, 0x3e, addr);
WriteByte(0x01a7, 0x1f, addr);
WriteByte(0x0030, 0x00, addr);

// Recommended : Public registers - See data sheet for more detail
//WriteByte(0x0011, 0x10, addr); // Enables polling for 'New Sample ready'
// when measurement completes
WriteByte(0x010a, 0x30, addr); // Set the averaging sample period
// (compromise between lower noise and
// increased execution time)
WriteByte(0x003f, 0x46, addr); // Sets the light and dark gain (upper
// nibble). Dark gain should not be
// changed.
WriteByte(0x0031, 0xFF, addr); // sets the # of range measurements after
// which auto calibration of system is
// performed
WriteByte(0x0040, 0x63, addr); // Set ALS integration time to 100ms
WriteByte(0x002e, 0x01, addr); // perform a single temperature calibration
// of the ranging sensor

//Optional: Public registers - See data sheet for more detail
WriteByte(0x001b, 0x09, addr); // Set default ranging inter-measurement
// period to 100ms
WriteByte(0x003e, 0x31, addr); // Set default ALS inter-measurement period
// to 500ms
WriteByte(0x0014, 0x24, addr); // Configures interrupt on 'New Sample
// Ready threshold event'
WriteByte(0x016, 0x00, addr); //change fresh out of set status to 0
WriteByte(0x01C, 0x31, addr); //max convergence time on range
WriteByte(0x2A3, 0x00, addr); // turn interleaved mode off
}
return 0;
}

```

```

////////////////////////////////////
// Start a range measurement in single shot mode
////////////////////////////////////
int VL6180X_Start_Range(char addr)
{
    WriteByte(0x018,0x01, addr);
    return 0;
}

////////////////////////////////////
// poll for sample ready
////////////////////////////////////
int VL6180X_Poll_Range(char addr)
{
    char status;
    char range_status;

    // check the status
    status = ReadByte(0x04f, addr);
    range_status = status & 0x07;

    // wait for new measurement ready status
    while (range_status != 0x04)
    {
        status = ReadByte(0x04f, addr);
        range_status = status & 0x07;
        wait_ms(1); // (can be removed)
    }
    return 0;
}

////////////////////////////////////
// Read range result (mm)
////////////////////////////////////
int VL6180X_Read_Range(char addr)
{
    int range;
    range=ReadByte(0x062, addr);
    return range;
}

```

```

////////////////////////////////////
// clear interrupts
////////////////////////////////////
int VL6180X_Clear_Interrupts(char addr)
{
    WriteByte(0x015,0x07, addr);
    return 0;
}

////////////////////////////////////
// Change an I2C address
////////////////////////////////////
int VL6180X_Change_Address(char addr, char newAddr)
{
    WriteByte(0x0212, newAddr, addr);
    return 0;
}

////////////////////////////////////
// Main Program loop
////////////////////////////////////
int main()
{
    int range_l, range_b, range_r;

    i2c.frequency(400000);

    Write2Bytes(0x1312, 0x00F0, stmpe1600);
    wait_ms(100); // (make sure device resets)

    Write2Bytes(0x1312, 0x00D0, stmpe1600); //Bottom Sensor
    VL6180X_Change_Address(def_addr, 0x2C); // use tof_bottom addr now

    Write2Bytes(0x1312, 0x0090, stmpe1600); //Left Sensor
    VL6180X_Change_Address(def_addr, 0x2B); // use tof_left addr now

    Write2Bytes(0x1312, 0x0010, stmpe1600); //Right Sensor
    VL6180X_Change_Address(def_addr, 0x2D); // use tof_right addr now
}

```

```

// load settings onto VL6180X
VL6180X_Init(tof_left);
VL6180X_Init(tof_bottom);
VL6180X_Init(tof_right);

VL6180X_Clear_Interrupts(tof_bottom);
VL6180X_Clear_Interrupts(tof_left);
VL6180X_Clear_Interrupts(tof_right);

while (1)
{
    VL6180X_Start_Range(tof_left);
    VL6180X_Poll_Range(tof_left);
    range_l = VL6180X_Read_Range(tof_left);
    VL6180X_Clear_Interrupts(tof_left);

    VL6180X_Start_Range(tof_bottom);
    VL6180X_Poll_Range(tof_bottom);
    range_b = VL6180X_Read_Range(tof_bottom);
    VL6180X_Clear_Interrupts(tof_bottom);

    VL6180X_Start_Range(tof_right);
    VL6180X_Poll_Range(tof_right);
    range_r = VL6180X_Read_Range(tof_right);
    VL6180X_Clear_Interrupts(tof_right);

    // send range to pc by serial
    ser.printf("%d %d %d\r\n", range_l, range_b, range_r);
    //wait(0.1); //delay
}
}

```


APPENDIX B | C++ CBSS Data Logger

```
// necessary libraries
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <termios.h>
#include <phidget21.h>
#include <thread>
#include <iostream>
#include <fstream>
#include <ctime>
#include <string.h>
#include <cstdio>
#include <unistd.h>
#include <sstream>

//allows std library function to be used without std:: prefix
using namespace std;

//Define Constants, Phidget refresh rate, phidget serial numbers,
//logger sampling rate, and nucleo device name
const int PHIDGET_REFRESH_RATE = 192;
const int PALM_A_FORCE_SENSORS = 476983;
const int PALM_B_FORCE_SENSORS = 404917;
const int PALM_C_FORCE_SENSORS = 410770;
const int SAMPLE_TIME_US = 200000;
const int NO_FORCE = -10;
const int NO_RANGE = 255;
const string NUCLEO = "/dev/ttyACM0";

//Variables to sample, CBSS forces and ranges
double forceA0 = NO_FORCE, forceA1 = NO_FORCE, forceA2 = NO_FORCE,
forceA3 = NO_FORCE;
double forceB0 = NO_FORCE, forceB1 = NO_FORCE, forceB2 = NO_FORCE,
forceB3 = NO_FORCE;
double forceC0 = NO_FORCE, forceC1 = NO_FORCE, forceC2 = NO_FORCE,
forceC3 = NO_FORCE;
double rangeA = NO_RANGE, rangeB = NO_RANGE, rangeC = NO_RANGE;

//Other useful state variables
int serial = 0;
bool logOver = false;
bool serialDone = false;
```

```

//display basic phidgets board information, called by attach handler
void display_generic_properties(CPhidgetHandle phid)
{
    int sernum, version;
    const char *deviceptr;
    CPhidget_getDeviceType(phid, &deviceptr);
    CPhidget_getSerialNumber(phid, &sernum);
    CPhidget_getDeviceVersion(phid, &version);

    printf("%s\n", deviceptr);
    printf("Version: %8d SerialNumber: %10d\n", version, sernum);
    return;
}

// attach handler - executes everytime a phidgets board is plugged in
// via USB to pc
int CCONV AttachHandler(CPhidgetHandle phid, void *userptr)
{
    // create handle to phidgets board
    CPhidgetBridgeHandle bridge = (CPhidgetBridgeHandle)phid;

    // enable each bridge input to load cell
    CPhidgetBridge_setEnabled(bridge, 0, PTRUE);
    CPhidgetBridge_setEnabled(bridge, 1, PTRUE);
    CPhidgetBridge_setEnabled(bridge, 2, PTRUE);
    CPhidgetBridge_setEnabled(bridge, 3, PTRUE);

    // set each bridge input to the highest gain (128)
    CPhidgetBridge_setGain(bridge, 0, PHIDGET_BRIDGE_GAIN_128);
    CPhidgetBridge_setGain(bridge, 1, PHIDGET_BRIDGE_GAIN_128);
    CPhidgetBridge_setGain(bridge, 2, PHIDGET_BRIDGE_GAIN_128);
    CPhidgetBridge_setGain(bridge, 3, PHIDGET_BRIDGE_GAIN_128);
    CPhidgetBridge_setDataRate(bridge, PHIDGET_REFRESH_RATE);

    // evidence that handler worked
    printf("Attach handler ran!\n");
    display_generic_properties((CPhidgetHandle)bridge);

    //exit function
    return 0;
}

```

```

// detach handler - executes everytime a phidgets board is removed
// via USB from pc
int CCONV DetachHandler(CPhidgetHandle phid, void *userptr)
{
    // create handle to phidgets board and get serial no.
    CPhidgetBridgeHandle bridge = (CPhidgetBridgeHandle)phid;
    CPhidget_getSerialNumber((CPhidgetHandle)bridge, &serial);

    // set sensors on specific phidgets board that was removed
    // from pc to NO_FORCE
    if (serial == PALM_A_FORCE_SENSORS)
        forceA0 = forceA1 = forceA2 = forceA3 = NO_FORCE;
    else if (serial == PALM_B_FORCE_SENSORS)
        forceB0 = forceB1 = forceB2 = forceB3 = NO_FORCE;
    else if (serial == PALM_C_FORCE_SENSORS)
        forceC0 = forceC1 = forceC2 = forceC3 = NO_FORCE;

    // evidence of handler working
    printf("Detach handler ran!\n");
    display_generic_properties((CPhidgetHandle)bridge);

    //exit function
    return 0;
}

// error handler. Runs if Phidget boards experience error
int CCONV ErrorHandler(CPhidgetHandle phid, void *userptr, int
ErrorCode, const char *errorStr)
{
    //print phidget specific error
    printf("Error event: %s\n",errorStr);
    return 0;
}

```

```

// data received handler. Runs when a single load cell sample is done
int CCONV data(CPhidgetBridgeHandle phid, void *userPtr, int index,
double val)
{
    // create handle to phidgets board and get serial no.
    CPhidgetBridgeHandle bridge = (CPhidgetBridgeHandle)phid;
    CPhidget_getSerialNumber((CPhidgetHandle)bridge, &serial);

    // determine which phidgets board sampled a load cell based on
    // the board serial number
    if (serial == PALM_A_FORCE_SENSORS)
    {
        //if palm A phidgets board, then update the specific force
        //corresponding to the sampled load cell. Index is the
        //phidgets board physical input
        switch(index)
        {
            case 0: forceA0 = val; break;
            case 1: forceA1 = val; break;
            case 2: forceA2 = val; break;
            case 3: forceA3 = val; break;
        }
    }
    else if (serial == PALM_B_FORCE_SENSORS)
    {
        //if palm B phidgets board, then update the specific force
        //corresponding to the sampled load cell. Index is the
        //phidgets board physical input
        switch(index)
        {
            case 0: forceB0 = val; break;
            case 1: forceB1 = val; break;
            case 2: forceB2 = val; break;
            case 3: forceB3 = val; break;
        }
    }
}

```

```

else if (serial == PALM_C_FORCE_SENSORS)
{
    //if palm C phidgets board, then update the specific force
    //corresponding to the sampled load cell. Index is the
    //phidgets board physical input
    switch(index)
    {
        case 0: forceC0 = val; break;
        case 1: forceC1 = val; break;
        case 2: forceC2 = val; break;
        case 3: forceC3 = val; break;
    }
}

// exit function
return 0;
}

// logging function that runs constantly on its own thread
// after phidgets devices are opened
void logger()
{
    // file handle
    ofstream myfile;

    // create filename containing current timestamp
    time_t rawtime = time(NULL);
    struct tm *timeinfo;
    char stamp[80];
    timeinfo = localtime(&rawtime);
    strftime(stamp, 80, "%F_%H-%M-%S", timeinfo);
    string timestamp(stamp);
    string filename = "LOG_" + timestamp + ".txt";

    //open file for logging purposes
    myfile.open(filename.c_str());

    // logger time variables
    clock_t start = clock(); // test start time
    clock_t t2, t1 = clock(); //periodic trackers

```

```

//log until user exits on main thread
while(!logOver)
{
    // get current time
    t2 = clock();

    // if time elapsed is greater than or equal to
    // sampling time, then log data
    if((t2-t1) >= SAMPLE_TIME_US)
    {
        // update other time variable for comparison
        // to other one (update now to have a time
        // stamp before slower write to file operations)
        t1 = clock();

        //log timestamp, forces, and ranges to one line
        //of the txt file the separated by spaces
        myfile << (double)(t2 - start)/1000000 << " ";
        myfile << forceA0 << " " << forceA1 << " ";
        myfile << forceA2 << " " << forceA3 << " ";
        myfile << forceB0 << " " << forceB1 << " ";
        myfile << forceB2 << " " << forceB3 << " ";
        myfile << forceC0 << " " << forceC1 << " ";
        myfile << forceC2 << " " << forceC3 << " ";
        myfile << rangeA << " " << rangeB << " ";
        myfile << rangeC << "\n";
    }
}

//close the file and exit function
myfile.close();
return;
}

```

```

// Connects to nucleo and attempts to read range sensors.
// Function is ran on a separate thread
void rangeReader()
{
    //serial structure, handle, and buffer
    struct termios tio;
    int tty_fd;
    char buffer[256];

    //open serial device defined globally
    tty_fd=open(NUCLEO.c_str(), O_RDONLY | O_NOCTTY);

    // if invalid serial handle print error and exit
    if(tty_fd < 0) {printf("Error opening Port!\n"); exit (-1);}

    //fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);// make the reads non-
    //blocking

    // set serial device parameters
    memset(&tio,0,sizeof(tio));
    tio.c_iflag=IGNPAR;
    tio.c_oflag=0;
    tio.c_cflag=CS8|CREAD|CLOCAL;    //8n1, see termios.h for
                                     //more information
    tio.c_lflag=ICANON;
    tio.c_cc[VMIN]=1;
    cfsetospeed(&tio,B9600);        // 9600 baud
    cfsetispeed(&tio,B9600);        // 9600 baud

    // flush port and set parameters
    tcflush(tty_fd, TCIFLUSH);
    tcsetattr(tty_fd,TCSANOW,&tio);
}

```

```

// read port and update range variables until user exits
while (!serialDone)
{
    // if serial data is present, parse it
    if (read(tty_fd,&buffer,sizeof(buffer))>0)
    {
        //printf("Data: %s\n", buffer);

        // input buffer variables
        string input = buffer;
        string temp;

        // split the input buffer based on space character
        stringstream split(input);

        // get range A and convert from string to double
        getline(split, temp, ' ');
        rangeA = atof(temp.c_str());

        // get range B and convert from string to double
        getline(split, temp, ' ');
        rangeB = atof(temp.c_str());

        // get range C and convert from string to double
        getline(split, temp, ' ');
        rangeC = atof(temp.c_str());
    }
}

//close serial port when user exits
close(tty_fd);
}

// Main program
int main(int argc, char* argv[])
{
    // bridge pointers brA,B,C are CBSS specific
    CPhidgetBridgeHandle brA, brB, brC;

    // create bridges
    CPhidgetBridge_create(&brA);
    CPhidgetBridge_create(&brB);
    CPhidgetBridge_create(&brC);
}

```



```

    // attach events
    CPhidget_set_OnAttach_Handler((CPhidgetHandle)brA, AttachHandler,
NULL);
    CPhidget_set_OnDetach_Handler((CPhidgetHandle)brA, DetachHandler,
NULL);
    CPhidget_set_OnError_Handler((CPhidgetHandle)brA, ErrorHandler,
NULL);
    CPhidgetBridge_set_OnBridgeData_Handler(brA, data, NULL);

    CPhidget_set_OnAttach_Handler((CPhidgetHandle)brB, AttachHandler,
NULL);
    CPhidget_set_OnDetach_Handler((CPhidgetHandle)brB, DetachHandler,
NULL);
    CPhidget_set_OnError_Handler((CPhidgetHandle)brB, ErrorHandler,
NULL);
    CPhidgetBridge_set_OnBridgeData_Handler(brB, data, NULL);

    CPhidget_set_OnAttach_Handler((CPhidgetHandle)brC, AttachHandler,
NULL);
    CPhidget_set_OnDetach_Handler((CPhidgetHandle)brC, DetachHandler,
NULL);
    CPhidget_set_OnError_Handler((CPhidgetHandle)brC, ErrorHandler,
NULL);
    CPhidgetBridge_set_OnBridgeData_Handler(brC, data, NULL);

    // open CBSS specific devices with serial numbers
    CPhidget_open((CPhidgetHandle)brA, PALM_A_FORCE_SENSORS);
    CPhidget_open((CPhidgetHandle)brB, PALM_B_FORCE_SENSORS);
    CPhidget_open((CPhidgetHandle)brC, PALM_C_FORCE_SENSORS);

    // phidget devices are now online, spawn logger and nucleo
    // threads
    std::thread logData(logger);
    std::thread updateRanges(rangeReader);

    //Wait for user input to exit program
    getchar();
    printf("Closing...\n");

    // update state variables so remaining two threads close
    logOver = true;
    serialDone = true;

```

```
// return to sequential execution
logData.join();
updateRanges.join();

//break down phidget devices
CPhidget_close((CPhidgetHandle)brA);
CPhidget_delete((CPhidgetHandle)brA);
CPhidget_close((CPhidgetHandle)brB);
CPhidget_delete((CPhidgetHandle)brB);
CPhidget_close((CPhidgetHandle)brC);
CPhidget_delete((CPhidgetHandle)brC);

//exit program
return 0;
}
```

APPENDIX C | REMAINING NOMINAL CONTACT FSS TESTS

C.1 Test #2 Plots

Figure C.1: FSS to JR3 Force-Torque Comparison (June 2016, Test #2)

Figure C.2: FSS Percent Errors with Data Deletion (June 2016, Test #2)

Figure C.3: Average FSS Percent Errors at each Applied Force (June 2016, Test #2)

C.2 Test #3 Plots

Figure C.4: FSS to JR3 Force-Torque Comparison (June 2016, Test #3)

Figure C.5: FSS Percent Errors with Data Deletion (June 2016, Test #3)

Figure C.6: Average FSS Percent Errors at each Applied Force (June 2016, Test #3)

APPENDIX D | REMAINING AUGUST 2016 TESTS

D.1 Test #2 Plots

Figure D.1: FSS to JR3 Force-Torque Comparison (August 2016, Test #2)

Figure D.2: FSS Percent Errors with Data Deletion (August 2016, Test #2)

Figure D.3: Average FSS Percent Errors for both Test Regions (August 2016, Test #2)

D.2 Test #3 Plots

Figure D.4: FSS to JR3 Force-Torque Comparison (August 2016, Test #3)

Figure D.5: FSS Percent Errors with Data Deletion (August 2016, Test #3)

Figure D.6: Average FSS Percent Errors for both Test Regions (August 2016, Test #3)

D.3 Test #4 Plots

Figure D.7: FSS to JR3 Force-Torque Comparison (August 2016, Test #4)

Figure D.8: FSS Percent Errors with Data Deletion (August 2016, Test #4)

Figure D.9: Average FSS Percent Errors for both Test Regions (August 2016, Test #4)

D.4 Test #5 Plots

Figure D.10: FSS to JR3 Force-Torque Comparison (August 2016, Test #5)

Figure D.11: FSS Percent Errors with Data Deletion (August 2016, Test #5)

Figure D.12: Average FSS Percent Errors for both Test Regions (August 2016, Test #5)

D.5 Test #6 Plots

Figure D.13: FSS to JR3 Force-Torque Comparison (August 2016, Test #6)

Figure D.14: FSS Percent Errors with Data Deletion (August 2016, Test #6)

Figure D.15: Average FSS Percent Errors for both Test Regions (August 2016, Test #6)

D.6 Test #7 Plots

Figure D.16: FSS to JR3 Force-Torque Comparison (August 2016, Test #7)

Figure D.17: FSS Percent Errors with Data Deletion (August 2016, Test #7)

Figure D.18: Average FSS Percent Errors for both Test Regions (August 2016, Test #7)

D.7 Test #8 Plots

Figure D.19: FSS to JR3 Force-Torque Comparison (August 2016, Test #8)

Figure D.20: FSS Percent Errors with Data Deletion (August 2016, Test #8)

Figure D.21: Average FSS Percent Errors for both Test Regions (August 2016, Test #8)

D.8 Test #9 Plots

Figure D.22: FSS to JR3 Force-Torque Comparison (August 2016, Test #9)

Figure D.23: FSS Percent Errors with Data Deletion (August 2016, Test #9)

Figure D.24: Average FSS Percent Errors for both Test Regions (August 2016, Test #9)

D.9 Test #10 Plots

Figure D.25: FSS to JR3 Force-Torque Comparison (August 2016, Test #10)

Figure D.26: FSS Percent Errors with Data Deletion (August 2016, Test #10)

Figure D.27: Average FSS Percent Errors for both Test Regions (August 2016, Test #10)

D.10 Test #11 Plots

Figure D.28: FSS to JR3 Force-Torque Comparison (August 2016, Test #11)

Figure D.29: FSS Percent Errors with Data Deletion (August 2016, Test #11)

Figure D.30: Average FSS Percent Errors for both Test Regions (August 2016, Test #11)