Firing Room Remote Application Software Development & Swamp Works Laboratory Robot Software Development

Janette Garcia
National Aeronautics and Space Administration
Kennedy Space Center
Computer Science

NIFS 2016 Spring Session
04/06/2016

Firing Room Remote Application Software Development

Janette Garcia[1]
NASA Kennedy Space Center (KSC), Merritt Island, Florida, 32899

ABSTRACT

The National Aeronautics and Space Administration (NASA) is creating a way to send humans beyond low Earth orbit, and later to Mars. Kennedy Space Center (KSC) is working to make this possible by developing a Spaceport Command and Control System (SCCS) which will allow the launch of Space Launch System (SLS). This paper's focus is on the work perform by the author in her first and second part of the internship as remote application software developer. During the first part of her internship, the author worked on the SCCS's software application layer by assisting multiple ground subsystems teams including Launch Accessories (LACC) and Environmental Control System (ECS) on the design, development, integration, and testing of remote control software applications. Then, on the second part of the internship, the author worked on the development of robot software at the Swamp Works Laboratory which is a research and technology development group which focuses on inventing new technology to help future In-Situ Resource Utilization (ISRU) missions.

Nomenclature

ACL	=	Application Control Language
COTS	=	Commercial off the Shelf Software
CUI	=	Compact Unique Identifier
ECS	=	Environmental Control System
GHe	=	Gaseous Helium
GSDO	=	Ground Systems Development & Operations
GSE	=	Ground Support Equipment
GN2	=	Gaseous Nitrogen
IDE	=	Integrated Development Environment
ILOA	=	Integrated Launch Operations Applications
KSC	=	Kennedy Space Center
LACC	=	Launch Accessories
LCC	=	Launch Control Center
LCS	=	Launch Control System
NASA	=	National Aeronautics and Space Administration
SCCS	=	Spaceport Command and Control System
SLS	=	Space Launch System
SRDS	=	Software Requirements and Design Specification
TD	=	Test Driver
VM	=	Virtual Machine
GUI	=	Graphical User Interface
ML	=	Mobile Launcher
PLC	=	Programmable Logic Controller
ICPS	=	Interim Cryogenic Propulsion Stage

[1] Remote Application Software Development, NASA NE-ES, Kennedy Space Center, The University of Texas – Rio Grande Valley.

I. Introduction

The author's[1] work in her second-term internship consisted in developing software for the Spaceport Command and Control System's (SCCS's) software application layer for the first part of her internship. The significance of this software is that it monitors and controls local hardware that is used to be able to launch SLS rocket to space. In order to accomplish her work, the author[1] was required to complete multiple steps including technical training, demonstrating technical skills, and assisting LACC, and ECS ground subsystems by meeting their milestones in the software development lifecycle. Specifically, the author[1] developed software applications (test scripts) in order for NASA engineers to spend less time in the testing phase of remote control graphical user interfaces (GUIs) also referred as remote control displays. The author[1] also developed multiple displays which will help monitor, and control the measurement on the mobile launcher (ML). After that, she developed a display for the Interim Cryogenic Propulsion Stage (ICPS) which is a significant part of NASA's new Space Launch System (SLS) rocket. After that, the author[1] developed another remote control display which will provide power to the ECS instrumentation located on the ML.

The first part of this report explains the objectives for the first part of the internship. Then, a technical approach is describe in order to accomplish the author's[1] objectives. After that, a brief description of the architecture of command and control system is presented. The next section is about the software developed by the author[1] during her two-month period for the first part of the internship. This last section focuses on the development of Mobile Launcher (ML) Distributer Power, Vehicle Assembly Building (VAB) ML Measurements, and Interim Cryogenic Propulsion Stage *(*ICPS) remote control graphical user interfaces (GUIs). It also talks about the development of Application Control Language (ACL) software applications (test scripts) to perform automated testing on the Tail Service Mast Umbilical (TSMU) and ML remote/local control user-interfaces.

II. Objectives

The objective for the first part of the internship was to develop remote control software applications in order to remotely control the local hardware used to launch rockets into space. Another objective is to perform automated testing on displays in order to reduce the time spent on their testing phase. Another objective is the documentation of software, and to make it accessible to the author's ground subsystem teams in order for them to re-utilize the code to perform future automated tests.

III. Technical Approach

The technical approach taken to reach the objectives was that of understanding each of the software development lifecycle phases that each ground subsystem team follows, and to requested and set up the information technology (IT) accounts that are required to support subsystem applications and displays such as ILOA Share Drive Access, Enhanced ICE Account, KDDMS Account, LCSDEV Domain Account, X-Win32 Account, Admin Rights, ClearQuest Account, AccuRev Account, and VEMS. The author[1] learned about the software development lifecycle phases from the Integrated Launch Operations Applications (ILOA) training provided by her mentor, and throughout her internship. She also learned about the implementation of remote applications using Application Control Language (ACL), a custom language created by KSC Application Services and Framework (ASF) team. This language provides an extension to C++ that allows the developers to focus on the application logic required to perform operations, and for executing sequences and rules within a Launch Control System (LCS) set. Control software applications were created in a Netbeans integrated development environment (IDE) in a Linux virtual machine (VM). The author[1] used Display Editor (DE) to develop user-friendly interfaces to allow NASA engineers at the Launch Control Center (LCC) to remotely control equipment out in the field. Without any of this software, the control of ground equipment such as valves, pumps, and motors can only be controlled manually. To make the code available to her team, the author[1] promoted the software into AccuRev, a configuration management program used by the SCCS team. In order to promote the code into AccuRev, the author[1] had to create multiple work orders (WO) in ClearQuest.

IV. Architecture

Learning about the command and control system high level architecture was important in order for the author[1] to complete her work. The way this works is that end items which are pumps, valves, motors, and power supplies publish their data on an industrial controller which is located in the field. This industrial controller is also known as a programmable logic controller (PLC). When the PLC receives the data, it then publishes it to a network where it is

received by the ground support equipment (GSE) gateway; that GSE gateway publishes the data over a message bus for any participant in the Launch Control System (LCS) to obtain. Therefore, the path of measurements go from end-items (hardware) to the PLC, then transfer to the GSE gateway, then to a message bus, and finally obtained by a user-interface or an application running in the application server.

The opposite process happens when commands are sent from the launch control center (LCC) to perform a task on a specific hardware component, or end-item. Commands are sent through the user interface, then they are published on the message bus, after which the GSE gateway acquires those commands, which publishes them to the appropiate PLC; the PLC then sends the command directly to the end-items. Finally, the specified task stated by the command is performed on the hardware component.

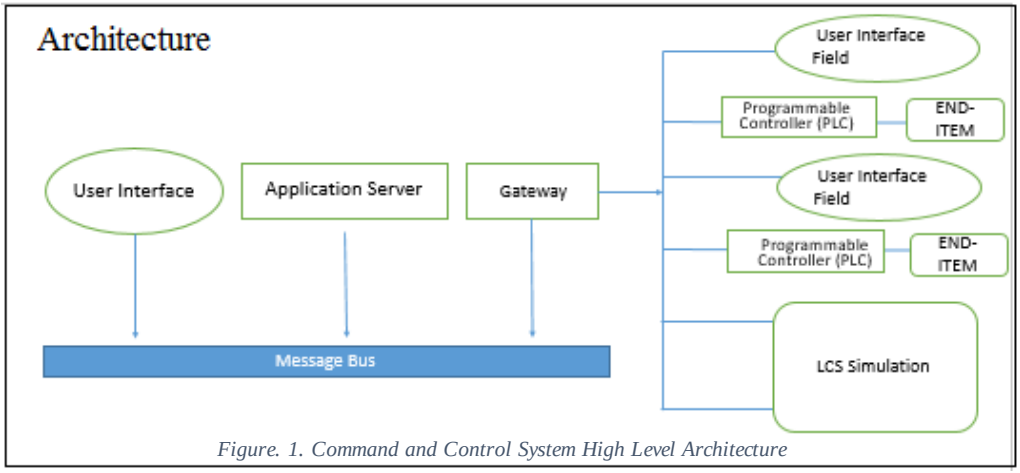

Figure. 1. Command and Control System High Level Architecture

V. Software Applications Developed

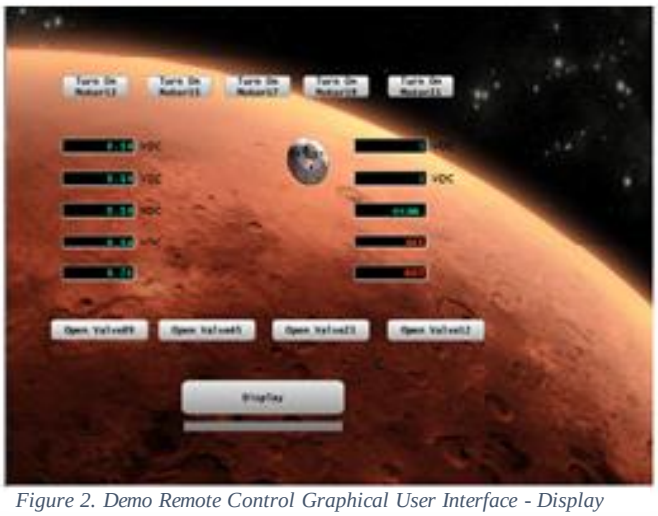

Figure 2. Demo Remote Control Graphical User Interface - Display

A. Skills Demonstrations

The remote control display as illustrated in **Figure 2.** was developed using Display Editor (DE) on a Linux Virtual Machine (VM). It consisted of at least one of each of the 4 different widgets, and a background image. The display also contained multiple text measurement widgets, which included one analog/float measurement formatted to show at least 2 decimal places and an enumerated string type measurement showing the text of the value. It was tested using the Display Test Driver (TD).

B. Remote Control Graphical User Interfaces (GUIs)

1. *Interim Cryogenic Propulsion Stage (ICPS) G12 Remote Control GUI*

The Interim Cryogenic Propulsion Stage (ICPS) is a significant part of the of NASA's new SLS rocket; it lies below the Orion capsule, at the top of the SLS rocket. It is a liquid oxygen/liquid hydrogen-based system. On the first test mission of Orion and SLS called Exploration Mission-1, the ICPS will give Orion the big push needed to fly beyond the moon before the spacecraft returns to Earth. The LC-39 Pad Environmental Control System (ECS) requires the ability to remotely control the ICPS gaseous nitrogen purge panels located on the ML. The purge panels provide a heated nitrogen purge to components within the SLS core stage. The remote control of this equipment is accomplished through the LCS using the ICPS remote control GUI that the author[1] created in a customized Display Editor (DE) as shown in **Figure 4.** This user-interface will send commands and receive data from the Pad ECS console operators to monitor and control the ICPS purge from the launch control center (LCC) in Firing Room 1. Understanding the ground integrated schematic created by NASA electrical engineers, and referring to the appropriate Software Requirements and Design Specification (SRDS) was essential for the author[1] to develop the remote control user-interface as shown in **Figure 3.**

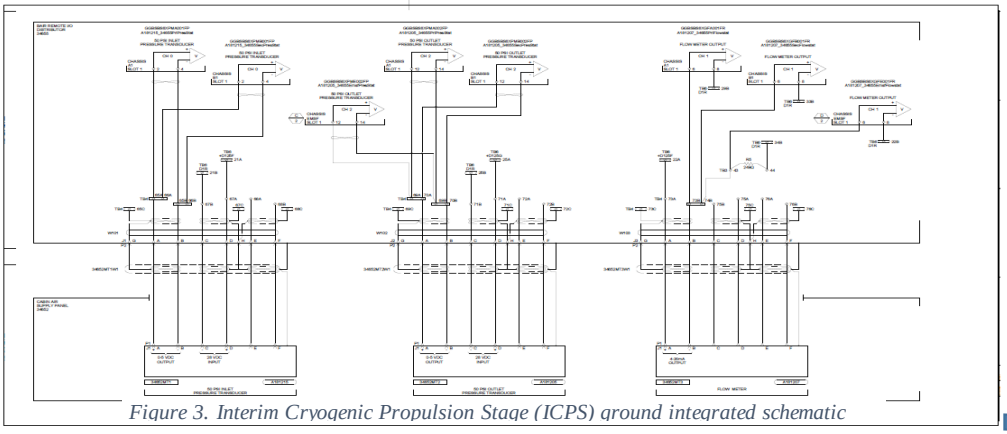

Figure 3. Interim Cryogenic Propulsion Stage (ICPS) ground integrated schematic

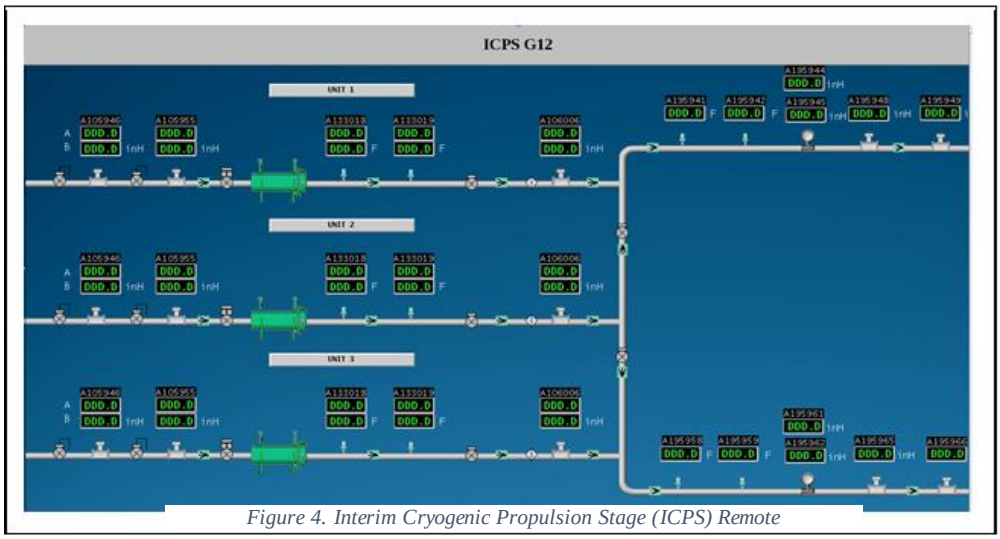

Figure 4. Interim Cryogenic Propulsion Stage (ICPS) Remote

2. *VAB ML Measurements Remote Control GUI*

The Vehicle Assembly Building (VAB) ML Interface Measurements Remote Control GUI was developed by the author[1] to allow the ECS Engineer to monitor all interface measurements on the ML. These measurements consist of System A and System B redundant measurements for duct temperature, duct pressure, and a calculated air humidity ratio based on relative humidity measurements. It will also provide the engineer with the ML elevation and PPU number that is providing air to the specific ML duct. The user-interface is also selectively colored to aid in identification of purge circuits, which are common to one of the three PPUs as shown in **Figure 5**. The SRDS, and a customized Display Editor (DE) were used by the author[1] to create the remote control user-friendly interface. Similar

to the previous GUI, each command button was set to a specific compact unique identifier (CUI).), which are utilized to control a specific piece of equipment out in the field that is essential for the launching of rockets to space.

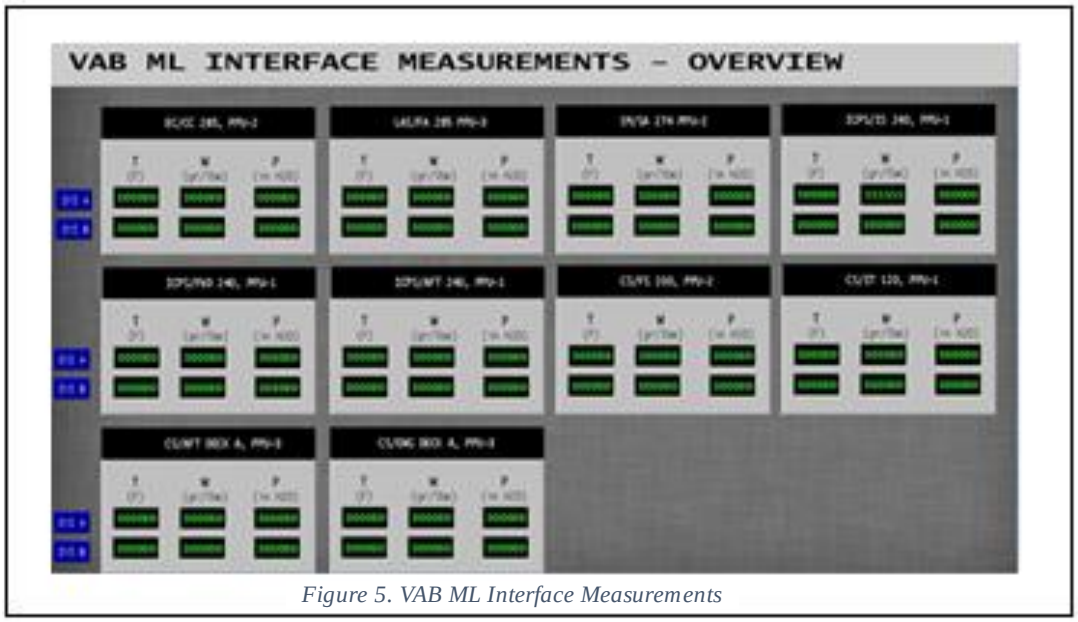

Figure 5. VAB ML Interface Measurements

3. *PAD ECS ML Remote Distributer Power Remote Control GUI*

The PAD ECS ML Remote Distributer GUI as illustrated in **Figure 6**. is made up of approximately 16 end item measurements and 50 commands. This user-friendly interface was created to provide power to the Environmental Control System (ECS) instrumentation located on the ML. It will allow the NASA engineers to power off primary and secondary electrical busses within the remote distributor, as well as to monitor the Ground Special Power (GSP) voltage and amperage measurements of the main power feeds. The Software Requirements and Design Specification (SRDS) and a customized a Display Editor (DE) were used to create the GUI. Each command button was then set to a specific Compact Unique Identifier (CUI).

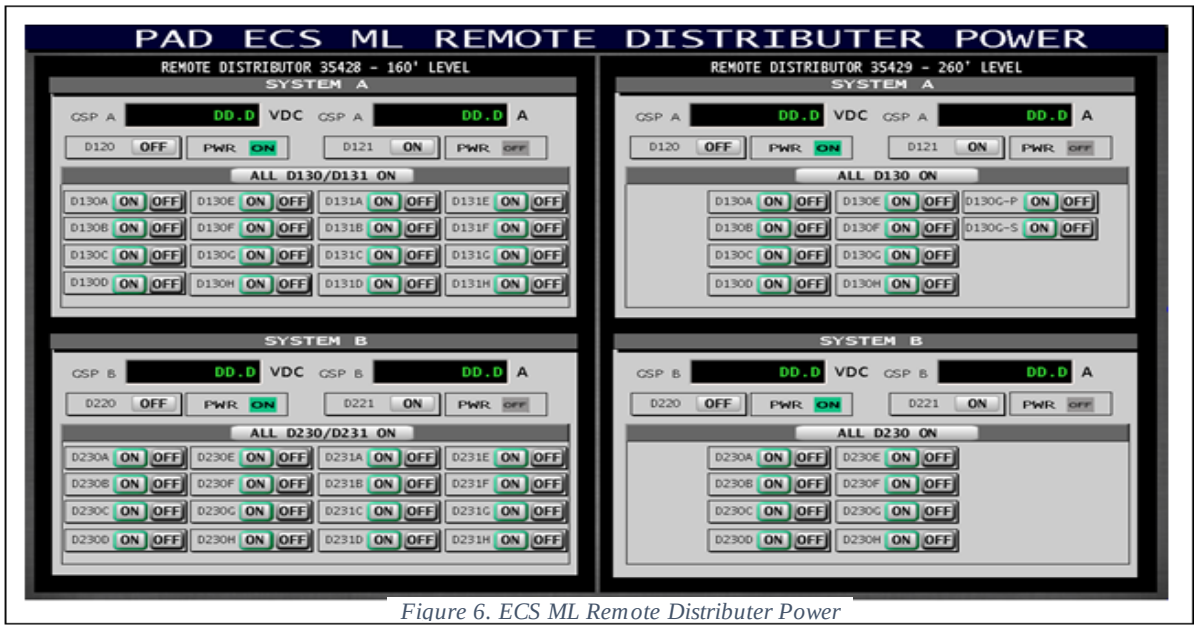

Figure 6. ECS ML Remote Distributer Power

C. Remote Control Software Applications To Perform Automated Testing

4. Remote Control Application for Tail Service Mast Umbilical (TSMU) Remote Control GUI

ACL software applications in a form of test script were developed by the author[1] to perform automated testing on the Tail Service Mast Umbilical (TSMU) remote control GUI. This test script verified approximately 291 CUIs that consisted of command buttons, indicators, and feedback commands for the (TSMU) remote control display. Performing manual testing on the TSMU remote control user-interface takes approximately 4 minutes for each command button, which adds up to 19.4 hours of testing. While manual testing takes 19.4 hours, automated testing takes approximately 8 minutes. Hence, performing automated testing reduces the time spent on the testing phase of the software development lifecycle. Other test scripts were also developed for the previous remote control GUIs as well as for the local versions. The author[1] promoted the scripts into AccuRev for her team to access them. She also created work orders (WO) on ClearQuest in order to be able to promote the code into the appropriate AccuRev repository of her ground subsystem team.

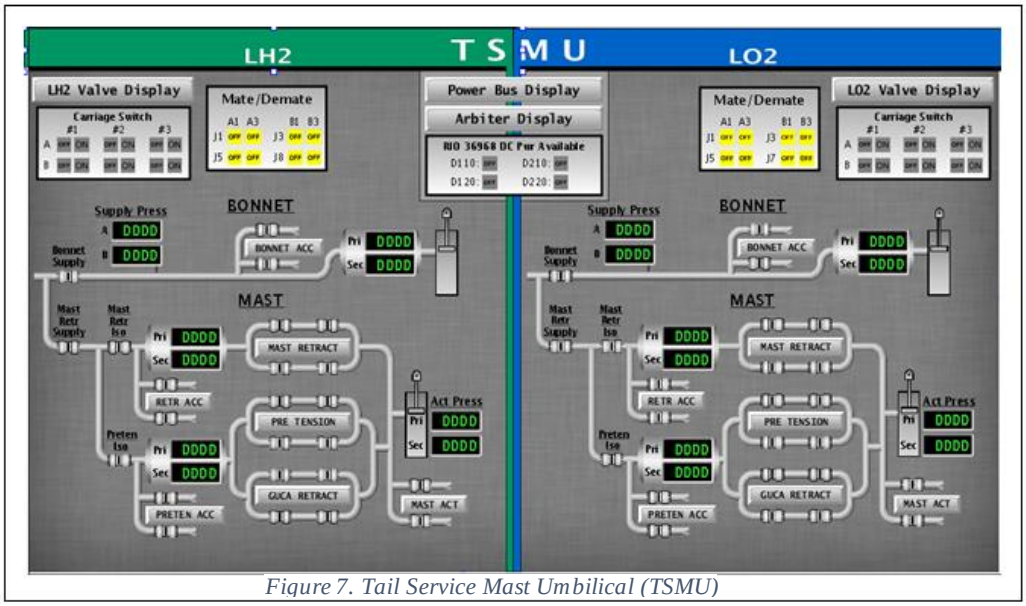

Figure 7. Tail Service Mast Umbilical (TSMU)

Robot Software Development

Janette Garcia[1]
NASA Kennedy Space Center (KSC), Merritt Island, Florida, 32899

Nomenclature

ISRU	=	In-Situ Resource Utilization
ROS	=	Robot Operating System
RASSOR 2.0	=	Regolith Advanced Surface Systems Robot
Swarmies	=	Small robots that performant behaviors
Gazebo	=	Robotic simulator
Swarmathon	=	Collegiate competition to develop robotics that aid in space exploration
Algorithm	=	Set of operation to be performed to solve a specific problem
GA	=	Genetic Algorithm

II. Introduction

The second part of the internship started on March 14th, 2016 and will last until May 6th, 2016. The author's[1] work consisted of robot software development at the KSC Swamp Works laboratory. The Swamp Works laboratory is a research and technology development group which focuses on inventing new technology to help future In-Situ Resource Utilization (ISRU) missions. Such technologies include Regolith Advanced Surface System Robot (RASSOR 2.0), a mining robot whose mission is to be a cargo delivery to Mars to prove In-Situ Resource Utilization (ISRU) capabilities[4]. Another technology is the creation of small autonomous robots referred as Swarmies which are a ground based research for ISRU missions[5]. During her three weeks of work at Swamp Works laboratory, the author[1] managed to teach herself about the robotic software that is utilized for both technologies. Moreover, she assisted her mentor in the testing phase of 16 Swarmies as well as the calibration of their inertial measurements unit (IMU). Additionally, she assisted her mentor in the NASA first Swarmathon Competition, a collegiate competition to develop robotics that will help in space exploration.

Note that this report only presents the author's[1] three weeks of work at Swamp Works laboratory. The first part of this report explains the objectives for the second part of the internship. Then, a technical approach is describe in order to accomplish the author's[1] objectives. After that, a brief description of the Robot Operating System (ROS) architecture is presented. The next section illustrates the author's[1] learning experience on RASSOR's 2.0 robotic software. The following section illustrates the learning experience on the Swarmies's robot software in a simulated environment using Gazebo, a robot simulator. Then next section describe the challenges that the author[1] overcome while working at Swamp Works Laboratory. Finally, a section describing the accomplishments during three weeks of work at Swamp Works are illustrated. The last sections of this report briefly summarizes the author's[1] overall internship experience as a software developer, and the individuals that the author[1] acknowledges.

I. Objectives

The author's[1] main objective is to assist in the KSC preparation for its first Swarmathon competition, which is a challenge to invent cooperative robotics to revolutionize space exploration. Another objective is to work on the robot software for RASSOR 2.0, a mining robot constructed at NASA Swamp Works Laboratory which will be utilized for future in-situ resource utilization **(**ISRU**)** missions.

III. Technical Approach

To assist in the development of robot software the author[1] taught herself how to build robot applications using Robot Operating System (ROS) framework. This framework provides libraries, device drivers, message passing, and

package management to help built robot applications[2]. In addition, the author[1] taught herself how to develop virtual robots using Gazebo, a robot simulator that is essential to test algorithms, design robots, and perform regression testing using realistic scenarios. The author[1] found Gazebo simulation tools very useful since it allows the developer to test algorithms without spending money on the construction of physical robots.[3] Moreover, the author[1] learned about the Swarmies software current implementation in order to better assist in the preparation of the first Swarmathon competition. She also furthered her understanding on RASSOR 2's current software implementation.

III. ROS

A. Architecture

The author[1] enjoyed learning about the ROS architecture, which consists of nodes, the master, messages, and topics. The role of nodes is to perform computations. The master plays an important role in ROS architecture since it acts as the registrar which keeps track of the nodes that are actively sending or receiving messages. Throughout her training, the author[1] understood the importance of the master in the ROS framework because when she did not run the master she ran into problems. Through this experience, she learned that there cannot exist any communication between nodes without the master. By writing simple robot applications, the author[1] understood how nodes communicate with each other through messages. A message consists of data that sends information to other nodes. When a node sends data, it is publishing to a topic as illustrated in **Figure 1**. These nodes can receive topics from other nodes by subscribing to the specific topic that they want to receive information from.

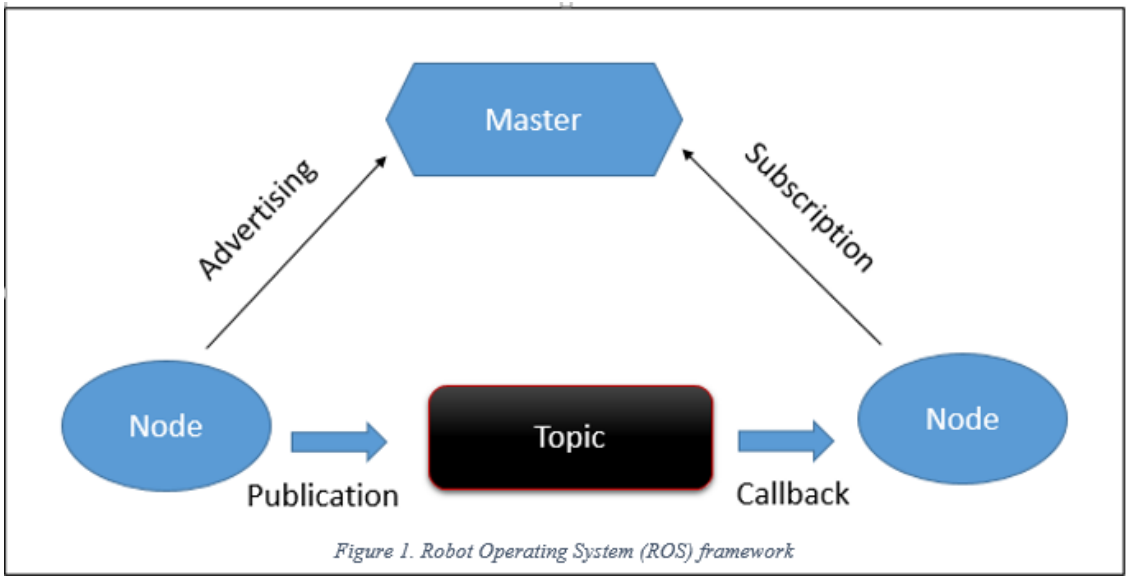

Figure 1. Robot Operating System (ROS) framework

IV. RASSOR 2 .0

A. RASSOR 2.0 Description

RASSOR 2.0 is a mining robot whose mission is to be a cargo delivery to Mars to prove ISRU capabilities. The way this this is going to work is that a lander will carry the RASSOR 2.0 and ISRU (in-situ resource utilization) processing plant. Once it is on the surface, RASSOR 2.0 will drive to the specific mining site that is assumed to be 100 meters from the lander. Once RASSOR 2.0 is on the mining site, it will lower its bucket drum and begin excavating regolith while slowly driving forward where there are higher concentrations of water[4].

B. RASSOR 2.0 High Level Onboard Software

During her three weeks of working at Swamp Works, the author[1] learned about RASSOR 2.0 software by going over the code and understanding every line of code out of more than 5,000 lines of code. **Figure 2.** illustrates the high level onboard software for RASSOR 2. To be specific, it illustrates the ROS nodes that make up RASSOR's 2.0 software. Each node is written in the C++ programming language, and they communicate with each other by sending messages such as target data, obstacle data from sensors, manipulator commands, motor commands, GPS data, robot

position, and Inertial Measurements Unit (IMU) data. In the coming weeks of her internship, the author[1] will be working on RASSOR 2 in order to improve the existing software implementation.

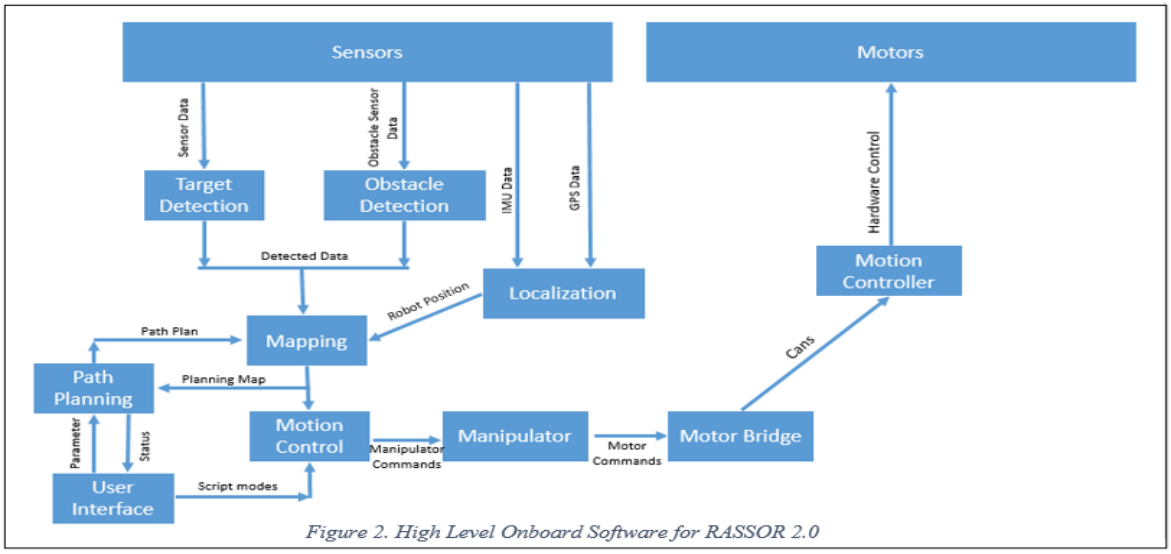

Figure 2. High Level Onboard Software for RASSOR 2.0

V. SWARMIES

A. Swarmies Description

The behavior of the robot swarm imitates the central-place foraging strategy of ants to find and collect resources in an unknown environment, and return those resources to the central site. These small robots do not have prior knowledge of the environment. They use trails as a simple indirect communication strategy and use a genetic algorithm to evolve a set of behavioral parameters[5].

B. Swarmies in a virtual environment

The author[1] managed to learn about the swarmies simulations using Gazebo. This simulation is conducted in order to test their software without having to do it in physical robots. The author[1] made some experiments with them in this virtual environment as shown in **Figure 3**. Those experiments consisted of controlling the simulated robots manually by using the arrow keys in the keyboard, as well as controlling them using a random search algorithm using C++ programming language. The implemented code is located inside their respective ROS workspace src folder, and then it is built using the catkin_make tool.

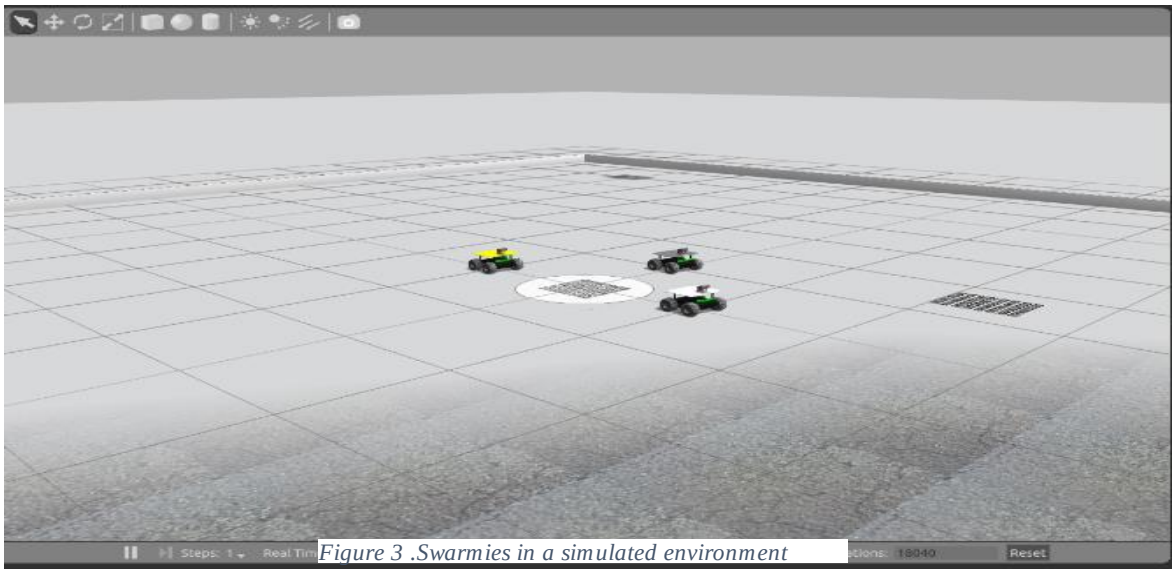

Figure 3 .Swarmies in a simulated environment

VI. Challenges

It has been challenging, and at the same exciting to learn and understand the RASSOR's 2.0 software in less than a month. The author's[1] main challenge was to understand how each of the components that make up the robot work in the world of robotic software. After going over each line of code on RASSOR's 2.0 software, and learning about each of the nodes that make up RASSOR 2.0 high level architecture, the author[1] has a better understanding of how to implement its robot software.

VII. Results for Three Weeks of Work at Swamp Works

1. Virtual Robots

The author's[1] work is in progress for this portion of the internship. So far the author[1] taught herself to create virtual reality robots in order to do regression testing faster, without the need of constructing physical robots. The virtual robots illustrated in **Figure 5.** illustrate a robot with camera and sensors which can detect obstacles just as a physical robot. **Figure 4.** illustrates a kitchen simulated environment with a virtual robot. If the author[1] invents a new algorithm, she can test her code on a virtual robot. The author[1] found this fascinating because she will be capable of writing code that can be run on a virtual, yet inexpensive robot.

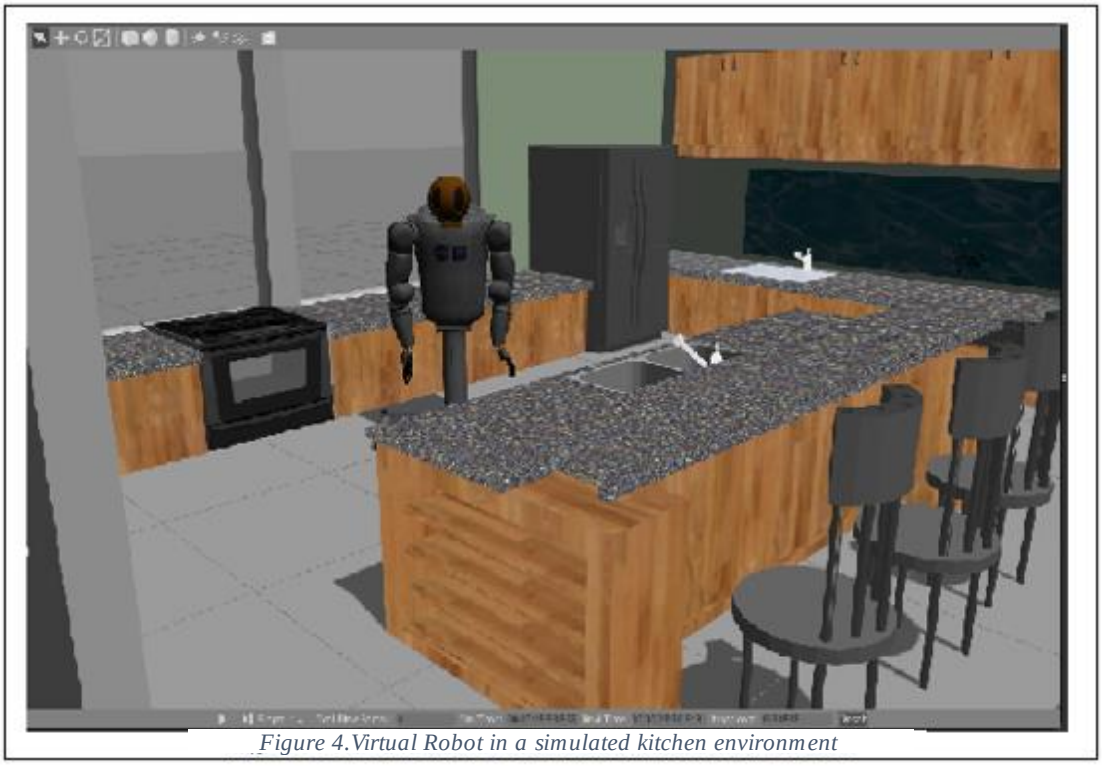

Figure 4.Virtual Robot in a simulated kitchen environment

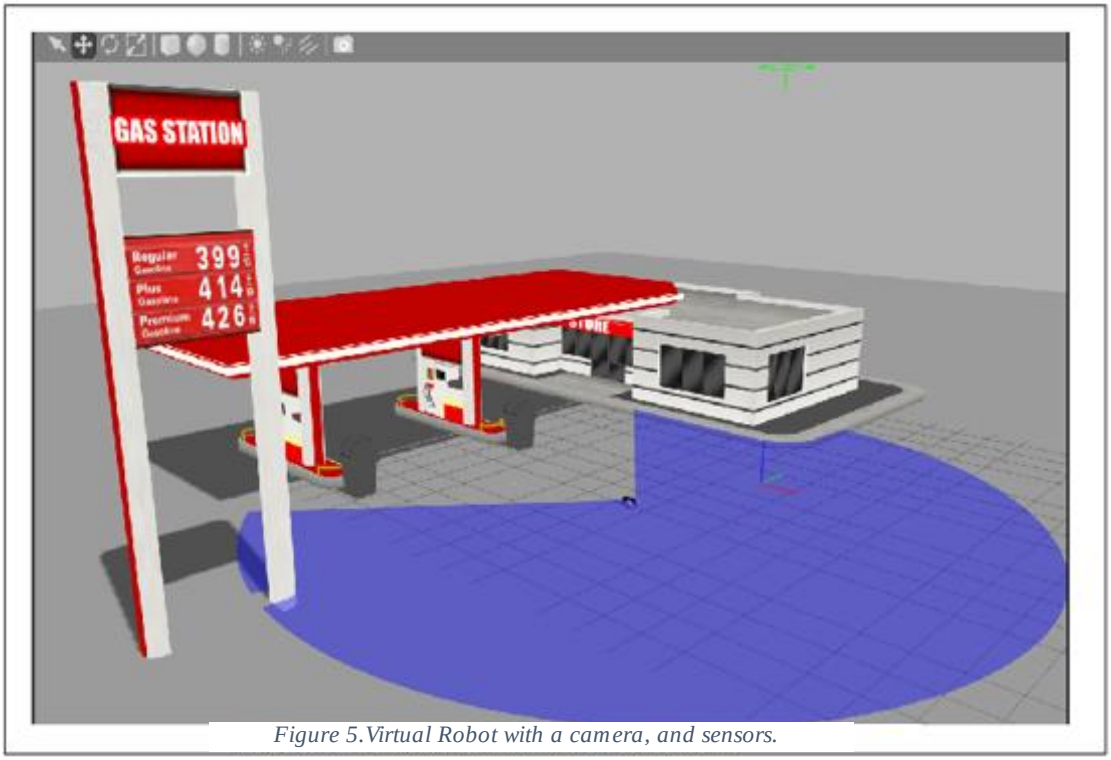

Figure 5.Virtual Robot with a camera, and sensors.

2. Construction of 10 Swarmies for 1st NASA Swarmathon Competition at KSC

The author[1] assisted in the construction of 10 swarmies, small robots, which are composed of 3D printed parts, a Global Positioning System (GPS), NUC Base, Battery Base, Battery Brace, Ultrasound Tower, Motors, Wheels, Ultrasound sensors, USB web camera, Arduino, and inertial measurement unit (IMU). This was a tedious process. The wheel motors and the batteries are placed inside the enclosed chassis. The micro-controller stack and all sensors are placed on top of the enclosed chassis. Then, the USB web camera is placed on top of the robot's protective lid in order to get a better angle on the AprilTags that will be on the ground. The IMU sensor is also placed on top next to the USB web camera.

3. Swarmies Software Testing/ Inertial Measurements Unit (IMU) calibration/Installation of Software for NASA Swarmathon

The author[1] has been assisting in the testing phase of 16 swarmies,which will be utilized in the NASA first Swarmathon Competition. This competition will take place at KSC in April 2016. The intention of this competition is for minority-serving universities to develop efficient search algorithms to make the swarmies work together to find targets. This code will be a great help in space exploration. The testing consists of connecting via secure shell (SSH) into each of the swarmies, cloning each of the universities code from their specific Github repository, and then verifying that the swarmies perform the steps on the code. A graphical user interface is used to observe that all the components of the robots are working correctly. Thanks to the numerous hours that the author[1] spent teaching herself about ROS framework, she is able to perform her task on the testing phase of the Swarmies software development lifecycle. Additionally, the author[1] helped with installing the software from GitHub for each of the 16 robots, as well as starting the IMU calibration. This IMU is utilized to measure linear and angular motion of the robot with a three axis accelerometer, which then sends the data to the central processing unit in order for the swarmies to be able to determine its own orientation in space.

IV. Conclusion

Overall, the author's[1] internship experience has been a worthwhile one, providing her with the opportunity to collaborate with multiple subsystems at KSC, and most importantly, for allowing her to contribute in the development of SCCS's top layer, as well as on the development lifecycle of robot software such as the one for RASSOR 2.0 at Swamp Works laboratory. The author[1] accomplished the development of remote control displays for Mobile Launcher, and, Interim Cryogenic Propulsion Stage. She also accomplished the development of ACL control applications to perform automated testing on the Tail Service Mast Umbilical remote control display as well as for the Mobile launcher remote control displays. In addition, she taught herself in less than a month about the robotic software used for RASSOR 2.0, and about the Swarmies simulation using Gazebo. Thanks to the shadowing opportunities that she participated in, the author[1] has a better understanding of each of the phases that make up the software development lifecycle. She enjoyed collaborating with individuals who have different fields of study, including electrical, and mechanical engineers.

Acknowledgments

The author[1] is pleased to thank and acknowledge Kurt Leucht, and Greg Rawl for the support and mentoring provided throughout the course of the internship. Additionally, thanks to Oscar Brooks, Caylyne Shelton, and Jamie Szafran for making her internship an exciting one. Thanks to all the reviewers for providing feedback on the author's[1] final report. Thanks to all those NASA employees who have made the author's[1] internship a great one. Special thanks to the KSC Engineering Directorate and the Education Office for making this project possible. Thanks to the president of the United States for creating this programs for undergraduates, and for increasing the diversity of NASA employees.

References

[2]Open Source Robotics Foundation, 2015, Robot Operating System (ROS) project, http://www.ros.org/

[3]Open Source Robotics Foundation, 2015, Gazebo simulator project, http://gazebosim.org/

[4]R. P. Mueller, R. E. Cox, T. Ebert, J. D. Smith, J. M. Schuler, A. J. Nick, , "Regolith Advanced Surface Systems Operations Robot (RASSOR)" in *IEEE Aerospace Conference 2013 Proceedings*, IEEE, 2013, pp 1-12 (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6497341&tag=1)

[5]K. Leucht, S. Shelton, L. Moore, C. Mako, K. Strolleis, J. Hecker, M. Moses, M. Nugent, G. Montague, "Autonomous Navigation, Dynamic Path and Work Flow Planning in Multi-Agent Robotic Swarms" in (techport.nasa.gov/externalFactSheetExport?objectId=14799) (https://techport.nasa.gov/view/14799)