Software Design Document
for
CSULA Swarmathon Team

Version 1 approved

Prepared by:
Jonathan Sahagun
Jose Ambrosio
Christopher Portugal
Christian Soltero
Mikey Thong
Advisors:
Dr. Jose Macias
Richard Cross
4/18/2018

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Draft #1	12/04/2017		
Draft #2	12/06/2018	updating	
Final	5/5/2018	Completing first version	1.0

1. Introduction

1.1 Purpose

This document will outline in detail the software architecture and design for the Swarmies. This document will provide several views of the system's design to facilitate communication and understanding of the system. It intends to capture and convey the significant architectural and design decisions that have been made for the Swarmies.

1.2 Scope

This document provides the architecture and design of Swarmathon,

1.3 Intended Audience and Reading Suggestions

This document is written on a technical level to address the CSULA Computer Science department.

1.4 System Overview

Swarmies Software is being built for the UNM/NASA Swarmathon Competition. The operating system being used is Ubuntu 16.04. The software frameworks that Robotic Operating System has to offer, will be utilized to work on the Swarmies. All coding will be done in C++.

2. Design Considerations

2.1 Assumptions and Dependencies

- Related software or hardware: ROS
- Operating systems: Ubuntu 16.04
- End-user characteristics: By competition rules, we are not allowed to change anything pertaining to the GUI. However, the search and retrieval algorithm may be altered.
- Possible and/or probable changes in functionality: There are plans to alter the algorithms pertaining to localization, mapping, navigation, and data retrieval. This should give our Swarmie a better approximation of the perceived world.

2.2 General Constraints

- Hardware Limitations: We are not allowed to alter the state of the Swarmie rover's hardware.
- End-user environment: We are not allowed to alter the state of the GUI.
- Availability or volatility of resources: We are limited on the number of engineers that can be of help. All of which have a finite amount of time to stay caught up on current trends.
- Standards compliance: We must use the ROS version, Kinetic Kame, as that is the version that will be used during the competition.
- Interoperability requirements: All communication between Swarmie's must be done via ROS topics and the ROS master.
- Data distribution requirements: All communication between Swarmie's must be done via ROS topics and the ROS master.
- Memory and other capacity limitations: The Swarmie technically has limited storage on its main processing unit (Intel NUC).
- Performance requirements: Velocity of Swarmie's must not exceed 1 meter per second.
- Network communications: To control a Swarmie from another pc, the Swarmie and the controlling pc must be on the same local network.
- Verification and validation requirements (testing): By competition rules, there are check-ins for the competition that we must meet on time with no exceptions.

2.3 Goals and Guidelines

- The KISS principle ("Keep it simple stupid!")

- The Software has a mandatory delivery date that must be met: 3/20/2018 (Final Code Submission on Github is 3/20/2018)
- Competition guidelines must be followed as stated on the competitions' Github.

2.4 Development Methods

The development method used is Crystal Methods Methodology, developed by Alistair Cockburn. Crystal Method's philosophy was that each team has a different set of talents and skills and therefore each team should process uniquely tailored to it. The Swarmies had ample areas that needed to be improved. Some of the major areas that we attempted to improve were localization, mapping and search algorithm. Our team was divided into separate teams each tackling one of the major areas. The teams were assigned based on skill and interest. We also integrated portions of the Scrum methodology, developed Ken Schwaber. There was a backlog kept of the prioritized work that needed to be done. Items within the backlog were completed or attempted on a weekly basis. Weekly meetings were held to discuss and explain progress as well upcoming work and obstacles raised.

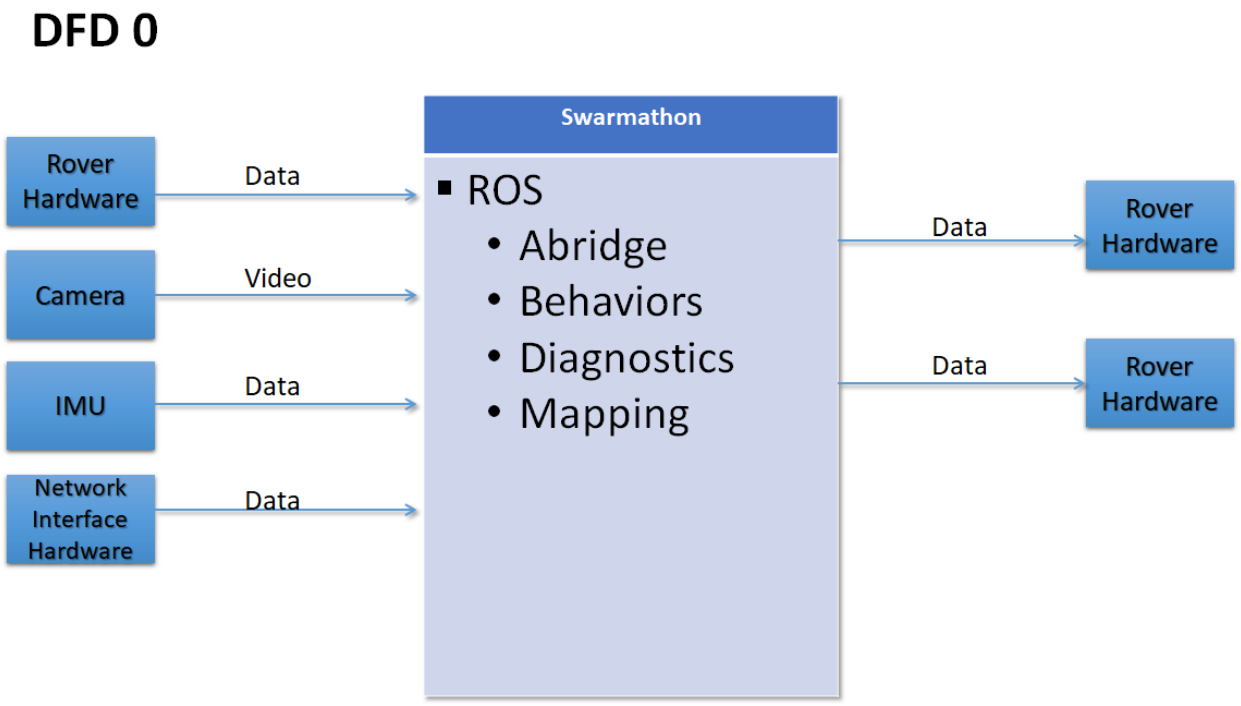
3. Architectural Strategies

- Used solely C++ instead of pairing it with Python because managing and integrating two languages result in slowing down project progress.
- Reuse of existing software components to implement various parts/features of the system: Base software given by competition
- Future plans for enhancing the software: Develop a better search algorithm from the default algorithm given. Make changes in Swarmie's behavior when it identifies a cluster of cubes. Improve accuracy of rover's localization, more specifically regarding its attempt to return to the collection zone.
- User interface paradigms (or system input and output models): GUI and 3D models provided by competition organizers in STL files.
- Hardware and/or software interface paradigms: The competition organizers have given the resources to construct our own Swarmie rovers which can be used to run physical simulations.
- Error detection and recovery: If all Swarmie's don't all connect to the mapping system before it starts then stray(s) will create its/their own mapping system.
- Memory management policies: Since the map used by the Swarmies is a sizable 2D array we avoid doing unnecessary copies to it.
- Communication mechanisms: The host pc communicates with the Swarmie rovers over a local network.

4. System Architecture

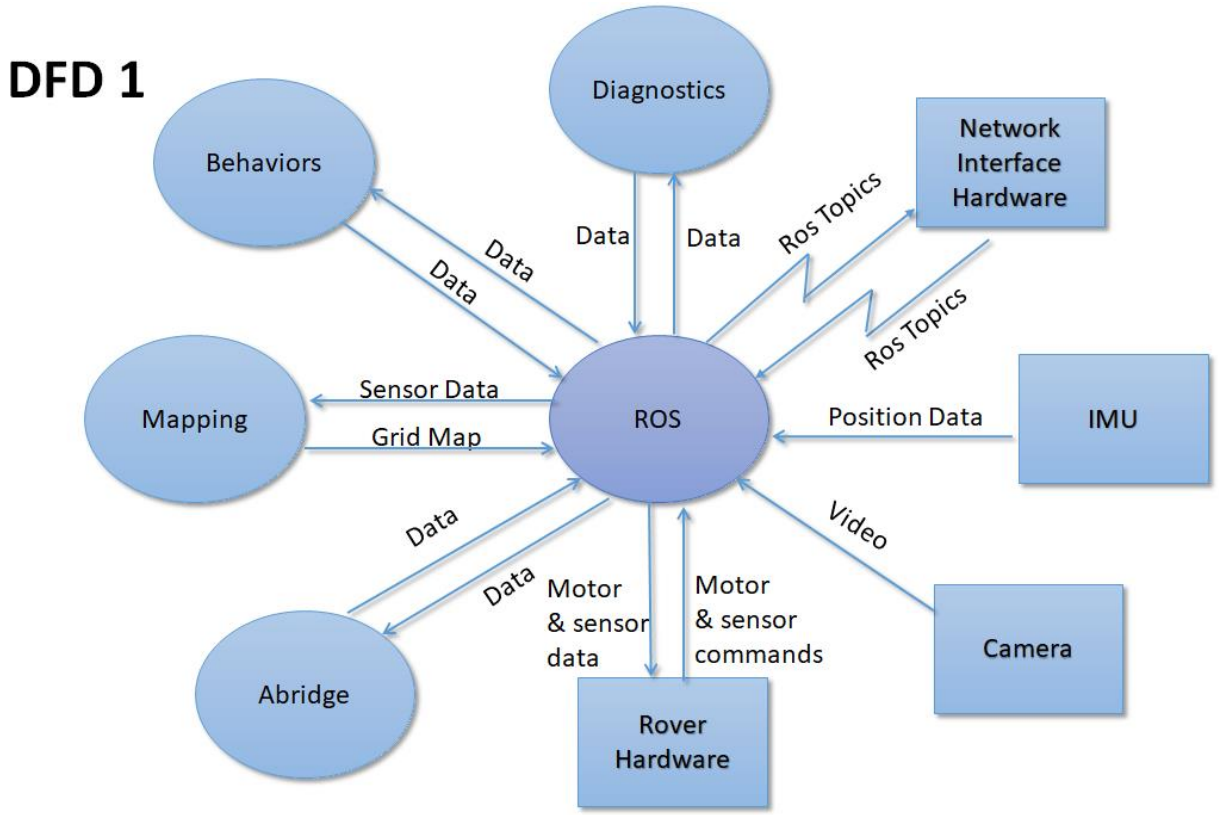
4.1 DFD 0

Rovers are equipped with various sensors to aid in its tasks. The most important sensors are the for input the camera, IMU, motor encoder and GPS. From those inputs ROS is able to autonomously run and complete its tasks. The output is simply data to the motors that move the rover and actuate its claw.



The image above is the DFD 0 of a Swarmie. The Swarmies have four inputs which are from the rover hardware, camera, IMU, and network interface hardware. The input that a Swarmie receives from the rover hardware is ultrasonic and encoder data. Ultrasonic data is used to identify obstacles. The encoder data is used to help track how much a rover has moved. The Swarmie receives video from the camera which the software uses to identify April tags. The Swarmie takes in orientation, angular velocity and GPS data which help identify the Swarmie's position. The last input is network interface hardware which is just a router which ROS uses to receive data from other Swarmies. The Swarmies outputs to the rover hardware are commands to Servos and DC motors. It also outputs data back to the network interface hardware which is just data to other Swarmies.

4.2 DFD 1



4.2.1 ROS

ROS is the core framework that allows communication between all the modules.

4.2.2 Behaviors

The Behaviors module is the logic that runs the automatous mode of the rover. It uses data from retrieved from the hardware deicide what action to take using a state machine to switch between tasks.

4.2.3 Diagnostics

The Diagnostics module allows the program to log the information that is being passed between the modules. It is used to create files and logs to simulate inputs from previous runs, to debug errors, and other diagnostic needs.

4.2.4 Abridge

The Abridge module is the module that reads the hardware data and publishes that data. It takes the data from the IMU, GPS, Camera and the custom Arduino board and creates ROS Topics for each sensor. Not only does this module publishes the raw data, it also publishes filtered data. The data filtering my be done at the Arduino level or in this module.

4.2.5 Mapping

The Mapping module creates and publishes a map using a ROS package called Grid_Map. A grid map is, at its core, a 2D array with values representing some real world object. For example a value of 10 may denote an obstacle while a value of 0 denotes unmapped. As the rover's sensors collect data, this module updates the map; so if the sonar does not ping we can safely say there are no obstacles in front of the rover, and if it does ping, the map is updated to make the obstacle. All rovers connected to the ROS Master shall contribute to the map and the rovers shall use a shared map. This is done by having all rovers publish and subscribing to the same grid map topic.

5. Policies and Tactics

5.1 Choice of which specific products used

(QTCreator, C++)

5.2 Plans for ensuring requirements traceability

A GitHub repository has been set-up for the CSULA-Swarmathon team to ensure traceability.

5.3 Plans for testing the software

Test simulated Swarmie's as often as possible to make sure logic is working as intended. Test actual Swarmie(s) at least three hours a week.

5.4 How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

To generate the system's deliverables, a terminal is used to navigate to the project's root folder the run command: ./run.sh

To build the system's deliverables, a terminal is used to navigate to the project's root folder or subfolders and then run the command: catkin build

6. Detailed System Design

Omitted as the **Section 4.2** sufficiently describes the System Design

7. Detailed Lower level Component Design

Omitted as the **4.2** sufficiently describes the System Design

8. Database Design

Not Applicable

9. User Interface

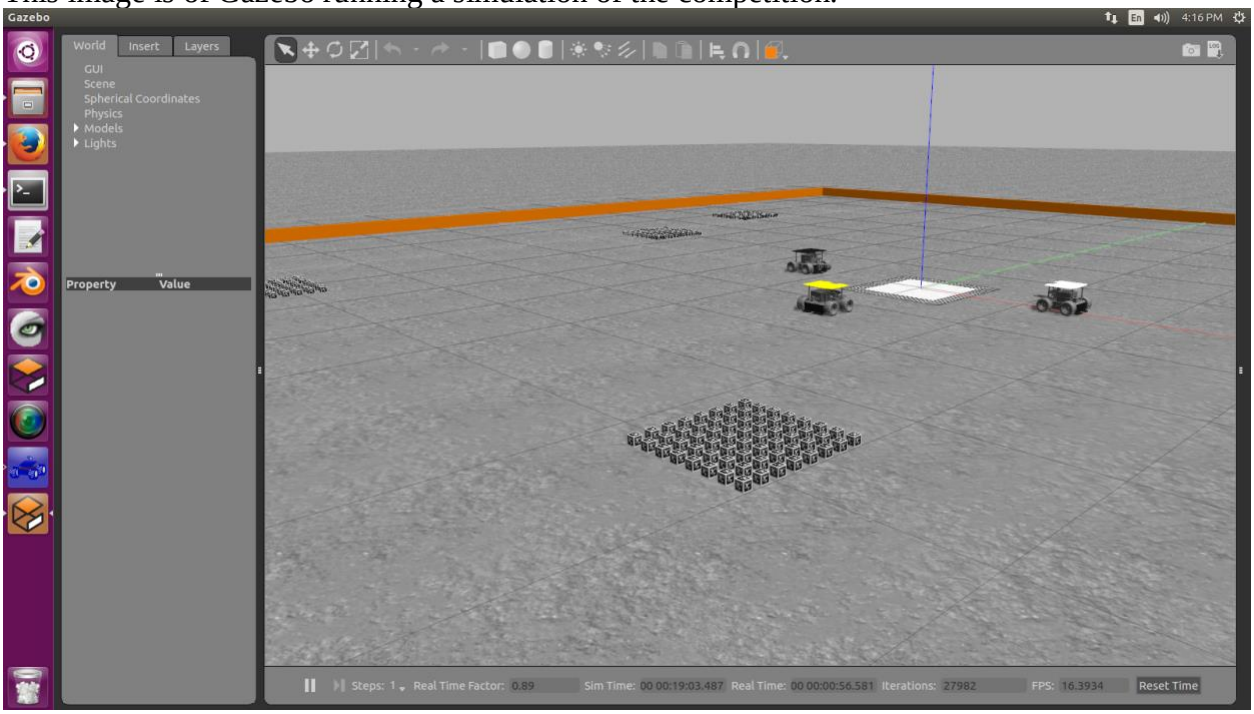
9.1 Overview of User Interface

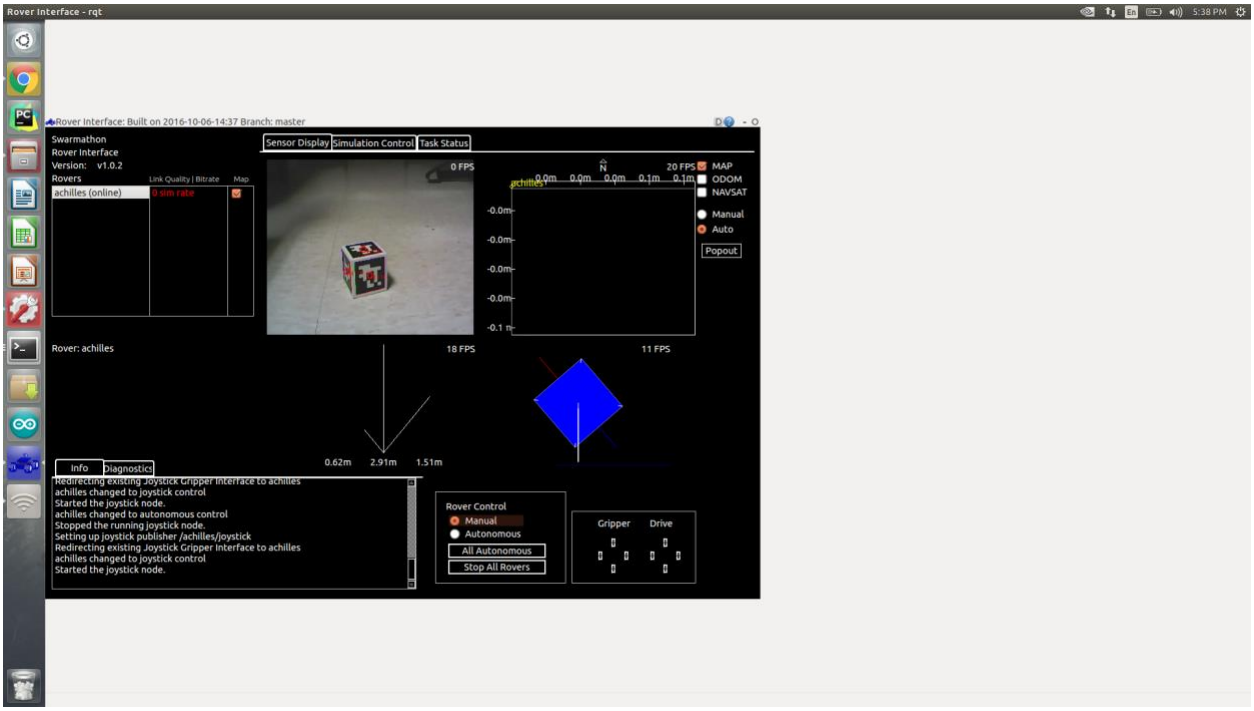
Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

The goal of this software project is to develop an autonomous search algorithm so our Swarmie can search and retrieve april cubes. A user can choose to simulate Swarmie(s) in a virtual environment to observe a demo of the applied search algorithm without having to use an actual Swarmie. The user also has the option of using a real Swarmie in combination with the rover GUI to observe all of the returned data such as camera, ultrasonic, etc.

9.2 Screen Frameworks or Images

This image is of Gazebo running a simulation of the competition.





9.3 User Interface Flow Model

The image above shows the central component user interface. The chart on the top left shows the connected Swarmies to the computer running the GUI. The image in the top center is video feed from the camera. The Graph in the top right is the map of the chosen Swarmie's path. In the center of the GUI there are three lines with a measurement in meters which represents the sonar readings. The cube in the middle of the right side Represents the orientation which is given by the IMU. The text box in the bottom left is information about the selected Swarmie. The bottom center allows the user to choose between the Swarmie's two modes which are autonomous and manual. The bottom right Is the visualization of control inputs when the Swarmie is in manual mode.

10. Requirements Validation and Verification

Requirement	Component Module	Testing
The rover hardware shall receive the motor and sensor data from another rover.	Network Interface Hardware	Grid Map updating
The rover hardware shall send the motor and sensor commands to other rovers.	Network Interface Hardware	Checking the ROS topics
The rover hardware shall transfer the rover's data over a network.	Network Interface Hardware	Checking the ROS topics
The rover software shall detect the Apriltags.	Camera	Adding squares around Apriltags on the camera feed
The rover software shall map obstacles.	Mapping	Checking grid map in RVIZ
The rover software shall receive sensor data.	Abridge	Checking the ROS topics
The rover software shall send grid map data.	Network Interface Hardware	Checking grid map in RVIZ
The rover software shall check whether the sensors are running.	Diagnostics	Checking the ROS topics
The rover software shall check sensors for connection status.	Diagnostics	Checking the logs for error/ Error message in GUI
The rover software shall avoid obstacles.	Mapping	Autonomous rover drive test
The rover software shall avoid pushing the april cubes out the collection zone.	Behaviors	Autonomous rover drive test
The rover software shall keep track of the sensors.	Abridge	Checking the ROS topics
The rover software shall avoid collision with another rover.	Behaviors	Autonomous rover drive test
The rover software shall search for april cubes.	Behaviors	Autonomous rover drive test
The rover software shall pick up april cubes.	Behaviors	Autonomous rover drive test
The rover software shall drop off april cubes in the collection zone.	Behaviors	Autonomous rover drive test
The rover software shall check if the rover is in the collection zone.	Mapping	Autonomous rover drive test
The rovers shall move autonomously.	Behaviors	Autonomous rover drive test

11. Glossary

ROS	Robot Operating System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
STL	Stereolithography
DFD	Data Flow Diagram

12. References

https://github.com/BCLab-UNM/Swarmathon-Docs/blob/master/Competition%20Rules.md