SOFTWARE REQUIREMENTS SPECIFICATION

for

The ARLISS Project

Capstone Group 27
Version 1.0

Steven Silvers
Paul Minner
Zhaolong Wu
Zachary DeVita

Oregon State University
November 3, 2016

Contents

Revision History

Name	Date	Reason For Changes	Version
21	22	23	24
31	32	33	34

1 Introduction

1.1 Purpose

The purpose of this document is to provide a thorough and verbose description of our teams software which we will be developing for use in the ARLISS International Competition; the acronym ARLISS referring to 'A Rocket Launch for International Student Satellites'. The document will explain the purpose and features of the system, as well as, objective of the system, and the constraints under which it must operate. This document is intended for any stakeholders in the project, i.e. clients, instructors, and engineering collaborators, as well as, for the software developers who will be creating the system.

1.2 Document Conventions

This document follows IEEE Format. Bold-faced text has been used to emphasize section and sub-section headings. Highlighting is to point out words in the glossary and italicized text is used to label and recognize diagrams. Requirements will be prioritized into two categories; High-level requirements will be denoted with a heading of 'Obligatory Requirements', while stretch-goals will be denoted with a heading of 'Secondary Requirements'.

1.3 Intended Audience and Reading Suggestions

This document is to be read by the software development team, the extended team of engineers who will be collaborating on this project, course instructors, and the client for the project. The document will be publicly accessible as well, and may be read by anyone viewing our exhibit. The document can, and will, be read by our peers in software development, as well as, engineers of all disciplines.

Overall Description – The general public, as well as, the media may find the overall description to be most pertinent, and to give an adequate and effective overview of the ARLISS Project.

System Features – Developers, as well as, the extended team of engineers who are collaborating on this project will require a comprehensive understanding of the system features in order to integrate their design, and, additionally, for testing purposes.

Nonfunctional/Functional Requirements – The computer scientists developing the software, and the electrical and mechanical engineers developing the hardware will need to utilize, and adhere to, the requirements.

1.4 Project Scope

Our team of computer scientists will be developing software to operate an autonomously navigated satellite to a specified set of coordinates during the international competition. Our team's satellite will be ejected from a rocket at approximately 12,000' AGL, and fall safely to the surface of Earth with the aid of a parachute. During the estimated 15 minute "hang time" that the rocket spends in the air, there will be barometric reading taken every 1000ft by the satellite. Once the satellite reaches the ground it must detach from the parachute and circumvent any potential restraint created by the canopy or suspension lines of the parachute. Once moving, the autonomous satellite must utilize its system of cameras and radar to detect obstacles so that it may navigate around them. GPS will be necessary for the satellite to find the coordinates.

The most imperative objective of the software is to safely guide the satellite to its intended destination. This is a requisite, but the secondary objective of taking barometric readings is an additional goal which our group is confident in completing.

1.5 References

2 Overall Description

2.1 Product Perspective

This is a new and entirely self contained product. The software is composed of multiple independent systems which will be used at different steps in the satellite's mission. For example, once the satellite has been deployed from the rocket, the parachute deployment system will be used. These systems will be deployed in a specific order, and the timing of the deployment will be based on various sensor readings determining if the time is right to deploy the next system. The only user interface would be the graphing software used to display barometric readings obtained by the satellite during it's descent. The software would interface with many hardware components, including cameras, thermometers, servos, and various sensors. These components will serve as the software's eyes and ears. Memory and power will be constrained, so the software must be as memory and power efficient as possible.

2.2 Product Functions

The software has four main functions, which it performs at each stage of the mission. Each function is described in greater detail in section 3. Parachute deployment, change to drive mode, sensor and motor array initialization, and obstacle avoidance.

2.3 User Classes and Characteristics

The software as a whole is only useful in completing the specific challenges our specific satellite has to complete, although the individual parts of the software may be useful for other applications. Autonomous driving in particular is an upcoming field, so users who wish to create autonomous vehicles may find use from our obstacle avoidance system. Unfortunately, our obstacle avoidance system is designed specifically for navigating dry lake beds, so it's general use is limited.

2.4 Operating Environment

Since the mission is being performed only once in a specific location, the software will be optimized to work best in those conditions. The mission takes place in Black Rock Nevada, in a dry lake bed. This means the navigation software needs to deal with rocks, cracks, and tire tracks, but not much elevation change. Therefore, the navigation

software will only work properly in locations similar to Black Rock Nevada. The rest of the software functions could work correctly in other conditions, however.

2.5 Design and Implementation Constraints

Space is a precious commodity on this satellite, therefore, the least amount of hardware possible will be fitted to it. This limits resources such as power, computation speed, and memory. This means the software will need to be implemented as simply and efficiently as possible while still being adequate to complete it's task. In addition, the autonomous navigation software must act quickly. If not, the satellite will get stuck, or hit an obstacle before the software can correct course to avoid it.

2.6 User Documentation

There will be a manual included with the satellite in PDF format, which details what each function of the software does, as well as it's limitations. For example, in the Obstacle Avoidance section, the manual will list the types of obstacles the software recognizes and avoids. In addition, the code itself will be well commented, so that future developers can understand what it is doing.

2.7 Assumptions and Dependencies

Since the satellite is being designed at the same time as the software which runs it, hardware specifics aren't known. We have a general idea of what the rover should be when it's finished, but specifics such as the placement of sensors, or even which sensors will be included, isn't currently known. As the project progresses, more details will be revealed to us, and we should be able to adjust the software accordingly.

3 External Interface Requirements

3.1 User Interfaces

Because of the CanSat operates autonomously in such extreme condition(s), the team have made the decision on that the CanSat will not have a user interface at this time being, but it is possible to make a user interface in the future for the testing stage and during final competition. The team evaluated all possible solutions based on the project goals, such as being cost-effective, efficient and weight/space constraints to make the decision.
A user-interface would be useful during the testing stage, a user interface displays the comprehensive live data that allows engineers to do quick analyze and have the full control of the CanSat.
A user-interface would be a plus if we can have live data display during the final competition, although per the CanSat operates autonomously, it is unnecessary to have a user-interface and will add on another communication layer, due to the weight/space constraints, the team will evaluate the possibility of having a user-interface upon the completion date of the project.

3.2 Hardware Interfaces

Due to the mechanical and electrical team are still under their early modeling and developing stages, we have not yet gotten any hardware specs, it is fairly straightforward to the team that there will be microprocessor(s), memory, buses, micro-controller(s), sensor(s), I/O device(s) and motor(s) involved. The hardware interfaces runs parallel with several electrical connections carrying parts of the data simultaneously.

3.3 Software Interfaces

This CanSat will be implemented with object-oriented language(s). The team may have to design a special operating system for the CanSat, after that the team will create different classes to program the CanSat finds and goes to its target.

3.4 Communications Interfaces

A circuit board will perform most of the communications interfaces tasks in the CanSat. Communications interfaces will carry high level of intelligence and pre defined protocols in order to achieve the hardware and software parallelism.

4 System Features

4.1 Parachute Deployment

4.1.1 Description and Priority

This system will detect when the satellite has been deployed from the rocket and activate the parachute module to carry the CanSat safely to the ground.

4.1.2 Stimulus/Response Sequences

This subsystem will respond to being released from the rocket, after-which it will deploy the parachute module.

4.1.3 Functional Requirements

This system just needs to activate the on-board parachute.

4.2 Change to Drive Mode

4.2.1 Description and Priority

This system will need to get the rover ready to drive once it has landed on the ground. This involves activating various servos within the CanSat to expand the rover from the payload container. In addition, the can shaped fairing will be shed from the rover.

4.2.2 Stimulus/Response Sequences

Once the CanSat's sensors have determined that it has landed on the ground, the software will then activate this module.

4.2.3 Functional Requirements

This system needs to get the rover out of the soda can container and expand the rover's treads to the ground so that it is ready to start moving.

4.3 Sensor and Motor Array Initialization

4.3.1 Description and Priority

The driving sensors and motors need first time initialization before it can begin driving. This would include locking on to the target GPS coordinates, and preparing the sensors needed for the obstacle avoidance system.

4.3.2 Stimulus/Response Sequences

This process will begin the moment that the previous drive mode system completes its task.

4.3.3 Functional Requirements

The GPS location needs to be determined, and the sensors which will be used for the obstacle avoidance system need to be initialized.

4.4 Obstacle Avoidance System

4.4.1 Description and Priority

Once the rover is locked on to the target GPS location, it will need to autonomously drive to the target location. This could be anywhere from less than a kilometer to fourteen kilometers depending on how the parachute is carried. There will be obstacles between where the rover lands and the target location such as rocks and tire tracks that will need to be avoided by our rover. This system will take input from the rover's on-board sensors and use that data to control motor function so that the obstacles will be avoided.

4.4.2 Stimulus/Response Sequences

As soon as the sensor and motor array initialization completes, this system will begin. The system will be driving the rover towards the target GPS location, unless it it is responding to a sensor detecting an obstacle. These obstacles include rocks, tire tracks, and other impassable terrain. The software will respond to the obstacle by finding an alternate route avoiding that obstacle.

4.4.3 Functional Requirements

Drive the rover to the target GPS coordinates while avoiding any obstacles that impede the rover's progress.

4.5 System Feature 5 (and so on)

4.5.1 Description and Priority

4.5.2 Stimulus/Response Sequences

4.5.3 Functional Requirements

5 Other Nonfunctional Requirements

5.1 Performance Requirements

The speed at which this software reacts is of crucial importance for the obstacle avoidance system. If the software doesn't react to it's surroundings fast enough, the rover will hit obstacles before it has a chance to avoid them. Therefore, the software must be able to react to obstacles faster than the rover can move.

5.2 Safety Requirements

This software is entirely safe. The rover is unable to do any kind of damage to any person or object in it's vicinity.

5.3 Security Requirements

Security is not a concern for this software. The rover on which this software is running will only be deployed once, and the software itself is very specific for a single task.

5.4 Software Quality Attributes

This software must be thoroughly tested to ensure it will be able to complete it's task on the first try. Therefore, the software will be tested both by simulating the environment it will be running in, and using the actual rover in environments like where it will be deployed. The simulations will be used to ensure the software behaves as inteded when faced with obstacles such as rocks and tire tracks, while the rover tests will ensure the software can detect real obstacles using the provided sensors on the rover. x

5.5 Business Rules

Each system in the software acts independently from the other systems. The parachute deployment system must activate first, followed by the drive mode, sensor and motor array initialization, and obstacle avoidance system. Each system has a specific task to perform, then it ends and a new system begins.

6 Other Requirements

6.1 Stretch Goals

6.1.1 Data Visualization

A stretch goal for this project would be to develop a system to visualize data sent back from the satellite to a remote control station. This would include data from the satellite's flight, as well as its trip across the desert.

6.2 Appendix A: Glossary

6.2.1 AGL

Stands for "above ground level"

6.2.2 ARLISS

A collaborative effort between multiple institutions to build, launch, test and recover prototype satellites miniaturized to fit inside a soft drink can.

6.2.3 CanSat

Name given to the prototype satellites used in the ARLISS competition.

6.3 Appendix B: Analysis Models

6.4 Appendix C: To Be Determined List