The ARLISS Project
Design Document
CS 461

Steven Silvers, Paul Minner, Zhaolong Wu, Zachary DeVita
Capstone Group 27, Fall 2016

Abstract

This document defines the plan for how the various pieces of the ARLISS Project will be developed and implemented. Each piece that was discussed in the previous technology review will be gone through in detail in its own section.

Contents

I. Introduction

In this document we have divided our project into twelve components, and a framework for each one of these components has been designed by the corresponding member from the list below. Each section will detail how a component will operate, and how it will interact with other related components. This document is intended to provide a structure, and the architectural layout for the entire project.

Project Framework
 Zachary DeVita
CMOS Image Sensing
 Zachary DeVita
Ultrasonic Radar
 Zachary DeVita
Obstacle Avoidance
 Steven Silvers
Control Board
 Steven Silvers
Motorized Tracks
 Steven Silvers
temp name
temp name
temp name
temp name
temp name
temp name

II. Framework

For our overall project design and framework, our team has decided to utilize the C++ programming language. We will be writing our individual portions of code using a C compiler in order to reduce the overall overhead and energy cost of the program, but we will eventually combine the portions into a C++ framework where we will be able to utilize classes and access modifiers. The team feels strongly that utilizing classes and access modifiers for a project of this magnitude would be a huge benefit, and, since C is a subset of C++, we can get the best of both programming languages. Using C++ compilers and C++ specific tools will also reduce time spent debugging and testing [1].

As far as the framework for the project, we will need a series of classes to pass control of the satellite between. This is due to the fact that our satellite's journey will be comprised of four primary stages where the controls for the satellite are vastly different from each other. All of these classes governing stages of control should have a private modifier so they can't be accessed by other classes.

The first stage of the satellite's expedition will be when it is encapsulated in its can-shaped housing. This stage will take place during the period of time that the rocket is on the ground prior to launch, and it will last until some period of time after the satellite has landed safely to the ground via the parachute. Though the satellite will remain in a static state during this period, it must turned on and ready to receive instructions.

There will be a class designated for the control of the satellite for this period of time. The satellite will simply be listening for interrupts from other classes. There will be an interrupt that tells this class when to deploy its parachute. This interrupt will be provided by another class, and it will be based on some form of a timer. This event should occur shortly after the satellite has been ejected from the rocket. Another interrupt will tell the satellite to pass control to the class responsible for the next control stage of the expedition. Both of these classes with the interrupts should have a protected modifier so they are only visible to the class which has control of the satellite. This event should occur shortly after the satellite has been ejected from the rocket. This event should occur shortly after the satellite has landed safely on Earth's surface.

The second stage of the satellite's expedition will encompass the majority of the satellite's expedition, and there will be a class written specifically for the segment. This class will be responsible for the period of time that the rocket navigates from its landing zone until it has reached approximately 8-9 meters from the pole. The pole marks the final destination of the satellite, and the satellite must physically come in contact with the pole to finish the competition. The reason for the 8-9 meters is due to the fact that GPS is only accurate at 7.8 meters with a 95% confidence interval [2].

During this period there will be many other classes interacting with the class retaining control of the satellite. There should be a class which tells this control class if there is obstacles in the way and what direction to move to avoid them. That class should be a protected class as well, and that class should have a parent/child relationship with each class governing the sensors being used to identify obstacles.

There should also be a class which governs the GPS sensor. This class should help direct the satellite on a macro level, not taking into consideration any obstacles at the local level. This protected class should have its navigation instructions be overridden in the case of an obstacle in its path. This class will also send an interrupt to the control class when the satellite has reached 8-9 meters of the pole which will tell the control class to pass off control to the final governing the satellite's navigation

. The third stage of control will be reached only in the case where the satellite becomes stuck or on its side. The satellite design is being made specifically to preventing this type of situation from occurring, but there is still a chance of it happening. There will be a specific routine devised for this circumstance, and control of the satellite will be passed to this stage in this type of event. This class will have access to the CMOD, ultrasonic sensors via inheriting from these protected classes. When the satellite has recovered from the event, and some routine to avert the obstacle has been completed, then the control will be returned to the previous control stage.

The final stage of control will cover the period of time when the satellite reaches the 8-9 meter range from the pole, and it will last until contact with the pole has been made. The class responsible for the satellite's control over this period will no longer take input from the GPS. The system used previously for the obstacle avoidance system, i.e. the CMOS imaging sensor and the ultrasonic sensor, will be used here for navigation. Even if the ultrasonic sensor is operating at 40kHz, considering the target has a spherical shape, the sensor will be only accurate for up to 20.2 feet so the CMOS imaging sensor will have to suffice until the satellite is close to the pole [3]. The CMOS imaging sensor will need to switch modes in this stage to look for a pole shaped object and direct the satellite towards it. A separate protected class should be governing the sensor in this stage of navigation. Instead of looking for obstacles the sensor should be searching for an object which is similar to some stored image, allowing for an appropriate level of error.

III. CMOS Image Sensing

For the ARLISS project we are using a combination of a CMOS imaging sensor and at least one ultrasonic sensor in order to detect and avoid obstacles. The path of the satellite, on a macro level, is determined by the GPS sensor, but modifications to the path will need to be set locally using this system.

The idea behind the imaging sensor is that, when there is an abrupt change in color in an image, especially when referring to nature, it often shows that there is an abrupt change in the depth being viewed. For our purposes we want to use this observation to detect objects in the satellite's path, as well as, changes in terrain like a steep incline or a steep decline.

Manipulating raw camera frames can require a relatively large amount of space, as well as consume a large amount of energy. Energy is the most valuable resource for our project because we have a strict limit to the size and weight of the satellite so we have a very limited space for a battery to fit. There are several methods which we must use to minimize this cost, and there is one critical library in particular which we will use to help conserve this limited battery power.

As far as libraries go, we will be utilizing the cv.h library provided by OpenCV. This open-source library provides the capability of image and video I/O, image processing for individual video frames, as well as, built-in object recognition functionality [4]. All foreseeable video manipulating should be able to be done with this single library.

The plan for reducing energy consumption is simple; first we will take a frame of raw footage as input from the CMOS device, then we will convert it to grayscale, and from grayscale we can convert it to binary data. Manipulating this binary data will consume far less energy than performing complex algorithms on the original, raw video footage [5].

Using the OpenCv library we will be able to capture video and directly load each video frame from the sensor to an IplImage object. OpenCv also has a built-in function called cvCvtColor() which will convert each image to grayscale. There will then need to be an algorithm which will create a digital map of the image using a two-dimensional array, and which will assign values of 1 or 0 depending on the difference in color change between pixels [5]. When objects or abrupt changes in elevation are detected we can combine data from the ultrasonic sensor to determine the distance of the object, and modify the satellite's route to avoid the obstacle.

This functionality of importing the live video feed, converting it to frames, and applying functions to the data to convert it to binary will need to exist in its own class. This class will be responsible for the flow of the input video data from the sensor, and outputting the data as an array of binary values to a separate class. The output from this class will combined with the output from the class responsible for the ultrasonic sensor data, and this combination of data will be used elsewhere to help determine what sequence of events needs to occur in order to safely navigate the terrain.

IV. temp

V. Continuous Tracks

For the method of how we would travel along the ground, the team members of the ARLISS Project decided to go with the continuous track, or tank tread, method of traveling.

VI. Control Board

VII. Obstacle Avoidance

As detailed in the CMOS Image Sensing section, the data that we will be working with to detect and avoid obstacles will be a 2D array of binary values. A value of '1' will represent a change in the gray scale image, which will ultimately outline the terrain and give an easy to traverse dataset with which to avoid obstacles with. A value of '0' will represent no major change in shade of gray between neighboring pixels of the image, meaning that this will either be the interior of an object bounded by '1' values, or empty space in front of the camera.

The biggest concern and challenge facing our satellite project is the conservation of power and limiting power usage. Due to the space constraints of the competition our batteries will be smaller than desired for this kind of journey so we must conserve where we can. One way that we are doing this within the obstacle avoidance system is by not continuously taking in video feed from the camera. The setting where our system will be tested is for the most part very flat, meaning we can realistically take a new image to analyze after a short time interval without having to worry too much about a new obstacle appearing before we can avoid it. The exact time interval is not yet known, as it will take physically testing the satellite system to determine the correct value to use.

The way that the 2D array will be used to avoid obstacles is by examining the values by column from left to right to detect if an obstacle has appeared in our path. We expect there to almost always be a row of solid '1' values along the bottom of the 2D array that represents the horizon, where the ground meets the sky in the cameras eye. The algorithm for avoiding an object is quite simple, if a column or close to a column of '1' values appear towards the right side of the array, we stop forward motion, and rotate counterclockwise while periodically stopping to take a new picture to see if we have angled around the obstacle. once we have a picture showing no obstacles, we travel forward in this direction for approximately five seconds before going back to driving towards the target GPS coordinates. If the obstacle appears on the left side of the 2D array, we do the same thing except rotate clockwise.

Once we have the satellite built and the CMOS imaging implemented with the gray scale to 2D array conversion working, we will be able to test our satellite in various conditions and with different obstacles to see what kinds of obstacles we can drive over and ignore, and what obstacles must be driven around. During this testing period we will also tweak the timing at which how often a new picture will be taken and analyzed. The target for this testing is to find the time cycle at which we can take the fewest pictures while still avoiding all of the obstacles that we need to, and thus saving battery usage by the camera.

VIII. TEMP

IX. TEMP

X. TEMP

XI. TEMP

XII. TEMP

XIII. TEMP

References

[1] jain.pk, "Using inline assembly in c/c," Oct 2006. [Online]. Available: http://www.codeproject.com/articles/15971/using-inline-assembly-in-c-c

[2] "Gps accuracy," Oct 2016. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[3] D. P. Massa, "Choosing an ultrasonic sensor for proximity or distance measurement part 2: Optimizing sensor selection," Mar 1999. [Online]. Available: http://www.sensorsmag.com/sensors/acoustic-ultrasound/choosing-ultrasonic-sensor-proximity-or-distance-measurement-838

[4] G. Agam, "Introduction to programming with opencv," Jan 2006. [Online]. Available: http://cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

[5] "Cmos camera as a sensor." [Online]. Available: https://www.ikalogic.com/image-processing-as-a-sensor/