# The ARLISS Project
# Progress Report
# Senior Capstone

Steven Silvers, Paul Minner, Zhaolong Wu, Zachary DeVita
Capstone Group 27
2016-17 Academic Year

**Abstract**

This document details our progress while developing software for the ARLISS competition over the duration of the academic year. This includes the purpose of the software, goals, current progress, problems encountered, and plans for the future.

## Table of Contents

## I. ARLISS Project

### A. Description

For our capstone project, our team of computer scientists have spent the course of three terms developing software to autonomously navigate a rover to a specified set of coordinates. This rover is intended to compete in the ARLISS competition held in September. ARLISS, standing for ”A Rocket Launch for Student Satellites,” is an international competition where teams of engineers from around the world will bring their rovers to compete. To compete in this competition the rover must be small enough to fit into a soda can and must have a weight of less than or equal to that of a soda.

In this competition, our team's rover will be launched by a rocket into the air, and will be ejected from the rocket at approximately 12,000′ AGL where it will use a parachute to fall safely to the surface of the Earth. Once the rover reaches the ground it must detach from the parachute and circumvent any potential restraint created by the canopy or suspension lines of the parachute. When the rover is free from the parachute and begins its expedition, the rover will rely on GPS to direct it to the coordinates while using the onboard camera to detect obstacles in its path. GPS is only accurate for approximately 7.8 meters so, when the rover is near the end of the expedition and GPS becomes no longer accurate, the rover will begin to search for the pole that marks the end of the trek. At the base of the pole is an orange traffic cone which is used for the image detection algorithm. When the cone has been detected the rover will drive towards it until contact has been made.

As the computer science team, our job is strictly developing the software for the rover. There have been five primary components devised for the implementation of the software.

- Parachute Deployment
- GPS Navigation
- Obstacle Avoidance
- Getting Unstuck from Obstacles
- Locating and Coming in Contact with the Pole

### B. Goals

The primary objective for this project is to compete in the ARLISS competition, and to have our rover finish its autonomous expedition by bumping into the traffic cone at the base of the pole which marks the finish. No team in the ARLISS competition history has yet completed this objective in the specific competition we are attempting. What distinguishes this competition from the other ARLISS competitions is the soda can restriction on the size and weight of the rover. Because of this, and the fact that we must rely on the performance of two other engineering teams, we have an alternate metric of success. If the rover is able to safely land on the ground, escape from the parachute, and successfully traverse the terrain for some period of time while navigating around obstacles, then we would consider our goals met. Battery life not lasting for the duration of the rovers trek is beyond our control, as is mechanical drawbacks or limitations. As computer scientists, our measure of success is based solely on the success of the five components listed above.

## II. Current Progress

### A. Zachary DeVita

I spent most of my spring break working on the traffic cone detection software. After many failed attempts at producing a working and effective machine learning classifier, I decided to develop a backup program for detecting traffic cones. My backup algorithm for detecting traffic cones used a complex series of image filters and counting convex hulls, and was able to eliminate noise and locate a traffic cone based on both its geometric features and its orange color. The major flaw with this method is that each time the frame goes through a single filter, each pixel in the image must be iterated over at least once, and this algorithm required many filters.

At the same time, while I spent my break developing this algorithm and testing the parameters for each of the filters, I also continued to search for the optimal machine learning classifier. I eventually did develop an amazing classifier and the way I did this was simple. There are a couple of algorithms commonly used for developing a classifier. The Haar algorithm is the most effective, but it requires a period of days to weeks of running to develop a single classifier. The LBP (Local Binary Pattern) algorithm produces a classifier with considerably more false positives, but only requires a few hours of running to develop. Both of these algorithms require almost identical parameters. So, what I did is I found the most optimal classifier that I could produce using the LBP algorithm, and then I simply used the same set of parameters for training my Haar classifier.

With the driver for the classifier, I was able to continue testing. I developed a simple system to determine what direction the rover needed to turn when the cone is located. This involved locating the center of the rectangle image object containing the traffic cone. The biggest problem I found with the initial implementation was that there were still some false positives being detected. From my research I found that this is impossible to avoid with any machine learning classifier, and further code would be necessary to verify the results.

A Haar classifier is purely a feature-based classifier which uses geometry and shape patterns to recognize an object. The images used to train the classifier are all converted to grayscale before the training begins so the color of the object is not taken into consideration. This is where a feature from my backup traffic cone detection program came in handy. Since the bright orange color uniquely identifies a traffic cone, and the color of a traffic cone is an atypical color to be observed in nature, I decided it would be best to verify results by checking that the color of the object being detected is orange. I had used a method of converting pixels to their HSV (Hue, Saturation, & Value) value equivalent to verify an object's color in my backup program, so it was a natural solution to this problem. I added a verification to my algorithm to check three points, or pixels, from the result by converting only those three points to HSV, and then checking if they are in the appropriate range of values which would be observed on a traffic cone. This method seems to work pretty well!

I continued to work and test both of my computer vision algorithms throughout the first half of the term, making little improvements. One step that I added to the obstacle avoidance algorithm was using a morphological opening conversion to reduce additional noise in the image. This specific transformation is referred to as a morphological opening and is defined by the image erosion followed by an image dilation. The erosion effect removes pixels near the boundary of objects in the image so the thickness or size of the foreground object decreases.

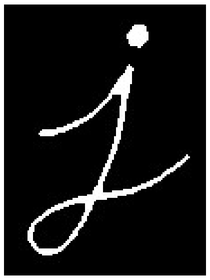
Fig. 1: Image with Erosion

Dilation does the opposite. Whereas erosion decreases the white region of the object by narrowing the boundary, dilation increases the white region in the image by increasing the boundary.

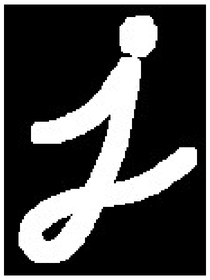
Fig. 2: Image with Dilation

What the combination of the two effects does is it removes noise in the image with the erosion, then brings the remaining nodes from the image back to their original size. Nodes which are small get removed during the erosion state.

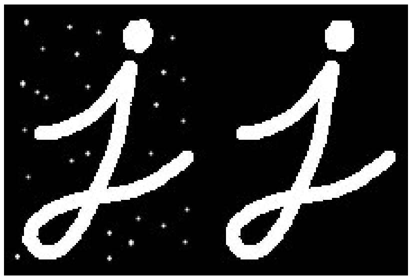
Fig. 3: After Morphological Opening

The most recent improvement to the traffic cone detecting algorithm has been targeted at eliminating the remaining false positives. We seem to get an occasional false positive at points in an image where light is reflected directly into the camera. I increased the size, i.e. number of pixels, of the minimum accepted result in the algorithm. These false positives are so small that I cannot physically identify the shape of a cone in the image at the location or identify any orange coloration, so I believe that enlarging the minimum accepted result will completely eliminate this issue.

### B. ***Paul Minner***

Over spring break, I worked on speeding up the obstacle avoidance system. Initially, the obstacle avoidance system worked, but took over 30 seconds to check single image on the pi. This wasn't good enough, because we needed to be able to take new pictures every few seconds in order to find obstacles before hitting them. I realized the algorithm was slow because the filtered image was being written to disk, then being read back in to actually check for obstacles. I just stored the image values in an array so we wouldn't have to read and write from disk, which sped up the algorithm about 10 times. Now, the algorithm can check a new image roughly every 3 seconds. If this is still too slow, we can also reduce the image quality, but 3 seconds should be fast enough.

After spring break, I started working on integrating the ECE team's device controller code into our motor control code. Before, all our motor and servo functions just printed what they would do, but now they should actually move motors and servos. The ECE team made this relatively simple for me by providing documentation for each of their functions and providing me a single source code file and header. The only difficulty I had was that their code was written in C, while our code is written in C++. I had to compile their code using gcc instead of g++ and surround their header file code with an extern "C", and surround that extern with an #ifdef which checks if the code is C++ or C because the header file needed to be read by C code and C++ code.

In order to remain being able to test the code on our own machines, I created two separate compilation options, test and final. Final includes the device controller code, while test just prints output like before. To do this, I needed to modify the makefile, so I also streamlined the makefile to automatically compile each source file without separate options in the makefile. I was able to test that the device controller code worked by putting the code on the unfinished rover and running it. Unfortunately, not all the motors work on the rover, so I could only test whether the motors moved or not. I was unable to actually move the rover.

For the code freeze, I also cleaned up the repo, as well as created a better Readme file with compilation instructions. I haven't worked more on the three modules I am responsible for, which are parachute deployment, getting unstuck from obstacles, the driving portion of finding the finish pole, because we still don't have a working rover to do testing on. Very likely, some of these modules will need to be adjusted once we have a working rover, but at the moment they do work with the simulated values I created to test them.

### C. ***Steven Silvers***

Major progress was made this term on the control board module of our project. At the start of the term we had a prototype PCB that was developed by the ECE team, but ultimately was too large for the rover and would not work for the project. Designs for a smaller PCB were developed and sent off for manufacturing and the finished boards arrived during week five of the term. Connecting sensors and Raspberry PI to the PCB was completed during week six and will be installed in the rover before EXPO.

The obstacle avoidance module had some improvements done at the beginning of the term as detailed in the plan from our previous progress report. The use of a csv file to store a binary array was removed to reduce file I/O operations, which drastically improved the performance of the module on the Raspberry PI Zero. A few OpenCV filters were also removed from the module as they were not being used in the final product. The obstacle avoidance module now runs quickly enough for use in a moving rover, and is currently awaiting completion of the rover for final testing and tweaking of the algorithm.

### *D.* ***Zhalong Wu***

Over the first half of the Spring term, the major progress I made is updating the rover's navigation function. Before the Spring term I was told that we will have a compass installed on the rover so we won't have to deal with finding rover's heading, which suddenly the ECE team told us we will only have a GPS module and an accelerometer for navigating the rover around, this became a problem because we could not know the rover's initial heading when it first landed on the ground. This problem bothered me a while and eventually I came up with the idea that as soon as the rover landed, the rover's X,Y coordinates will be recorded, then the rover will be ordered to travel a certain amount of distance that is enough for the GPS module to detect the movement, then record rover's coordinates again and compare these two sets of coordinates to get rover's heading as an initial reference. After that use destination heading - current heading will return the rover's travel heading. This heading function will be ran in time intervals to ensure the heading accuracy.

Other than updating the navigation function, I also encountered another problem with servo. Initially in the landonside module, I was letting the servo to rotate 360 degrees when the rover lands on its side, so rotate 360 degrees will for sure rescue the rover, but somehow I found that the servo we have only has a 180 degrees travel, which I had to update my function to make decisions on which side the rover lands on, where I use the accelerometer data to determine and have some if/else to make the decision. This module has never been tested because we don't get the servo driver function from ECE team, even they said they already implemented it at the beginning of the term. I also had to write our own servo function to make every servo features we need on the rover.

## III. Plans for the Future

At this point, each of our project code module is theoretically functioning correctly, and they should work together. The only concern for now is because of ECE and ME teams constantly misses the project deadlines, we don't get the physical rover to play with, which led we have zero chance to run the code on the rover to do further testing. By far we heard that the ME and ECE team will have the tentative rover completion date on May 15th, which will roughly give us 4 days to perform some fundamental tests on the rover, so our tentative testing plan is:

TABLE I: Tentative Plan for the Future term and beyond.

| | |
|---|---|
| 1 | Test to make sure the navigation function works flawlessly with all features. |
| 2 | Test the obstacle avoidance function on agileness and accuracy, the placement of camera |
| 3 | Test the cone recognition module with different backgrounds and terrains the narrow down the HSV value for the best detection. |
| 4 | Observe the rover's moving behaviors to make software compensation if necessary. |
| 5 | Test to make sure every modules work together and get the rover ready for the competition. |

For the coming up engineering EXPO, we are planing to show off our project with its highlight features. First we are going to show that the rover is capable to move around, then we hope to show the obstacle avoidance and recognition feature, last and most importantly, the finishing pole detection feature where on the EXPO day we will bring a traffic cone on site to let the rover to find, and by hock up the rover to a monitor that we can show off this feature to people within the rover's perspective.

## IV. Problems Encountered

The largest problem encountered by our group at this point is the missed deadlines by the ECE and ME teams on the project. The original time line of the project gave us a completed rover at the start of spring term, giving us seven weeks to test and implement our software system before engineering EXPO. With the setbacks that have happened it now looks like we will have approximately three to four days to implement and test our software system on a completed rover before engineering EXPO.

The first problem encountered was the ECE team attempting to test the motor control of their prototype PCB, but the motor housings of the rover were too tight to freely allow the motors to rotate, causing them to accidentally fry a motor. This led to the ME team having to build an entirely new rover frame and wheel housing, as the old motor could not be removed without destroying the old frame as well as order a replacement motor. When the ECE team finally got their final PCBs delivered, while installing sensors they fried a GPS sensor and a Rasperry PI. A new GPS sensor had to be ordered and they had to take the CS teams' Raspberry PI as it was the last one the group had without having to order a replacement.

## V. Code Samples

Computer vision provides an integral component in the success of the rover on its expedition. There are two separate and distinct functions which are performed using computer vision. While the rover is being directed by the GPS, computer vision will be used to detect obstacles in the path of the rover. This is accomplished by using a series of filters which are primarily intended to eliminate noise and to detect the edges of the obstacles. The other function of the computer vision is to identify the orange traffic cone. Our team has developed a complex system which utilizes machine learning and image processing to carry out this task.

### A. *Obstacle Avoidance*

The obstacle detection algorithm our team has implemented requires that each frame recorded by the camera is run through a series of filters which are intended to eliminate noise in the image and to detect the edges of the obstacles in the rover's path. Below is the algorithm which has been developed.

```
/** @function checkObstacle */
void Obstacle::checkObstacle()
{
  Mat frame, blurred, gray, canny;

  if (!DEBUG) {
    //Capture frame from camera
    VideoCapture capture(0);
    capture >> frame;
  }
  cvtColor(frame, gray, CV_BGR2GRAY);
  blur(gray, blurred, Size(40, 4));

  //  Morphological opening (remove small objects from the foreground)
  erode(blurred, blurred, getStructuringElement(MORPH_RECT, Size(5, 5)));
  dilate(blurred, blurred, getStructuringElement(MORPH_RECT, Size(5, 5)));
  dilate(blurred, blurred, getStructuringElement(MORPH_RECT, Size(5, 5)));

  Canny(blurred, canny, 50, 120, 3);

  int Image_Array[Width][Height];
  for(int i=0; i<canny.rows; i++)
    {
      for(int j=0; j<canny.cols; j++)
        {
          Vec3b color = canny.at<Vec3b>(Point(j,i));
          if(color.val[0] >= 25 && color.val[1] >= 25 && color.val[2] >= 25) {
            Image_Array[j][i] = 1;
          }
          else {
            Image_Array[j][i] = 0;
          }
        }
    }
  //start of obstacle avoidance section
  Analyze(Image_Array, Width, Height);
}
```

As for an explanation, the VideoCapture constructor opens the video capture device and records a single frame from the camera. This data gets stored in a Mat (i.e. matrix) object each time this function is called. The cvtColor() function converts the frame to its equivalent in grayscale, and this gets stored in a separate Mat object. The blur() function does exactly what it sounds like; it blurs the frame using a specified parameter which has been determined through testing. Next, the now blurred, grayscale frame is put through a morphological transformation. The set of nested loops is used to store the results from the Canny Edge detection algorithm into a double array containing 1s and 0s depending on if the individual pixel represents an edge of an object in the frame or not.

### *B.* ***Traffic Cone Detection***

The algorithm implemented for detecting the traffic cone at the base of the pole is a complex system which utilizes machine learning, as well as, some image processing filters. For the machine learning, our team has trained a Haar Classifier which can be used to detect a traffic cone based on its features and geometry. Though effective at only detecting a traffic cone the majority of the time, machine learning classifiers are prone to returning false positives. The way we have avoided this is by verifying the results by confirming that the color of the result is orange.

The way the color is checked is that several points from the potential result, i.e. pixels, are converted to their HSV (Hue, Saturation & Value) equivalent, and are then compared with values which would be typically observed on an orange traffic cone. If the HSV values cannot be confirmed, then the image is rejected. This results in a higher rate of false negatives, but it also ensures that the results are accurate. Below is the algorithm which has been developed.

```
1  /** @function detectAndDisplay */
2  int Finish::detectAndDisplay(Mat frame) {
3    std::vector<Rect> cones;
4    Mat gray;
5    int quintant = 0;
6
7    cvtColor(frame, gray, COLOR_BGR2GRAY);
8    equalizeHist(gray, gray);
9
10   //-- Detect cones
11   cone_cascade.detectMultiScale(gray, cones, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));
12
13   if (cones.size() != 1)
14     return 0;
15
16   if (!isOrange(frame, cones[0])) {
17     return 0;
18   }
19
20   rectangle(frame, Point(cones[0].x, cones[0].y), Point(cones[0].x + cones[0].width, cones[0].y + cones[0].height), Scalar(255, 255, 255), 2, 8);
21   Point target(cones[0].x + cones[0].width / 2, cones[0].y + cones[0].height * 0.8);
22   circle(frame, target, 3, Scalar(101, 255, 0), -1, 8);
23
24   imshow("FRAME", frame);
25
26   return selectQuintant(frame.size().width, target.x);
27 }
```

As for an explanation, the above function takes a video frame from the camera sensor and converts it to grayscale. The grayscale frame has its histogram equalized which essentially means that the brightness of the image is normalized and the contrast of the image is increased. The image is then scanned using the machine learning algorithm for traffic cones. If traffic cones are found, two catty-corner points are stored as a rectangle object in a vector. Then the object is sent to the isOrange() function to confirm the color is orange. Lines 20-24 in the function are purely used for displaying the result. This is useful for testing and showing the algorithm, but these lines will not be implemented in the rover. The final line of code simply calls a function which determines which of the five segments the center of the cone is located in.

```
/** @function isOrange */
bool Finish::isOrange(Mat original, Rect cone) {
  Mat HSV;
  int xVal = cone.x + cone.width / 2;
  int yVal[3] = {
    cone.y + cone.height*0.8,
    cone.y + cone.height*0.2,
    cone.y + cone.height*0.5
  };

  for (int i = 0; i < sizeof(yVal)/sizeof(*yVal); ++i) {
    Mat RGB = original(Rect(xVal, yVal[i], 1, 1));
    cvtColor(RGB, HSV, CV_BGR2HSV);
    Vec3b hsv = HSV.at<Vec3b>(0, 0);
    int H = hsv.val[0];   //  Hue
    int S = hsv.val[1];   //  Saturation
    int V = hsv.val[2];   //  Value

    if (H < 180 && S > 39 && V > 234)
      return true;
  }
  return false;
}
```

The above algorithm takes three points from the potential traffic cone result in the image. The three points are taken from the horizontal center of the rectangle object, and with y-values from near the top, another near the bottom, and another from the center of the object. The HSV values are taken from each point, and, if any one of the points turns out to be orange, then the function returns true.

## VI. Conclusion

Software development of our project has gone well, but due to the hardware setbacks experienced this term we do not believe we will have a finished project by the engineering EXPO. We plan to continue development after EXPO so we can still deliver a finished product to our client