### igvc vision

*Donnie Quamme, Ander Solorzano, Ruffin White-Magner, Kurtis Zimmerman*

Department of Computer Science

Rose-Hulman Institute of Technology  
{quammebd, solorzaa, whitemrj, zimmerka}@rose-hulman.edu

#### Abstract

**The purpose of this project is to aid the Rose-Hulman Robotics Team in the image detection features that they will use in the Intelligent Ground Vehicle Competition (IGVC). After researching previous solutions from scholarly sources, we decided to implement a full vision recognition approach that begins by distorting the image, then applying some grass filtering techniques followed up by line and obstacle detection extraction techniques. In MATLAB, grass filtering, using a SVM classifier proved to be a great method for filtering out noise. For line detection, a combination of grass filtering, HSV feature extraction, and Hough line transforms were used to extract information about lines. Lastly, using LabVIEW, the obstacle detection of construction barrels via template matching managed to correctly detect barrels in images were only barrels and no noise were present or in images with little background noise. The problems and solutions we encountered in this project only pertains to set number of sample images and results may vary significantly in real-time outdoor applications due to lighting, shading, point of view, amount of noise, and color sensitivity of the features present in the obstacle course.**

1. **Introduction**

**2.1. Problem summary**

IGVC is an international robotics competition where students at an undergraduate and graduate level compete to create an intelligent robot capable of navigating an arduous obstacle course. Teams are expected to design and construct autonomous ground robots capable of navigating from GPS waypoint to GPS waypoint, avoiding obstacle collision, avoiding crossing over lines, and efficiently planning a path to the end of the course.

In efforts to contribute to Rose-Hulman Robotics Team’s entry within the IGVC, our focus for this project was to implement vision based detection algorithms for autonomies robotic navigation; including planar perspective correction, line detection, grass filtering, and obstacle recognition. Using previously published procedures for automotive road navigation, we adapted what process we found for what would be required when traversing a smaller outdoor obstacle course at slower speeds. The importance of this project is that it can lead to potential breakthroughs and future applications in autonomous navigation and it is currently an on-going research for developing automated driving assistance.

Throughout this document we sequentially step through each procedure for initial setup to runtime execution to allowing our competing robot to better sense its environment visually.

**2.1. Previous work**

In researching prior work in this domain, we focused on seeking out processes and algorithms developed by groups working on autonomous robots or mobile units meant to traverse roads or courses similar to the competition setup.

First, we found in the GOLD Report [1] a good approach to identifying road lane lines and arbitrary obstacles by a mobile autonomous vehicle’s image processing system. Using a custom hardware setup tuned to the algorithms developed, the mobile unit was equipped with stereoscopic image acquisition to propel object recognition.

For line detection, the algorithm relied on a process similar to edge-finding, in which areas of increased brightness (which would later return to an area of decreased brightness) were marked as potential lane line pixels. The difference from an edge detection algorithm is that the lane lines must be of a predefined pixel width. Though the work completed as part of the report is applied to dark pavement with bright, contrasting white lines, the algorithm is still applicable to a course comprised of a grassy field and less distinct white lines. The algorithm also applies a geodesic morphological operation to strengthen the quality of the identified lines; this is not as applicable because it relies on knowing the expected shape of the lane lines, which we cannot assume to even be vertical or horizontal, to start.

Secondly, the unit identifies arbitrary objects using the stereoscopic view made available. If we find the difference between the two images, any significant areas will indicate left or right edges of obstacles in the field of view. Matching up those edges can provide a straightforward manner for computing the bounding box of potential obstacles.

In searching for reports along the lines of this project, we found the “Red Raven,” a project developed by a California State University team for the same IGVC competition [2]. While specific algorithms were not gleaned at as much detail as the GOLD Report, the overall process proved a good template for our own development. Passing the image through a series of pipelined filters, the system processed each image by filtering out grass and obstacles before beginning the line detection filter. Even the line detection step was comprised of several filters, many of which could be applied in parallel to maximize efficiency.

Lastly, we also applied some processes similar to that of the “Reagle V,” a robotics project developed by the Embry-Riddle Aeronautics University team [3]. The team utilized a series of filters to detect white lane lines, using brightness again as the primary indicator and Hough line transforms to define the lines of interest. Additionally, though our project focused very little on flag detection, this report showed a good approach for quick and effective recognition of red flags in particular, using a simple RGB and HSV threshold to find the distinctly red areas of interest.

All three reports provided a good baseline for the algorithms and processes developed for our final set of components, which will be combined to produce a successful autonomous navigator.

2. METHODOLOGY

We now present the four major components of the robot’s acquisition and recognition process, presented one-by-one with individual results following each step.

**2.1. Perspective correction**

In order for the line detection algorithm to detect relevant lines with respect to the reference of the ground plane, the acquired image from the robot’s perspective must be corrected for the planar distortion when observing the IGVC field from low elevation. A more suitable representation of lines and its locally geographical figures would be from a bird’s eye (top) view. Logically being within the same dimension as the robot will inevitably express its own occupancy grid, this will allow for easy conversion between raw data and geographical features such as describing avoidable obstacles and potential paths to sub goals.

Shown in Figure 1, after a sample calibration images has been acquired, the tedious process of point-based calibration and is performed by marking and identifying specific points and coordinates within the image that represent real space dimensions. In this instance, we have simply chosen the four ends of the pizza boxes as identifiable uniform objects as switch to serve for the point-based calibration routine. By utilizing a tripod to simulate the camera’s angle and elevation within the robotic superstructure, we measure the distances to serve as the reference coordinates to which the image coordinates will eventually map.

|  |
| --- |
|  |
| **Fig. 1.** Image after calibration has been acquired. Here you can see the pizza boxes used in point-based calibration routine. |

When generating the point-based calibration file, it is important to take detailed and consistent measurements, regardless of the calibration tool; a small deviation in error will result in a large skewing effect, further distorting upper background regions within the image. In Figure 2, the floor layout is sketched out using a modeling tool for easy future depiction or for extracting more futuristic points later if required.

|  |
| --- |
| Cal setup |
| **Fig. 2.** Metric measurements of image using a modeling tool. |

After sequentially specifying each marked point of interest and its respective real space coordinates, the distortion model is calculated. As a real-time operated navigational robotic system requires a high refresh rate and its perceptual environmental awareness, faster sensor acquisition and processing is a direct constraint for system response times. It was important to maintain processing structure that is minimalistic in computational intensity but still relatively accurate. In this case we use the rather simple and less mathematically accurate method of division rather than higher-order term computation such as polynomials or other methods due to its efficiency and relatively short computation time. It is possible to make this assumption by disregarding potential discontinuities and higher-order effects such as camera lens distortion. Such assumptions would be harder to render when considering the use of different hardware such as fisheye lenses when attempting to achieve a wider angle of peripheral perception.

To check the calibration accuracy we can take a look at the error map generated by our distortion model. By observing the error range within the bottom right corner of Figure 3, we can see that the magnitudes are severely minute. This is the direct representation of the distortion model based on the ideal pixel coordinates we specified, but there is still a great deal of human error in recording the calibration space as well as environmental variables such as the actual camera placement and angle as subject to physical mechanical variances. Extreme accuracy is not required within our robot, as the goal is to simply avoid lines rather than geologically surveying their existence. However, great care must be exerted when implementing the camera fixtures upon the robotic platform to ensure that unanticipated camera vibrations or displacement is minimized when traversing rough terrain in order to ensure reasonable and consistent results.

|  |
| --- |
|  |
| **Fig. 3.** Front-panel view (GUI) of the calibration training interface. |

Figure 4 exhibits the first rendition of perspective corrected calibration image. Notice that the image is mirrored horizontally and thus appears upside down. This is due to the respective orientation of the origin correlating with the origin of the image, mainly due the Y axis pointing in the same direction of ascending value of the row index as expected with most standards of representing images as matrices. Also notice the severe skew of the length of the left side versus the right side of the corrected image. This is due to the slight rotation and the subject of calibration and its reference in real space or orientation with that of the camera. It could be possible through trial and error to move the camera in a sweeping arc across the subject of reference to correct for this skew and leave the bottommost edge on a level plane. However, subjects that fall within this distance range are of lesser concern than closer objects that will be correctly represented, as reaffirmed within the line detection section of this report. It is such that the outermost edges may eventually be masked or ignored when converting detected lines spaces to an occupancy grid representation.

This entire process is repeated again in order to generate the second calibration file for the second and remaining stereoscopic camera.

|  |
| --- |
|  |
| **Fig. 4.** First rendition of the distorted image. |

Once the calibration files have been constructed, we can implement the results within our video feedback virtual instrument, or VI. In order to make the perspective distortion useful for both LabVIEW video acquisition and Matlab test images, we overhaul our acquisition process to allow for static image as well as methods for programmatically specifying right and left calibration files as well as image source and destination directories. It is important to note that even when working with relatively small stereo image dimensions of 800x600, that by including perspective correction subroutines, our vision acquisition loop is subjected to and increased cycle time from about 30 to 50 ms per frame to roughly 1.3 seconds. It is also important to know that average CPU usage also increased from around 30 to 40% to 95 or 100%.

|  |
| --- |
|  |
| **Fig. 5.** Undistorted and distorted real-time image video stream for both cameras in the front-panel GUI. |

Figure 6 shows our high-level block diagram, depicting the beginning image acquisition and perspective correction process. By reading the user-defined input parameter, the loop continues to either acquire a live video sample from the USB camera hardware, or read in an image from a specified directory. The image parameters are then displayed on the front panel while the stereoscopic images are forwarded on to perspective processing if enabled, where the results are then displayed and can be recorded. The periodicity of the entire loop is set for 3 seconds to allow for less strenuous CPU usage. That time could be reduced if further hardware improvements are made.

|  |
| --- |
|  |
| **Fig. 6.** Block diagram showing the image acquisition process. |

In Figure 7, observe the finished product of our corrected perspective for one of our previous calibration images. Notice the perfectly rectangular features of the centralized pizza boxes, as would be expected since they were the region of interest for our prior calibration efforts.

|  |
| --- |
| 2013-02-08_01-02-11-.953__L |
| **Fig. 7.** High resolution distorted image showing the centralized pizza boxes. |

**2.2. Grass filtering**

Grass filtering was an important part of this project. If grass can be filtered out well object and line detection becomes much simpler. Thus a functional grass filter is critical to the success of the vision system. Three approaches were used for grass filtering. They are: a naïve pixel classifier, an SVM pixel/group classifier, and a k-means approach.

2.2.1 *Manual filter*

The first attempt at a grass filter was done using hard coded threshold to get a feel for how difficult the problem would be. As the figure below shows, the results were fairly successful. However, certain portions of lines were very difficult to identify, and significant portions of background were frequently included as grass. Overall, the success of the manual filter encouraged progression into more advanced techniques.

|  |  |
| --- | --- |
| F:\Courses\ImageRec\FinalProjectStuff\Code\MatLab\Manual Identifier\img39685.png |  |
| **Fig. 8.** Sample manually grass filtered image. | |

2.2.2 *SVM filter*

The next option explored for a grass filter was an SVM filter. For this many chunks of known grass or non-grass pixels were extracted for a series of sample images. From these chunks features were extracted, including color, color moment, and variance. Using these values an SVM was trained. After several trials it was determined the best combination of accuracy and speed was generated using an SVM with kernel width of 5.

The SVM worked fairly well. It identified the majority of clearly defined lines, and a significant portion of lines that weren’t clearly defined. There was however more noise introduced into the image than the manual filtered provided. This was deemed not problematic because this is a preprocessing step, and future steps are well equipped to deal with small amounts of noise. Figure 9 provides a sample grass filtering using the SVM filter.

|  |  |
| --- | --- |
|  |  |
| **Fig. 9.** SVM grass filtered image. | |

2.2.3 *K-means filtering*

The final grass filtering method we applied was a filter applying the k-means algorithm. Using the known grass and non-grass pixels extracted from several images, we applied a k-means algorithm using a value of k equal to 10. From this, we identified the ten most common colors in grass, and stored these values. Then, to filter an image, we applied k-means to the image where the means were within a specified color distance from the known grass means was determined to be a grass pixel and included in the filter.

The run time for this algorithm was problematic. It took significantly longer than desirable to run the necessary k-means algorithm. This effect was mitigated by significantly resizing the images, although this led to slightly misshapen lines. However, the results were fairly good even with substantial resizing, as shown in the figure below

|  |
| --- |
|  |
|  |
| **Fig. 10.** Good results with k-means filter. |

Results similar to figure 10 occurred majority of the time. However, when the algorithm failed, it failed spectacularly, and in such a mode that the results were unusable. Figure 11 demonstrates this failure.

|  |
| --- |
|  |
|  |
| **Fig. 11.** Failure with the k-means grass filter |

**2.3. Line detection**

An integral portion of the robot’s navigation is identifying and remaining within the bounds constrained by white lines. Based on a lane line detection algorithm developed in the GOLD report (1), we developed a process for detecting white lines on a grassy field.

Whereas in (1), the lines being detected were bright white lane lines on a significantly darker pavement background, the lines in this competition are painted on a grassy background, so the contrast is significantly reduced. Additionally, the material difference between pavement and grass meant the lines are not as uniformly solid in our test images.

One last difference between our scenario and that from the GOLD Report is the type of lines we may encounter. Whereas lane lines on a road run parallel and – within some short distance – are completely vertical, within a small error, the lane lines on the competition field need not be parallel, may arc significantly, or – as in the case of a turn – may run horizontal.

|  |  |
| --- | --- |
| img39685 |  |
| **Fig. 12**. The types of lines to be detected include solid white straight lines, dashed lines, and arcs. The lines are not guaranteed to be uniformly distinct, as shown in these two images. | |

However, the basics of the lane line detection algorithm can be further enhanced to work with an acceptable degree of accuracy. While the original algorithm makes just one pass, scanning horizontally for significant changes in brightness, we add on a vertical pass to account for horizontal lines, arcs, and even diagonal lines.

To begin setting up, we first convert the image in RGB space to HSV space, using the value band to represent brightness. As is shown in Figure 13, while not perfectly consistent, HSV space provided the most consistent brightness indicator, as compared to two other calculations found in (1), distance in RGB space as well as a more refined weighted distance measure.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) RGB Distance | (b) Weighted RGB Distance | (c) HSV Value Band |
| **Fig. 13.** Available measures for perceived brightness in an RGB image | | |

Then, instead of doing two iterative passes over each image, we applied a filter which accounts for the same calculation that the GOLD Report computes, accounting now for our vertical pass as well. In Figure 14, the matrix shown can be expanded with more zeros to account for the expected line width. Applying the filter will amplify all pixels which are candidate line pixels, while grass field pixels will be reduced to lower intensities. The result of this filter is shown in Figure 14(b).

Because of dead grass and patches of dirt, some grass pixels are misclassified as line pixels, so we utilize the power of the grass filter to remove as much of the grass background as possible, as shown in Figure 14(c). Lastly, applying a simple Hough line transform to the remaining binary image, we end up with a binary image marking all remaining strong candidate lines, the final result given in Figure 14(d).

|  |
| --- |
| (a) |
| (b) |
| (c) |
| (d) |
| **Fig. 14.** Starting with the original color image, three primary steps are applied to detect white lines. |

As shown in the resulting image in Figure 15, our algorithm also worked well on arcs, which we did not initially anticipate, so no further refinements had to be made to account for different shapes of lines.

However, due to perspective in each of the images, faded lines and distant lines could not reliably be picked up by the line detection algorithm, like those shown in Figure 15. While not ideal, the approach is acceptable for two reasons: first, distant lines are not an immediate risk to the robot and can likely be picked up by the robot if approached; and second, the perspective mapping mentioned earlier will improve the line detector’s ability to pick up lines which appear smaller because they are further in the distance.

|  |  |
| --- | --- |
|  |  |
|  |  |
| **Fig. 15.** Lines which appear faded or distant cannot reliably be picked up by the line detector due to the brightness threshold and expected line width. | |

**2.4. Obstacle detection**

At the IGVC competition, the Robotics Team will encounter several types of obstacles as the robot autonomously navigates the course. Obstacles include construction barrels (orange and white pattern), solid colored barrels, construction flags (red and blue), sawhorses, construction fences, cones, and any other obstacle that the judges feel like introducing. Due to the symmetric properties of barrels from any angle, we decided to detect construction barrels through our camera using image recognition techniques. Other asymmetric obstacles such as fences and sawhorses will be detected with our on-board LIDAR due to its resolution and distance measurements.

In the past competitions, construction barrels have proven to be a rather tricky obstacle due to their pattern, reflective properties, and white-line features that tend to get mixed with white lines and thus detected as white lines and not barrels. Another obstacle that we had to overcome in this particular case consisted of dealing with barrels in the background and barrels that appear to overlap.

After reading the California State University report [2], the team decided to incorporate some of their ideas as well an exploring different ideas learned in class.

2.4.1 *Barrel detection via color classification*

Using LabVIEW as a programming language, the team explored barrel detection techniques using the Vision Assistant tools. This barrel detection algorithm is done in three major steps.

First, the vision assistant acquires the image from an image acquisition block. In our case this would be the direct output of the perspective correction with the image already distorted. Then, the algorithm will search in the entire picture for regions or matches whose parameters give it a score that is high enough to be detected. Such parameters include setting a color score weight, a minimum contrast, rotation invariance, feature types (i.e. color and/or shape), pixel accuracy, color sensitivity, and search strategy (i.e. conservative vs. aggressive). All these parameters can be tuned in LabVIEW to increase the accuracy of barrel detection but we must keep in mind that some parameters, like the conservative search, will increase computation time which will affect the speed of the robot. The final step of the color classifier algorithm searches everywhere in the image until a region with a high enough score based its parameters is found. Such regions will have bounding box drawn around them.

|  |  |
| --- | --- |
| C:\Users\solorzaa\Desktop\Barrels\barrel9.jpg | C:\Users\solorzaa\Desktop\Barrels\SR Results\figure1.JPG |
| C:\Users\solorzaa\Desktop\Barrels\barrel4.jpg | C:\Users\solorzaa\Desktop\Barrels\SR Results\figure2.JPG |
| **Fig.** **16.** This images show the results of some barrel detection via color classification. The top row shows the ideal case with the barrels being correctly classified due their features, sizes, pattern, and contrast. The bottom row shows a more realistic case where although the barrels were detected, the algorithm also detected some noise. | |

The problem with color classification barrel detection is that barrels that vary too much from the parameters set in LabVIEW cannot be correctly detected. Natural causes such as lighting conditions and background colors can also throw the algorithm off.

In order to improve the results using this type of algorithm, the team suggests a combination of grass and noise filtering to subtract all irrelevant information out of the image. Thus the algorithm would have to search for images where only obstacles and lines are observed.

2.4.2 *Barrel detection via template matching*

Another approach explored to observe changes and improvements in the barrel detector included the use of template matching techniques in LabVIEW. Using the Vision Assistant tool, this algorithm was done in four steps.

The first step involves image acquisition. A 32-bit RGB image, which can be the output image of the perspective correction, will feed into the input terminal of the Vision Assistant block. Then a green color plane extraction will transform the RGB image into a gray image. From here, the user can set some parameters to help in the template matching as well as select regions of interest to search for in the image. Thus the search can be reduced to the bottom half or upper half of the image as needed. Finally based on the template selected and some of the searching parameters optimized, the algorithm will search everywhere in the selected region of interest until matches of approximately the same size are found.

|  |  |  |
| --- | --- | --- |
| C:\Users\solorzaa\Documents\RHIT\RHIT SR\Winter Term\CSSE463\IGVC Vision\MatLab\Barrels\Templates\barrelTemp1.png | C:\Users\solorzaa\Documents\RHIT\RHIT SR\Winter Term\CSSE463\IGVC Vision\MatLab\Barrels\Templates\bigBarrelTemp.png | C:\Users\solorzaa\Documents\RHIT\RHIT SR\Winter Term\CSSE463\IGVC Vision\MatLab\Barrels\Templates\bigBarrelTemp3.png |
| **Fig. 17.** Types of barrel templates used in the template matching algorithm. | | |

Compared to barrel detection via color classification, template matching gave better results and managed to find more barrels. This algorithm managed to detect barrels with wider white strips and in different angles. Nevertheless, barrels that were either too bright or too dark from the template were not correctly detected. Thus multiple templates were required that vary in width and darkness had to be selected. To help the detector, fully sized barrels and strips of barrels were used as templates. However, the templates that consisted of strips of barrels managed to work a lot better and yielded more results than those templates consisting of a full barrel.

|  |  |
| --- | --- |
| C:\Users\solorzaa\Desktop\Barrels\barrel10.jpg |  |
| C:\Users\solorzaa\Desktop\Barrels\barrel9.jpg |  |
| C:\Users\solorzaa\Desktop\Barrels\barrel4.jpg |  |
| **Fig.** **18.** This figure shows some of the outputs and result of the barrel detection via template matching techniques. | |

2.4.3 *Improving barrel detection techniques*

To improve the barrel detection techniques, the team decided to use monochromatic bands for the templates used in template matching. The idea behind this process was to reduce the variations given by natural causes such as lighting and shading. This resulted in matches that were closely packed together and closer to the barrel.

|  |
| --- |
|  |
| **Fig.** **19.** This figure shows improvements in the barrel detection via template matching by using a modified set of monochromatic templates. |

The team also decided to apply a “skyscraper barrel mask” to the image wherever a barrel was detected. The “skyscraper barrel mask” would start from the point that the barrel was detected and then apply several masks (of similar size and shape) from the detected barrel to the top of the image. This would create blind spots past the barrel but since the robot cannot jump over the barrels or fly and since the robot should only be aware of its close immediate environment, the robot doesn’t need to know what’s beyond the first obstacle until after it passes the first obstacle. This would reduce computation time in our algorithm and provide faster reaction time for our robot.

The goal of the team, with respect to barrel detection, would be to implement this algorithm and have it ready for demonstration. Although this newly thought part of the project has not yet been implemented, the figure below shows the idea of what the team is trying to pursue. This idea would be handed down to the Robotics Team where they would focus on implementing this idea and combining it with the grass filtering and perspective correction.

|  |
| --- |
|  |
| **Fig.** **20.** This figure demonstrates the idea of using a “skyscraper barrel mask” that can potentially block out overlapping barrels in the background and random noise. *Though it was not implemented in this project, the idea will be passed down to the Robotics Team and further pursued.* |

2.4.4 *Detecting other obstacles*

Although the main focus of this project consisted of detecting construction barrels due to their reflective and misleading pattern, the team also decided to help out on detecting red flags using HSV color space information extraction.

We set up a straightforward red flag detector based on the hue in HSV space, much as was done in the fruit finder, dilating with a 3x3 square to account for edges and ensure that we overestimate the size of the flag. Although we don’t have any sample images consistent with the type of red flags used in the competition, we were able to run the detector on some slightly similar flag images with promising results. Because of the prominence of the red against the grass, flag detection should be reliable, varying only in situations with small flags or flags which are not fully visible due to orientation, wind, etc.

|  |  |
| --- | --- |
| cid:image001.png@01CE0581.EA523820 | cid:image002.png@01CE0582.49745090 |
| **Fig.** **21.** Sample of red flag detection using HSV color space. | |

3. CONCLUSIONS

Using a four-step pipelined process, the groundwork is in place to allow for accurate and efficient image acquisition and processing for the robot’s image-based vision. Because each component has been developed independent of one another, each filter acts much like a plugin, so components can be added or removed as deemed appropriate for the final competition setup.

The image calibration technique developed for remapping perspective provides a more advantageous view for line detection, and with significant hardware and algorithmic improvements, could help streamline the consistency of lane lines across all images, regardless of orientation or distance in the image’s view. At the same time, because barrel detection relies more heavily on the original perspective of the image, those two components could run in parallel to minimize total execution time for one frame without reducing any accuracy.

The same can be said for grass filtering and line detection. While both need to be applied to the same perspective, because the grass filter can take slightly longer due to its underlying SVM processing, the first two steps – computing brightness and applying the relative edge filter – can be started while the grass filter is still running, again without making any sacrifice to accuracy.

Because all components were tested on a non-ideal set of test images, accuracy is expected to improve dramatically when the components are applied to images taken from the competition field. In particular, the lane lines are expected to be much more uniformly white and bright, and the grass should be more consistent in its coloring and uniformity. The one downfall with this knowledge is that the grass filter may need to be reconfigured to account for the differences in grass pixels. Because the SVM was trained on grass pixels from the non-ideal images, there is the possibility that more ideal grass pixels will not actually be classified correctly. Fortunately, referring back to Figure 14(b), line detection can still be computed with very good accuracy even without the grass filter, so no serious drop will occur in line detection.

4. FUTURE WORK

There are several areas that can be expanded upon and improved to enhance the robot’s functionality and accuracy.

In the near future, the most pressing need is to combine all of the developed features to be used in tandem. The most effective sequence of filters would be the following ordering:

1. Acquire camera image(s).

2. Remap perspective to top view.

3. Detect and remove construction barrels.

4. Detect and remove grass pixels.

5. Detect white lane lines.

All information obtained would then be combined in an occupancy grid to map the robot’s perceived world at that time.

While the results so far acquired for grass filtering are promising, there is much work that could be done in the future. In particular, some form of normalization would probably be the most beneficial route to pursue. If the brightness in the images could be normalized, it seems probable that the spectacular failure shown in Figure 11 could be partially reduced if not completely eliminated.

Additionally, more features could be extracted for the SVM filter to potentially generate better results. The number of features extracted was limited by the immense amount of training time on the number of pixels tested with the number of features used. Work could be done to speed this up and potentially acquire a better SVM filter.

In terms of line detection, there are a couple of short-term enhancements and long-term enhancements that could be made to improve accuracy and computation time. In the short-term, further experimentation with morphological operations to improve the shape of detected lane lines could improve some of the lines which are not picked up as connected components; morphology may also be helpful in removing extra undetected grass pixels prior to detecting lines. Additionally, to deal with issues of reduced brightness of distant pixels, the brightness thresholds could be adaptive based on the position of the pixels within the overall image.

In the longer term, computation from frame-to-frame could be improved by interpolating the position of the lines in the next frame based on the detected lines in the current frame. Especially for a straight line, determining its slope would allow us to interpolate its starting point in the next frame and utilize that information to make line detection more intelligent.

Additional areas of improvement and further investigation would include techniques for reducing computation time and resource requirements. After attempting to re-create similar performance within software, it is easy to recognize the appreciable performance hardware implementations can bring when observing GOLD Report’s profound system-wide refresh rate of over 20 Hz. It would thereby be advantageous to invest time to port our own developed code into something executable on a real-time platform. An FPGA would be ideal for the high-speed and repetitive tasks and mathematical calculations; such could be achievable by utilizing pre-existing platforms like NI’s cRIO. Additional tunable parameters could include reducing the native image resolution when performing image acquisition; reducing the number of dimensions or color spaces when performing distortion calculations, as well as reducing the severity of perceived distortion could help ease the computational overhead of perspective correction process.

5. REFERENCES

[1] M. Bertozzi and A. Broggi, “Gold: A Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection,” IEEE Trans. Image Processing, vol. 7, pp. 62-81, 1998.

[2] Anikstein, Alex. “IGVC Red Raven 2.0.” *California State University,* pp. 9 – 11. June 2012.

[3] Randy Breingan, et. al., “Reagle V.” *Embry-Riddle Aeronautical University*.