VISION CAR

Vinura Paranagama

Contents

The objective of this project was to build a robot that could navigate through a simple and path identify signs indicating stop/go, turn left/right. Additionally, it served to develop my knowledge of C++ and image processing particularly using the OpenCV library.

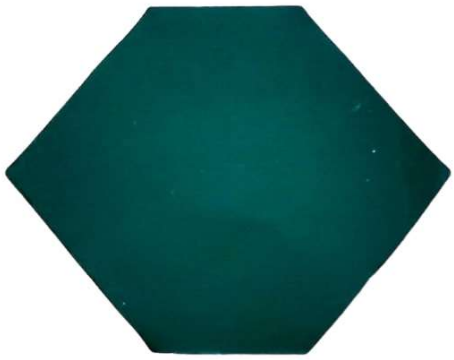
To detect the signs, the program uses images of the same signs that are read in during run time. It uses properties from these templates to asses regions of interest to determine if they are the signs. The vision class was designed with usability in mind, to enable relatively simple implementation of the class for the user. That is why the operations such as navigating lanes, detecting signs are separated into functions. Another reason is the goal of ultimately enabling multiple robots to be run simultaneously on the same program.

The main classes in this program are: Vision, Improc, Lines, Lanes, and Robot. The following pages will describe these functions in more detail.

Vision

The vision class is the most important class and it is what the user will be use to operate the robot. It uses Improc and lanes to process the images, and uses Robot to command the robot. Its main functions include: Road(), stop_go(), and left_right()

void Vision::stop_go()
This function detects the stop/go signs and reacts accordingly to them. The function detects colors by first utilizing the Improc class to obtain the histograms of the stop/go templates. The stop (red) and go (green) signs are shown below.



A histogram is the graphical representation of the intensity distribution of an image [1]. This is done through the void Improc:: get_hist() function. The function first converts the template image to the HSV format, it then seperates only the hue, using the cv::mixChanels() function, which is then used to obtain the histogram using the cv::calcHist(). The histogram is then normalized using cv::normalize() to enhance the finer details of the image. With this histogram, the function obtains a back projection.

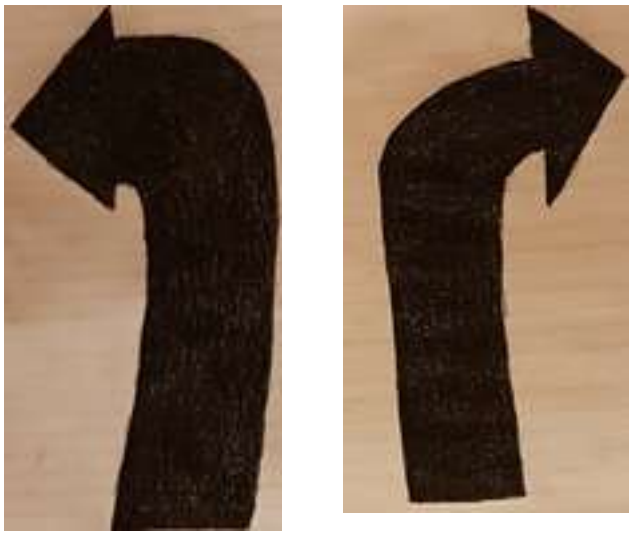
A back projection is a way to identify how well a certain image's pixels fit the distribution of pixels of a given histogram [2]. Essentially, it can be used to identify regions of an image that have the same color as that of the image the provided histogram is derived from. stop_go uses the Vision::get_back_proj() to obtain a back projection using the given histogram and to obtain a binary image, using predetermined values for the ranges, from the back projection. An example of the resulting binary image for the stop (red) histogram is shown below.

This method was utilized after using just the hue, saturation, and value figures proved challenging and very sensitive to changes in the camera angle and exterior lighting. This meant that relying on only the HSV values required those values to be monitored often. However, this method is sensitive to changes in lighting and is by no means perfect.

The function then uses specific geometric properties of the hexagonal shape to identify the signs. This is done in the Vision::shape_det() function. It uses the Improc class to obtain the perimeter, aspect ratio, and the angle the vertices make with respect to the centroid of the templates. Vision::stop_go() uses cv::findContours() to obtain the contours of the binary image that was obtained by the back projection operation. These contours are then passed to Vision::shape_det() which then finds the image moments, contour area, perimeter, and aspect ratio. If the perimeter and aspect ratio are similar to the values of the provided template and the area is within a predetermined range, the funtion proceeds to calculate the angles the vertices make with repect to the centre (image moment). The angles found are inserted into a vector, the Vision::vector_match() takes this vecotr as well as a vector containing the angles with respect to the centre from the template image and compares the two. For each angle, If it is within a specified range it is counted as a match. If there are more than 5 matching angles, Vision::shape_det() indicates to Vision::stop_go() that a stop/go sign has been found. Vision::stop_go() then uses the Robot class to command the robot depending on the found sign. Stop/go as seen by the webcam is shown below.

void Vision::left_right()

This function detects the turn right, left arrows. This is mostly done through the Vision::Match_arrow() function. The program primarily uses Hu moments to identify the arrows. Hu moments are a set of 7 numbers that are calculated using the central moments [3]. An image moment is the weighted average of image pixel intensities. The central moment is similar to the image moment expect that it is not affected by the position of the image. A unique property of these moments is that the 7th moments sign changes for the images reflection. What this means is that for the left and right sign, although the fist six numbers will be similar, the 7th will be of opposite signs. This allows for easier identification. The templates for both arrows are shown below.



Vision::Match_arrow() obtains a binary image, based on pre determined threshold values, from the webcam feed and uses cv::findContours() to find objects. It then determines if the found object is within predetermined values for aspect ratio, and area. If so, the function uses Vision::compare_hu() to asses the Hu moments of the image

found against the moments of a binary image of the templates obtained from Improc. Vision::Match_arrow() then uses a boolean to indicate to Vision::left_right() that a match has been found and the Robot class is then used to direct the robot to turn left or right accordingly.

The image below shows the binary image, here Vision::Right() is running, which only looks for the right arrow.

Turn left/right as seen by the webcam feed.

void Vision::road()

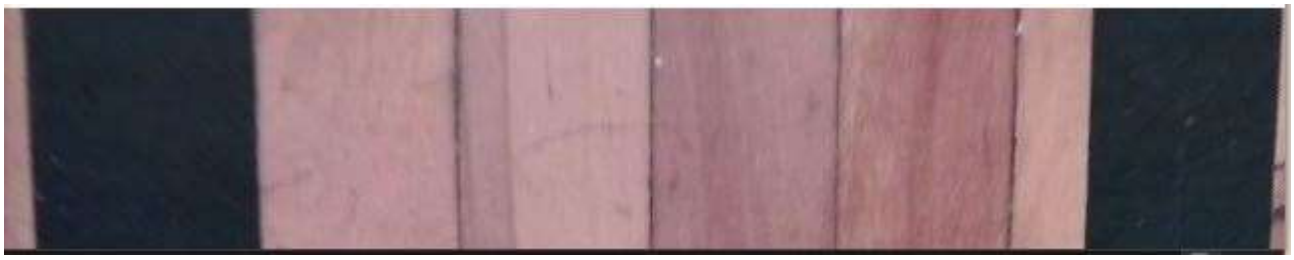
This function relies on the Lane class to identify when the robot deviates from the tracks it then uses the Robot class to steer the robot by spinning on servo and keeping the other stationary for a specific amount of time. The Lane class provides the deviation in the form of an angle. Vision::road() then finds the time need to steer by multiplying the angle by the time it takes for the robot to turn left or right by one degree. This value is determined experimentally.

Lane

This class is meant to detect the tracks. It does so by first cropping the bottom quarter of the webcam feed, then it applies a perspective transform and uses the Canny edge detection algorithm to obtain an 8 bit single channel binary image. Using this image, the probabilistic Hough lines transform function in OpenCV is used to find the lines.

Perspective transform essentially involves changing the viewpoint. A transorfmation matrix is required in order to achieve this and for a 2D image, this matrix is a 3×3 matrix [4]. OpevCV provides the cv::getPerspectiveTransform() function to obtain this matrix. It requires a set of two points, first the input points on the input image and second, a set of points where the first set should be mapped to on the points on the output image. The input points are described by the two white lines shown in the image below. That is where the road is expected to be.

The output points are described as the four corners of the cropped image above, essentially, the points described by the two white lines are mapped to the four corners of the cropped image. The resultant image is obtained by the cv::warpPerspective() function and is as shown below.

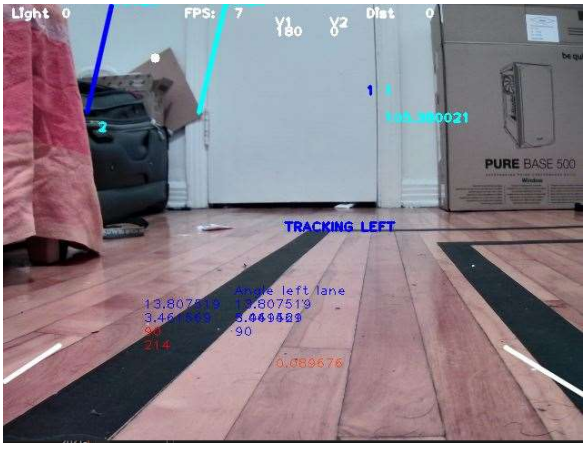


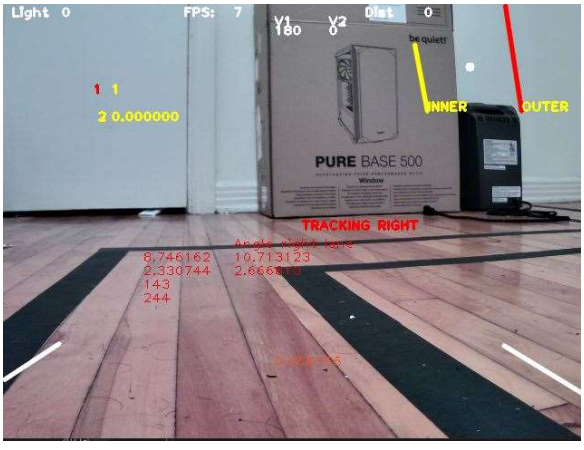
Once this transformation is done, the Canny edge detection algorithm is used to crate a binary image.

The Hough lines transform can then be used to detect the lines. For this program the probabilistic Hough line transform was used, cv::HoughLinesP(). The Lane::find_R_warped() and Lane::find_L_warped() functions detect the lines and work in the same manner. Only lines of a specified gradient are considered. Lane::find_R_warped() uses Lane::min_parrallel_dist() to find the parallel lines of the longest line found and inserts them into a vector and inserts all other lines into a separate vector. If the lines found are not within a specified parallel distance to the longest line, it implies that the found lines are on the longest line or so close that it can be considered to be the same line. The reason for this is identify the inner and outer lines of each lane. The cropped image seen above is split in half vertically and Lane::find_R_warped() finds the lanes on the right, and Lane::find_L_warped() finds those on the left. At this stage, the inner and outer lines still have multiple lines and to address this, Lane::line_aggr() is used to obtain the longest possible line from these lines. The end result is two lines, one each for the inner and outer lines of the lane if they are both visible. If not only the inner line is found.

If two distinct lines are found, The program does not know which line is the inner or outer line. To determine this the Lane::lane_poly_ctr() is used. This function uses the two parallel lines to create a polygon and then returns the centroid of the polygon. Lane::det_in_out() is then used to determine which line is the inner and outer line.

Once the lines have been found, Lane::det_warped_ang() finds the angle made by the line and the Y axis, i.e. the vertical. This is the deviation angle. However, this angle found differs from the actual angle of deviation of the robot. To address this, the corresponding real life angles where found for the deviation angles ranging from 0 to 60 in increments of 5. With these values, a polynomial regression analysis was carried out to determine a relationship. An 8th degree polynomial expression was found with an acceptable error. Lane::det_warped_ang() therefore uses this expression to obtain the corrected angle through the functions Lane::left_ang_actual() and Lane::right_ang_actual(). Lane::det_warped_ang() additionally sends this corrected angle as well as instructions on which direction to turn to the Vision class. If the deviation angle is more than 10 degrees, the robot is deviating. The following images show that.





Improc

This class provides the properties of the templates that Vision uses to identify the signs. It obtains the histograms for the stop/go signs using Improc::get_hist(), the binary image for the arrow signs using Improc::get_mask() and all of the geometric properties of the stop/go signs using Improc::shape_mom(). Improc::shape_mom() works in the same way as Vision::shape_det(). All of the properties Vision use are member variables of Improc and can be used through access functions.

Robot

The robot class sends commands to the Arduino controller using the SerialPort class. It additionally allows for the robot to be controlled manually.

Future work

As stated earlier, the color detection is not perfect. It is very dependent on the lighting conditions and a method to identify individual colors under different conditions is needed. One solution could be to use the light sensor to determine the optimal threshold values for the range of light intensity measured by the light sensor. Additionally, the robot could benefit from being able to navigate a curved path.

Acknowledgements

Credit for the SerialPort class: Zain Ul Mustafa, https://github.com/ZainUlMustafa/Connect-And-Use-Arduino-via-Cpp-Software-Made-In-Any-IDE

References

[1] OpenCV "Histogram Equalization" Accessed 10/08/2021
https://docs.opencv.org/3.4.15/d4/d1b/tutorial_histogram_equalization.html

[2] OpenCV "Back Projection" Accessed 11/08/2021
https://docs.opencv.org/3.4.15/da/d7f/tutorial_back_projection.html

[3] S. Mallick, K. Bapat "Shape matching using Hu Moments (C++/Python)" Accessed 13/09/2021
https://learnopencv.com/shape-matching-using-hu-moments-c-python/

[4] TheAiLeaner "Perspective Transformation" Accessed 15/09/2021
https://theailearner.com/tag/cv2-getperspectivetransform/