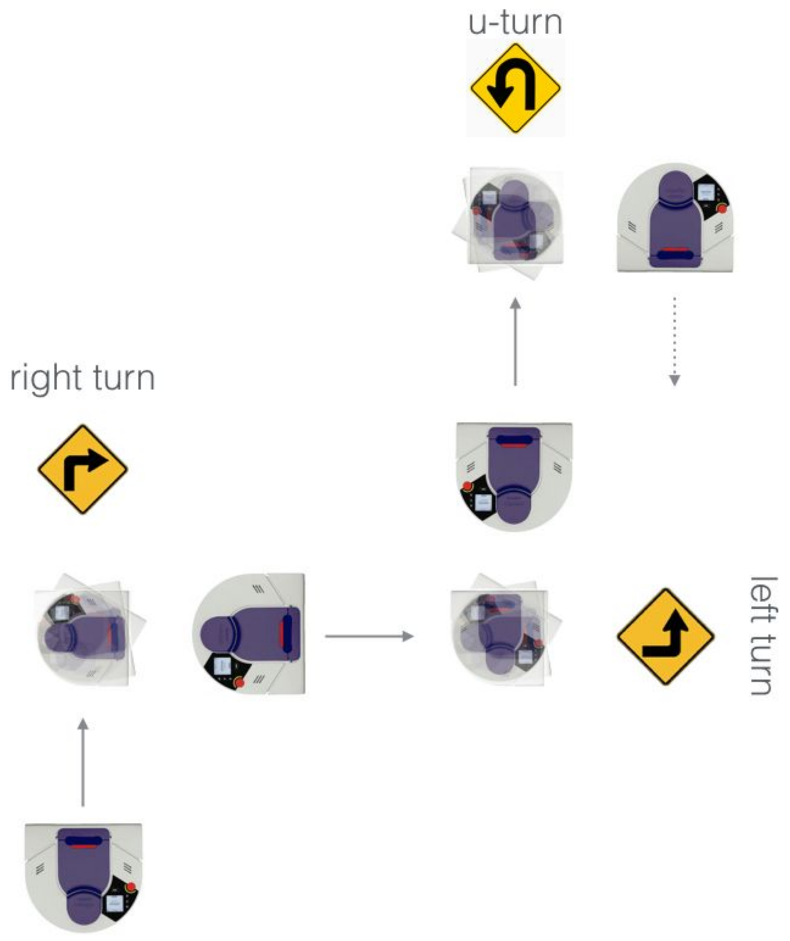
Sign Detector

ROS Package which enables a Neato to see and obey street signs.



Goal

The goal of the project was to use the Neato and have it behave like an autonomous car that follows road signs. As it can be seen in the image that follows, the robot is able to identify road signs, classify them as valid and move accordingly depending on the action that they dictate.

System

Our software system consisted of three main components. Tracker, SignDetector, and SignMatcher.

1. Tracker
    a. Class used to encompass the ROS tracker node. Includes methods responsible for processing an image, detecting a valid sign and making a move according to the sign detected.
2. SignMatcher
    a. Class that applies the basics of interest point detection in openCV. We implemented certain methods in that class that dealt with comparing signs and computing the relevant matches.
3. SignDetector

a. Class performing color tracking to locate the sign and cropped the image to include just the sign

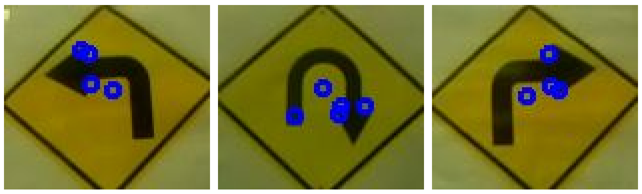
First, in order to locate the sign, we used a hsv filter to find the contours of the color “yellow”, since all of our signs were yellow. The neato looked for the biggest contour of “yellow” and determined the max contour area to represent the sign. This was all done in the SignDetector, which returns a cropped image.

The SignMatcher takes in the image that was cropped from the SignDetector and runs the matching algorithm to determine which sign is presented.

The Tracker first goes forward until the area of the sign is big enough - specified given threshold in code. It stops the neato and uses SignDetector to find the sign. To check that the biggest contour was the sign, we validated the “sign contour” by checking its height and width ratio. If the Neato correctly found the sign, then the height and width should be close to each other. Then it uses SignMatcher that determined which sign it is looking at from the cropped sign image. Then it performs the action depending on the sign and goes forward until it sees another sign.

Algorithms

After founding the sign using the Neato, we decided to use SIFT’s keypoint creator to identify the signs. When we determined that there was a street sign and correctly cropped the picture of the sign, we compared the image with the template for each sign in the image folder. After identifying the key points on both of the images, we ran linear regression test on both x1 and x2 points and y1 and y2 points to see the correlation. If r-squared values were close to the value of one, then the key points produced by both images were close to each other, which meant that two images are most likely similar images. Sometimes the matcher is not very accurate and will match a left turn sign with the right turn sign. In order to eliminate this error we computed the running sum of the matches.



Design Decision

Some of the design decisions we made were that the signs were always going to be directly facing the robot. We did not account for any rotation or tilt to use the key point matching algorithm. If the sign was rotated or tilted then our algorithm wouldn’t have worked as well. We also assumed that the yellow color in the sign would be the biggest contour of yellow that the robot would ever see. We only had the robot match between three different signs. If we had more signs, the keypoint matching algorithm might have more trouble identifying the signs. Moreover, we dealt with signs of the same color (yellow). If we had included signs of different color, then we would have had to filter for different colors, as well.

Challenges

The first challenge we faced was that of understanding how to track the road sign in space. We had to find the sign in that space and zoom into it to process whether it was a left, right, or u-turn sign. We did that by identifying the biggest contour of a given color in that space scanned. Moreover, we identified the leftmost, rightmost, uppermost and bottommost points of the contour and used them to crop a rectangle of the scanned space to be our output image to process. To verify that this was actually a sign, we wrote a validation method that took into consideration the width and height of that cropped rectangle and allowed for a ratio threshold value of 0.9. That helped us eliminate potential noise in the system that could have perhaps had bigger contour areas but did not fulfill the second requirement. It was also challenging actually validating using SIFT that the image we were examining was matching one of the stock images. SIFT would return “positive” results for opposite images. Using SIFT, the algorithm returned linear regression “r_values” for the x and y coordinates of the images that indicated the similarity of those points. r_values above a certain threshold implied that the image viewed, matched the stock image it was compared to. However, once in awhile, such high values appeared also for images that were not similar to each other. To tackle that, we implement a technique where we would trace a running sum for three cases; left, right and u-turn. We incremented that running sum after each point comparison and tracked whichever target threshold sum was reached first. The first to be returned, was the type the designated targeted image would be classified as.

Improvements

Something we did not look into was recognition of images/signs that were tilted. Implementing a homography matrix on the targeted image in order to account for potential rotations or tilted phases of the same image, is something that we could have implement to improve our accuracy. Moreover, we could have implemented proportional control on the actual movement of the robot in space to provide the system with a smoother motion. Instead of simply assigning it a given velocity, we could have used the size of the maximum contour tracked in space and relate it to the velocity of the Neato. Finally, considering different ways of performing a “turn” could have us working with a camera or the ladar sensor to better understand the position of our robot in space.

Lessons

Understanding how OpenCV and contour tracking is always subject to the environment the system is exposed to, was a very important lesson we got out of this project. Eliminating noise and potential disturbances was something that we encountered continuously and tried to eliminate by restricting our system and defining specific design decisions.