Final project: Picar Lane-Following

By: Nolen Kelley

Introduction:

The goal of this project was to implement the core concepts and skills from CSC 485. We were supposed to take the foundational robotics and computer vision principles we learned and implement them to make a fully functioning RC robotic car capable of navigating a marked track and responding to visual cues in real time. To do this, we used a Raspberry Pi 4B, a Raspberry Pi AI camera, and the SunFounder PiCar-V kit. The Raspberry Pi served as the central computing unit for processing our visual data, given by our camera input, while also running our computer vision algorithms and giving the motor and steering commands. The AI camera provided us with continuous video frames that were then analyzed to detect lane boundaries and specific visual cues on the track. The PiCar-V kit served us with the necessary components to assemble the vehicle, then steer and control it.

For this project, our robotic car was required to perform at least two tasks: (1) accurately detect and follow the line/lanes, and (2) recognize the stop indicator, in this case, it was a horizontal strip of red tape on the track. The Line detection was implemented through a color-based lane detection and centroid tracking, while the stop indicator recognition was achieved through identifying the visual cue on the track. Using HSV color segmentation and contour analysis, our system reliably detected this cue and initiated a controlled stop for three seconds when detected. Combined, these features allowed the car to interpret and appropriately respond to its environmental signals, demonstrating real-time decision-making utilizing the principles we learned in class.

Algorithm Design

It wouldn't be an understatement to say that the algorithm for this project changed about fifty times, due to hardware and other constraints. Overall though, the finished design integrates lane detection, centroid-based steering correction, and visual stop-cue recognition. Each component corresponds directly to a dedicated function within the code. I tried to make the code focus on interpretability.

1. **Video Acquisition and Preprocessing**

The Raspberry Pi AI camera was configured to capture the video frames at a 640 by 480 resolution. But the camera itself brought with it a whole slew of issues during development, with the camera's orientation and color formatting messed up on top of the Pi not registering it. We spent a lot of time just troubleshooting these and many other issues.

2. Lane Detection Using HSV Color Segmentation

The Lane following in this instance began with isolating the yellow lane tape through color thresholding in HSV. HSV was the best choice in this case because HSV is less affected by brightness changes comparatively.

Our predefined yellow range is as follows:

Lower = [20, 120, 120] Upper = [40, 255, 255]

This was applied to produce a binary mask highlighting only the yellow pixels of the image.

The color range was changed time and time again to optimize it, with these values ending up being the best. To reduce the noise and focus the analysis strictly on the region in front of it where the lane lines appear, an ROI (region of interest) mask was applied as well. The ROI retained only the bottom 65 percent of the image by making a four-point polygon, starting 35% down from the top to the bottom of the image, so the car can't be influenced by yellows that aren't from the track.

3. Centroid-based Lane Offset Calculation

Once the ROI was computed and the binary mask was made, the system then calculated its lateral position relative to the lane by using image moments. Image moments are a weighted average of pixel intensities; with this, we can find specific properties of the image in our case, as the centroid of the lane in the image.

Code to find centroid:

Centroid = int(moments[“m10”] / moment [“m00”])

M10 = sum of all x-position white pixels, M00 = total number of white pixels

Y-value not needed since were only want our car to be horizontally aligned with the x-position of the line.

The lane offset was then calculated by subtracting the image's horizontal midpoint. The following is the math used for finding it.

Offset = centroid - (frame width / 2)

This math function tells us how far off the centroid is from the center of the camera image

4. **Proportional Steering Control with Memory Tracking**

Steering adjustments were computed by using the lane offset to serve as the input for them. The Steering commands were then generated by applying a proportional gain to the offset. The PiCar was precalibrated, where the angle 90 corresponds to a straight angle.

Angle = 90° + (offset x 0.30)

The gain was tested and optimized, with 0.30 being the best value found to respond best to turns. Multiplying the offset by 0.30 converts the lane's pixel divergence into a steering direction optimal for correction that'll keep the robot centered without over/under steering.

5. **Stop Indicator Detection Using Red Masking and Contour Analysis**

To satisfy the stop detection requirement of the car, the robot had to identify a horizontal red strip of tape placed across the track. Because the red line appears at both ends of the HSV spectrum we had to combine two different hue ranges. Both of these rnages were then combined and merged into one single binary mask.

Then an ROI was applied, this roi covered the bottom 60% of the image to prevent premature stoppage of the car. The algorithm then obtained the contours from this mask and evaluated each contour based on various factors, one of them being area, we declared the area to be over 500 pixels to eliminate some of the noise in our image, another factor would be the aspect ratio, the width needed to be above 35% of the frame width, we did this to further prevent early stoppage on small red object. If the contour satisfied these factors, the system would recognize it as a stop indicator.

One issue we had in testing was the car stopping and then not moving again as it would keep detecting the same red line, to fix this we added a “cooldown” feature to our algorithm. We made it so that the car would only stop if it's been over three seconds since the last time it detected a red line as to give it enough time to cross the red line without making it stop.

6. **Control loop**

All of our functions were integrated into a continuous real-time loop that went as follows:

1. Capture and pre-process the video frame
2. Detect the red stopping indicator (with the cooldown)
3. If the stop sign was detected then stop for three seconds
4. Otherwis compute the yellow mask
5. Apply the ROI and calculate the lane offset
6. Execute the steering and motor control commands
7. Repeat

Results

This project successfully implemented color-based vision, contour analysis, and steering control to obtain a fully reliable lane following and stop-detection system in real time. The yellow lane markings were consistently detected using HSV color threshholding. ROI was used as well to limit the processing of irrelevant parts of the image while shifting focus to the lower region of the frame, thus reducing noise and making it easier on the Pi. Moments were used to compute the centroid of the detected lane, allowing the system to calculate the offset between the lane position and the camera's center.

We used this offset to translate into steering commands through a simple system where the angle of the wheel changes based on how far off the car is from the line, this allowed for smooth lane tracking. When the lane is lost however, the system kept the previous steering angle, to prevent instabilty as we had an issue where the camera would lose the lane in its frame during turns so this allows the lane to keep turning until it eventually finds the lane again.

Outside of just lane following, the car accurately detected red horizontal stop tape using HSV ranges. Contour area and aspect ratio filtering also help with the consistency of the stop line detection by eliminating some of the noise from irrelevant objects off of the track. Upon detection the vehicle would then stop for a fixed duration of three seconds and would then continue. A cooldown system is in place as well to prevent repeated stops from a single detection. Overall, the system in place demonstrates consistent lane racking and reliable stop detection under controlled conditions.

Conclusion

This project demonstrates computer vision techniques can be effectively applied to autonomous navigation on embedded hardware such as the Raspberry Pi. By using color masking, contour detection, and centroid based tracking, our vehicle was able to follow the given lane markings and respond appropriately to stop cues without the need for computationally expensive models. Using steering control that was proportional allowed us to smoothly adjust our car, while retaining simplicity.

The use of contour filtering, stop cooldowns, and other features that were added while troubleshooting improved our cars performance and applicability. These results just go to show that you dont need a bunch of fancy pretrained models to achieve a reliable result, adapting to constraints and purposefully designing optimal solutions, you can build systems that are both efficient and effective.

Resources

On top of the material we learned in class, there were several other things I had to learn in order to complete this project; they are linked below.

https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

https://stackoverflow.com/questions/70734625/good-technique-for-color-masks-with-opencv

https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html

Issues, fixes, and recommendations for future attempts

One of the reasons for the difficulty of this project was the slew of issues that the hardware brought about, in this section I will discuss some of the issues that came about and fixes for them for any future attempts people make at this project.

I would like to preface this by saying that before you make any attempts at this project, familiarize yourself with the hardware and what each part is/what it does, this is vital because it

makes diagnosing certain issues way easier in the grand scheme of things. The best way to do this is to disassemble the car and rebuild it if you're working with a model that was previously used. When doing this project make sure you try and follow the sunfounder manual online, it makes the process way easier.

Hardware issues:

One of the main things this project taught us was the limitations of hardware, my original plan for my code was way different than the end result. The original algorithm I made included mostly things that we learned in class, for example, originally canny edge and hough transformation was used for line detection. In the end though this turned out to be very computationally expensive thus causing the Pi to crash within 30 seconds of running the code. to put it into perspective just to be able to see the live camera and mask video feed you need to limit the FPS severely so as to not crash the pi, and it still crashes from time to time doing this, be aware that brownouts are also very common.

One thing to note, if your car is up and running with your code but randomly wont move despite previously moving, take notice of the two LED lights on the circuit attached directly to your Pi (robot HATS), they should be labeled D2 and D3. If one of them is flashing or even turned off your batteries are dying or don't have enough voltage to make the car work. Firstly I recommend restarting the Pi because there's also a chance that the Pi has too much power and shorted. If this doesnt work and it's still not on, swap out your batteries/ recharge them.

If youre not using a brand new picar then keep in mind that the battery life should be about an hour for a fully charged one, in the case that your batterys die within 10 minutes or one run of your code, change the PiKit that youre using.

If you suspect that your motors for your back wheels aren’t working, take the two pin jumper wire (black tipped wires) and gently tap or place them on the red metal rod instead of the yellow one, your wheels should start spinning very fast since when doing this youre feeding them straight electricty, be careful as doing this for too long can drain your battery/short your pi.

Finally, note that if youre using an old PiCar and not a new one there's a good chance that some of your parts are broken, be prepared to disassemble and assemble with parts from other car kits.

A final suggestion: if you end up doing object detection for your project use the AI camera as it already has models on it that process on the camera, make sure you use these because they will put less strain on the Pi itself.

Software issues:

If you're using the AI camera first make sure you check that your Pi is updated to the latest or at least a compatible OS, bookworm should be the oldest OS compatible with it. Next when you are installing the servo for your car, please note that your car will not move with your code unless you install Django the installation makes it look optional, it is not. you specifically want Django version 2.2 as other versions are not compatible with the Pi. On top of this you need to use the preset calibrations that the Sunfounder library has, this is in the folder Sunfounder_V from here go into the remote control folder (this is where your file should be for your code), from here there should be a file with the calibration. make sure that in your program for the car you use the config, Here is how we called ours and made it work.

```
DbFile = "/home/pi/SunFounder_PiCar-V/remote_control/remote_control/driver/config"

fw = front_wheels.Front_Wheels(debug=False, db=DbFile)
bw = back_wheels.Back_Wheels(debug=False, db=DbFile)
```

When using PiConnect make sure your Pi and computer are on the same wifi network (school wifi won't work use your hotspot if in school), as it won't work if you dont. if youre using mac sometimes the screenshare feature won't work unless you connect your Pi to a monitor so in most cases it's easier to connect your Pi to a monitor and work from there. For windows if youre experiencing crashes inconsistently (within 1-10 minutes of use), like just navigating around or writing code, it would be in your best interest to change the Pi's SD card. In our opinion the most likely reason for this is probably your Pi being corrupted.