Self-driving RC Car

**Overview**

The object-following RC car is capable of autonomously following an object of known color and size using computer vision. The object parameters are specified in the code and can easily be changed. Using these parameters, the car is able to find the specified object within the image frame of a camera and determine its distance from the object. The car uses a Raspberry Pi for image processing and high-level control and an Arduino for controlling the servo and motor in real-time. The Arduino also samples input from a receiver to determine whether a manual takeover has occurred. Manual takeovers allow a user to take control of the car with a transmitter in case of emergency or to let people have fun driving it.

**Basic Use**

The car is turned on by flipping the switch on the chassis (between the receiver and the motor). The program starts as soon as the Raspberry Pi finishes booting (takes about one minute). When you turn the car on, you will hear the ESC give connection confirmation three times. The third confirmation indicates that the program has started. The car is currently set up to follow an orange cone (though this can easily be changed). When the object is held in front of the car, the wheels will point directly toward it and, if the object is more than 24 inches from the camera, the throttle will go at 12% power (as slow as the car is able to go).

To perform a manual takeover, turn on the transmitter and turn the steering wheel about 45 degrees clockwise. The Arduino will now follow commands from the transmitter instead of the Raspberry Pi. To give control back to the Raspberry Pi, the car will need to be reset by flipping the switch off and on.

**Finding/Setting New Object Parameters**

To be able to alter the code on the car, you will need to SSH into the Raspberry Pi. To do this, connect an ethernet cable from your Linux machine (or use PuTTy for Windows; I don’t know how to do this with a Mac) to the Raspberry Pi. Turn on the Pi by turning the car on or by connecting it to a wall outlet (wall outlet is preferred) and wait one minute for it to boot. Find the Pi’s IP address with the command “ping raspberrypi.local”. Then SSH into the Pi with “ssh -X pi@<IP\_address>”. Enter the password “raspberry”. Whenever the Pi boots, it starts the driving program automatically. You will need to disable this in order to use the camera. To disable the program, open the file /etc/rc.local (from the home directory, “sudo nano /etc/rc.local”) and comment out the line that starts with “python3” (this line allows the program to run whenever the Pi boots; you will need to comment it back in after you’re done setting new object parameters). Save (ctrl+x), then reboot the Pi (“sudo reboot”). After it reboots, ssh back into it. You’re now ready to find new object parameters.

Pick an object that is small enough to completely fit within the image frame at a reasonable distance but large enough so that it can be seen easily from far away. Any object that is between 6 inches and 3 feet in diameter will work well. The program will work best if the object is relatively round and has a unique color (e.g. orange, pink, neon green, etc.) that isn’t likely to be found in the background of the image frame (to avoid false positives). Measure the radius of the object and set the variable “object\_radius” in the control\_loop function of controller.py to that value (to edit files in the terminal, you can use nano or vim; use the command “nano <file>”; or you can use file sharing to edit files in a text editor on your own machine).

Next, you will need to find high and low HSV color values that contain the color of the object without permitting false positives. To do this, run the script called range\_detector.py by running this command from the self\_drive directory: “python3 range\_detector.py --filter HSV --webcam”. Three windows will appear. One shows the unaltered camera feed, one shows the output of the color filter, and the third is a GUI with track bars that allow you to change the minimum and maximum HSV values for the color filter. Find the values that enable the entirety of the object to pass through the filter while blocking out everything in the background. Widen the threshold values as much as possible without permitting excessive false positives. Write down the threshold values.

At the top of the perception function in the controller.py script, create two tuples corresponding to the min/max HSV values (e.g. orange\_low = (0, 60, 183), orange\_high = (119, 200, 255)). Set the values object\_color\_low and object\_color\_high to the new low and high threshold tuples, respectively. After saving the script, go back into /etc/rc.local and uncomment the line you commented out previously. Reboot and the car will follow the new object.

**Code Overview**

The car has a script running on the Arduino for real-time motor and servo control and two processes running on the Raspberry Pi for image processing and control.

**Improvements to be Made**

The car has several issues that could be improved to make it function much better.

Perception:

Color thresholding is not a very good way to do object detection because lighting varies from place to place and because there is always a chance that the color you’re trying to locate is in the background of the image. Rather than using color, you could use a simple blob detector/SIFT/SURF/Harris corner detector to try to find a known object without relying on color. You could use a BYU sign or something and look for the shape of the sign in the frame (using a canny edge detector, contour finder, and approxPoly). You could have a sign that has a word with large letters and use tesseract to find that word in the image frame. You could use a known sign and template matching to find it (don’t know if this would work very well). Or, if have/want background in machine learning, you could use pytorch or tensorflow to find the object (I highly recommend doing it this way; it would be very easy to train because you only ever have to find one object).

Controls:

The controls side of this project has a lot that could be made better. Right now, the angle of the object from the direction the car is pointing is calculated and the wheels are commanded to point to that value. The problem with this approach is that it can be easy for the car to overcorrect and swerve back and forth. A better way to do this is to use a PID controller to steer towards the ball. Another issue the car has is that the throttle essentially uses a bang-bang controller, where the throttle is turned on (at 12%) if the car is farther than a certain distance from the object and turns off when it reaches the object. This doesn’t work very well for maintaining a set distance, and will be even worse at higher speeds. A better way to do throttle control is to calculate the distance from the object, find how far it is from the desired distance, and feed this error into a PID controller. These two improvements will allow the car to chase objects at high speeds.

**Future Projects**

For starters, the improvements listed above can be made. Once done, you could do things like throw a ball and make the car chase it at high speeds, or do speed runs for autonomously navigating an obstacle course.

Here are some other ideas:

* Set up a mini city with traffic lanes, signs, etc. that the car can autonomously drive through.
* Have people drive a second RC car around and have the car follow it.
* Do this same project with a quadcopter (it might be kind of terrifying, but it would be cool!)
* Integrate a GPS sensor and have the car follow GPS waypoints while avoiding obstacles.