Line Following M-Bot

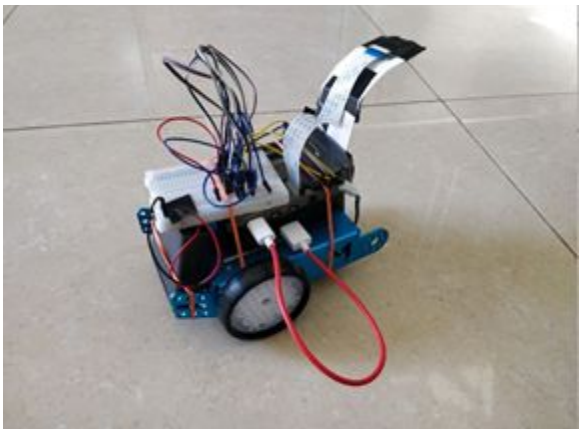
Clyde Bujari | Hannah Clarke

INTRODUCTION

Our system is a robot that uses CV2 image analysis to follow a line. This was inspired by demos shown to us in class, especially the demo by Einsteinium Studios[1]. A system like this represents how a robot can use image processing to detect the desired path to follow and detect when to turn to stay on the desired path as it moves forward.

METHOD

Our goal with the mechanical build of our project was to use parts we already had or could borrow from friends, to minimize cost as well as uncertain shipping times during the COVID-19 pandemic. Using existing parts/borrowing parts was only successful due to the wide variety of leftover robot pieces from previous projects, which include the robot chassis, wheels, and motors. We did have to purchase a motor controller chip that was compatible with the motors that we already had, and a camera. In addition, we also used two battery packs to power the raspberry pi and the camera.



Our hardware design started off with an M-Bot[2] kit lent to us from a friend. This kit does come with a light sensor which can be used for line-following projects, but we wanted to use image processing for a more scalable example of modern robotic systems. Additionally, the M-Bot is designed for beginners and its processor would likely not be compatible with many libraries we needed, so we just used its chassis and motors.

For the brain of our robot, we chose to use the Raspberry Pi 4. While we both had several Pi systems, the Pi 4's additional RAM and processing power provide more than enough headroom to match the demo's BeagleBone system. We initially had some difficulty with libraries from older tutorials not being readily available on the more recent Pi 4, but after much trial and error we were able to get a working CV2 installation ready. The USB webcam we originally planned to use was faulty, so we went with a cheap $8 Pi-compatible camera board that works well for our needs (the cost of webcams was fairly inflated at the time).

[1] https://einsteiniumstudios.com/beaglebone-opencv-line-following-robot.html
[2] https://www.makeblock.com/mbot

Driving motors with a Pi required some additional work but is a well-documented undertaking online[3]. We bought an L293D motor control chip, which can drive the 2-wire motors provided in the M-Bot kit off the Raspberry Pi's GPIO pins. Powering the motors on their own circuit, separately from the Pi, allows us to isolate it from our potentially noisy motors and change out batteries less often.

Our software approach uses CV2 to analyze each of the camera's frames. After a frame is read, we crop it to a much more manageable window and convert it to grayscale. Then, after applying a gaussian blur, we perform color thresholding on the image; any pixels above a certain value are made white, and any below are made black.

From here, we run the image through the findContours function. Analyzing the output of this function gives us an approximate horizontal position for the center of the contours, which we use to determine the robot's next action; move forward if the contours are fairly centered in the image, and turn right or left if they are right or left of the center.

While moving slower may have made this task easier for the Pi, we chose to drive the motors at full power. PWM control of the motors is possible in software, but we decided the benefits were not worth the time it would take to implement this without slowing down the primary frame-capturing loop and delaying the robot's reaction to changes in the contours' center.

For easy debugging of our code, we draw the contours and their approximate center and show the frame, before repeating the process until an escape character ('q') is entered into the frame window or an exception is caught, so that we can safely stop all motor movement and end program execution.

DISCUSSION

While we showed our system's best-case behavior in our demo video, getting it to follow the line correctly took us a lot of trial and error. The main issue we came across was finding a good threshold between dark/light areas; too low and the robot may not recognize the line (we used electrical tape, which appears brighter at times near the camera due to glare from light sources), too high and it would begin to follow its own shadow if it were cast in front of it.

After much trial and error with this threshold value (as well as the room's lighting), we finally reached a state which worked. However, from our experience working with this throughout the day it is likely that this would not continue to work as lighting conditions change.

If we had more time to evaluate our system, the code could have been optimized to slow the robot down as it detects a curve, so that it takes corners slower than when driving straight. This would reduce the overswing when changing direction, if the robots speed were to be increased.

CONCLUSION

Our line following M-Bot is a good introduction to how computer vision can be used to determine a path for robots to follow. In the future, more functions that implement computer vision can be added, allowing the robot to change its speed depending on the curvature of the line, or following a specific colored path when more than one path is available. In addition, the robot could be improved to more reliably detect paths even when the lighting has changed, so that it doesn't follow its own shadow or fail to detect a path when the environment's lighting is not optimal.

[3] https://business.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051

REFERENCES

1. Einsteinium Studios BeagleBone System: https://einsteiniumstudios.com/beaglebone-opencv-line-following-robot.html
2. M-Bot Site: https://www.makeblock.com/mbot
3. Tutorial for Raspberry Pi Motor Control: https://business.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051