

SMART RC CAR

-using Raspberry Pi-

Team name : KHar

Leader : 2018102106 문희준(HEEJOON MOON)

Team members : 2018102137 장재윤(JEYUN CHANG)

2018102125 이승현(SEUNGHYUN LEE)

- Index

1. Team
2. Project Purpose
3. Materials & Supplies
4. About Technology
5. Code Examples
6. Expected Results

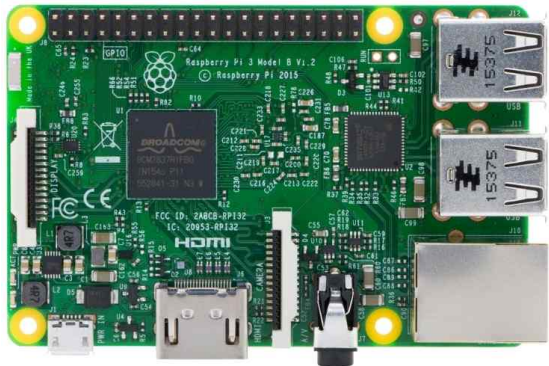
1. Team


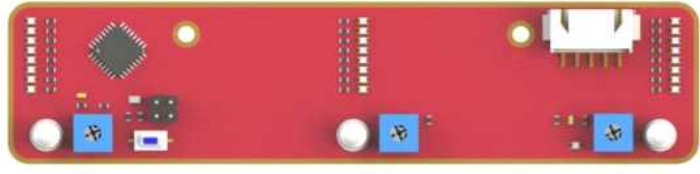


Our team name is KHar, a compound word for KHU(abbreviation of KYUNGHEE UNIV) and car. The team now consists of three team members who are planning to major in future vehicle & robots track.

2. Project Purpose

we're gonna make a raspberry-car that embodies the early technologies of automatic vehicles(line-tracking, light-tracking, avoiding obstacles). Also, using camera module for openCV, our goal is to implement Hough Transform, recognizing both lanes and running along the center of the lanes.

3. Materials & supplies

Material & supply photo	Name & usage
	Raspberry Pi Raspberry Pi 3 B + is one of the most commercially available single board computer. In the project, we want to build a main body by connecting several kinds of drivers to Raspberry Pi and installing each

	module in this driver.
	Line Tracking Module It can use the built-in camera to track the lines attached to the floor.
	Light Tracking Module It is a module that can recognize and track light using sensor.
	Avoiding Obstacle Module It detects the returning ultrasonic wave after launching a constant speed ultrasonic wave and it can grasp the distance to the object.
	120° Wide-angle USB Camera Normally, it is used to track moving objects, but the project plans to utilize Open CV to recognize and drive lanes.

4. About Technology - Hough transform

Theory

Note: The explanation below belongs to the book Learning OpenCV by Bradski and Kaehler.

Hough Line Transform

- 1.The Hough Line Transform is a transform used to detect straight lines.
- 2.To apply the Transform, first an edge detection pre-processing is desirable.

How does it work?

- 1.As you know, a line in the image space can be expressed with two variables. For example:

a. In the Cartesian coordinate system: Parameters: (m,b).

b. In the Polar coordinate system: Parameters: (r,\theta)

Line variables

For Hough Transforms, we will express lines in the Polar system. Hence, a line equation can be written as:

$$y = \left(-\frac{\cos \theta}{\sin \theta} \right) x + \left(\frac{r}{\sin \theta} \right)$$

Arranging the terms: $r = x \cos \theta + y \sin \theta$

1. In general for each point (x_0, y_0) , we can define the family of lines that goes through that point as:

$$r_{\theta} = x_0 \cos \theta + y_0 \sin \theta$$

Meaning that each pair (r_{θ}, θ) represents each line that passes by (x_0, y_0) .

2. If for a given (x_0, y_0) we plot the family of lines that goes through it, we get a sinusoid. For instance, for $x_0 = 8$ and $y_0 = 6$ we get the following plot (in a plane $\theta - r$):

Polar plot of a the family of lines of a point

We consider only points such that $r > 0$ and $0 < \theta < 2\pi$.

3. We can do the same operation above for all the points in an image. If the curves of two different points intersect in the plane $\theta - r$, that means that both points belong to a same line. For instance, following with the example above and drawing the plot for two more points: $x_1 = 4$, $y_1 = 9$ and $x_2 = 12$, $y_2 = 3$, we get:

Polar plot of the family of lines for three points

The three plots intersect in one single point (0.925, 9.6), these coordinates are the parameters (θ, r) or the line in which (x_0, y_0) , (x_1, y_1) and (x_2, y_2) lay.

4. What does all the stuff above mean? It means that in general, a line can be detected by finding the number of intersections between curves. The more curves intersecting means that the line represented by that intersection have more points. In general, we can define a threshold of the minimum number of intersections needed to detect a line.

5.This is what the Hough Line Transform does. It keeps track of the intersection between curves of every point in the image. If the number of intersections is above some threshold, then it declares it as a line with the parameters (θ , r_{θ}) of the intersection point.

5. Code Examples

(1) Light Following - Open Source(part)

```
def read_digital(self):
    analog_list = self.read_analogs()
    #print "Analog_result = %s, References = %s"%(lt, self._references)
    digital_list = []
    for i in range(0, 3):
        if analog_list[i] >= self._references[i]:
            digital_list.append(0)
        elif analog_list[i] < self._references[i]:
            digital_list.append(1)
    return digital_list

def get_average(self, mount):
    if not isinstance(mount, int):
        raise ValueError("Mount must be interger")
    average = [0, 0, 0]
    lt_list = [[], [], []]
    for times in range(0, mount):
        lt = self.read_analogs()
        for lt_id in range(0, 3):
            lt_list[lt_id].append(lt[lt_id])
    for lt_id in range(0, 3):
        average[lt_id] = int(math.fsum(lt_list[lt_id])/mount)
    return average
```

(2) Line Following - Open Source(part)

```
def read_digital(self):
    lt = self.read_analog()
    digital_list = []
    for i in range(0, 5):
        if lt[i] > self._references[i]:
            digital_list.append(0)
        elif lt[i] < self._references[i]:
            digital_list.append(1)
        else:
            digital_list.append(-1)
    return digital_list
```

```

def found_line_in(self, timeout):
    if isinstance(timeout, int) or isinstance(timeout, float):
        pass
    else:
        raise ValueError("timeout must be interger or float")
    time_start = time.time()
    time_during = 0
    while time_during < timeout:
        lt_status = self.read_digital()
        result = 0
        if 1 in lt_status:
            return lt_status
        time_now = time.time()
        time_during = time_now - time_start
    return False

```

(3) Hough Transform

```

# -*- coding: cp949 -*-
# -*- coding: utf-8 -*-
import cv2 # using opencv
import numpy as np
def grayscale(img): # Converting to black and white image
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

def canny(img, low_threshold, high_threshold): # Canny algorithm
    return cv2.Canny(img, low_threshold, high_threshold)

def gaussian_blur(img, kernel_size): #gaussian filter
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)

def region_of_interest(img, vertices, color3=(255,255,255), color1=255): # ROI setting

    mask = np.zeros_like(img)

    if len(img.shape) > 2:
        color = color3
    else:
        color = color1

    cv2.fillPoly(mask, vertices, color)

```

```

ROI_image = cv2.bitwise_and(img, mask)
return ROI_image

def draw_lines(img, lines, color=[0, 0, 255], thickness=2):
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)

    return line_img

def weighted_img(img, initial_img,  $\alpha=1$ ,  $\beta=1.$ ,  $\lambda=0.$ ):
    return cv2.addWeighted(initial_img,  $\alpha$ , img,  $\beta$ ,  $\lambda$ )

image = cv2.imread('solidWhiteCurve.jpg')
height, width = image.shape[:2]

gray_img = grayscale(image)

blur_img = gaussian_blur(gray_img, 3)

canny_img = canny(blur_img, 70, 210)

```

6. Expectation

- (1) All the sensors you have planned can be connected and driven in one body.
- (2) Light tracking, line tracking and avoiding obstacle modules can be operated normally to implement various kinds of autonomous driving.
- (3) By studying Open CV technology, we can understand the utility and possibilities of Open CV technology.
- (4) Open CV technology works normally so that it can recognize the lane and run along the center by recognizing the distance between the lane and the lane by oneself.

7. Refferences

Hough transform

-https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html