Inha University in Tashkent

School of

Computer Science Engineering

Smart Car Using Raspberry Pi

Safe and Autonomous Car Development

By

U1510402 Avaz Alimov

U1510429 Darhonbek Mamataliev

U1510395 Nasiba Lutfullaeva

U1510430 Dilbar Bobokulova

Submitted as a Final Design Project Report for the B.Sc. Degree,
School of Computer Science Engineering,
Inha University in Tashkent

May 2019

Table of Contents

Abstract

This project is aimed to make an implementation for the prototype of Smart Car using single - board computer Raspberry Pi, capable of getting over the obstacles as well as observing traffic lights and road signs.

1. Introduction

The motivation behind our project was that an autonomous driving is a field of great intense of the automobile manufacturing. The core goal of such cars is to reduce the number of traffic accidents.

The algorithms of an autonomous car make it ideally follow the rules on road — showing turns, following signs, and so on. However, technology is unable to control the behavior of other drivers. Robots will have to deal with those who exceed speed, drives a red one and so on. Consequently, the main solution to this problem lies in equipping smart cars with sensors that fix their dynamic position relative to surrounding objects.

However, these improvements and modifications require time, money and practical experience. On this occasion, the project, using Smart car prototype and its requirement and mission was presented.

2. Project Properties

Smart car prototype is constructed in order to visually demonstrate the problems that drivers might face with on the road, and how, with the help of proper software routine, developers are able to cope with such problems. This is a great opportunity to facilitate driving process and to ensure safety for both car owners and pedestrians.

The project was divided into several tasks that should be performed one after another and the following mission tasks were determined:

- Autonomous driving with lane detection;
- Obstacle detection and avoidance;
- Traffic Light Recognition;
- Traffic Sign Recognition;
- Smart parking.

3. Project Functional Components

Software and OS dependencies:

- Raspberry pi OS – Raspbian;
- Development tools – g++ compiler;
- Image Processing library – OpenCV;
- GPIO access library – WiringPi.

Hardware dependencies:

- Raspberry pi computer;
- DC motors;
- Ultrasonic Sensor;
- Camera;
- IR sensor;
- IR line tracer sensor.

4. Product Functions

- Autonomous motion;
- Obstacle avoidance;
- Follows traffic signals;
- Follows traffic signs;
- Safe parking.

Detection: The use of sensors to recognize pedestrians and obstacles in front of the car, traffic lights sensing and the line or board recognition.

Tracking: Using the scanner and algorithms, track the position of the obstacles and pedestrians as it moves within the sensors scanning range.

Apply Braking: When a pedestrian crops up in front of the car and a collision is possible, a signal is sent to the actuator to begin deceleration.

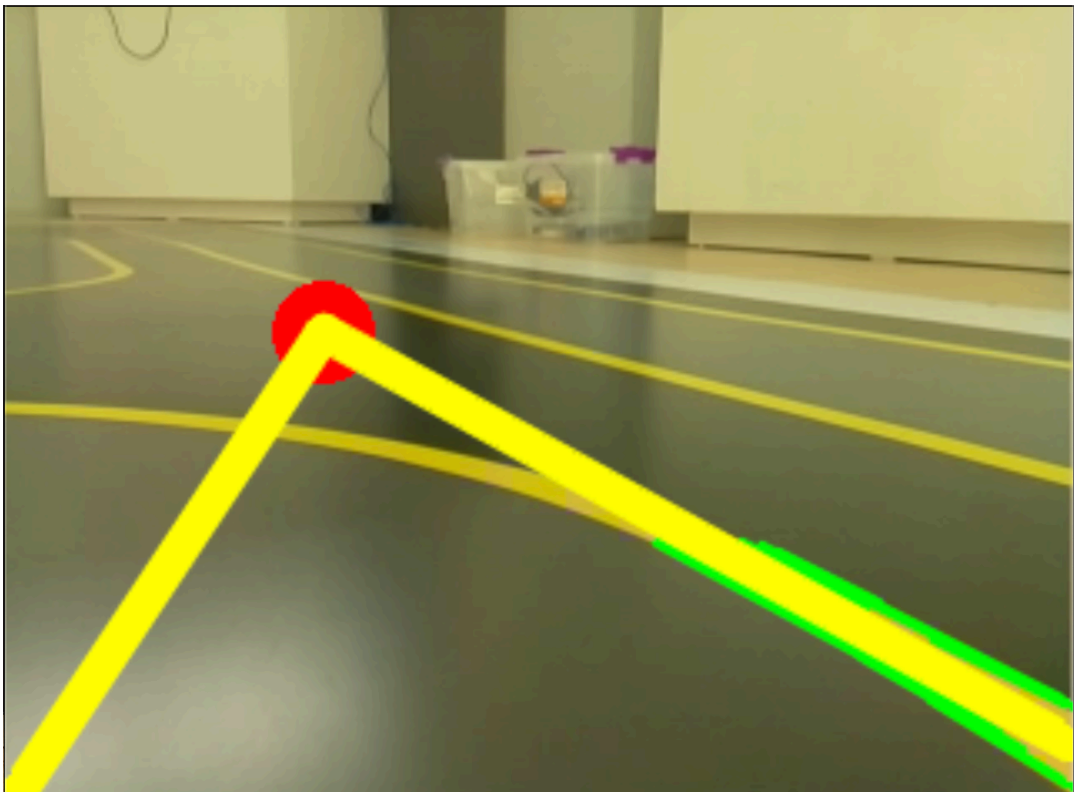
Restore Velocity: When the pedestrian and the obstacle threat is gone and braking has been applied, program car to restore its previous velocity.

Fail Safe: If there is a problem with braking, the fail-safe mode is applied. This increases the time allowed for braking to ensure collisions are avoided.

5. Software Implementation

Lane Detection Algorithm

1. Resize image.
2. Transform image from BGR to GRAY.
3. Apply Gaussian Blur.
4. Find edges using Canny.
5. Find contours from edges.
6. Cut ROI.
7. Find lines using Hough Line Algorithm.
8. Filter lines based on slopes to left and right lines.
9. Find average lanes for left and right.
10. Find the intersection point of left and right lines using algebraic functions.
11. Return direction.

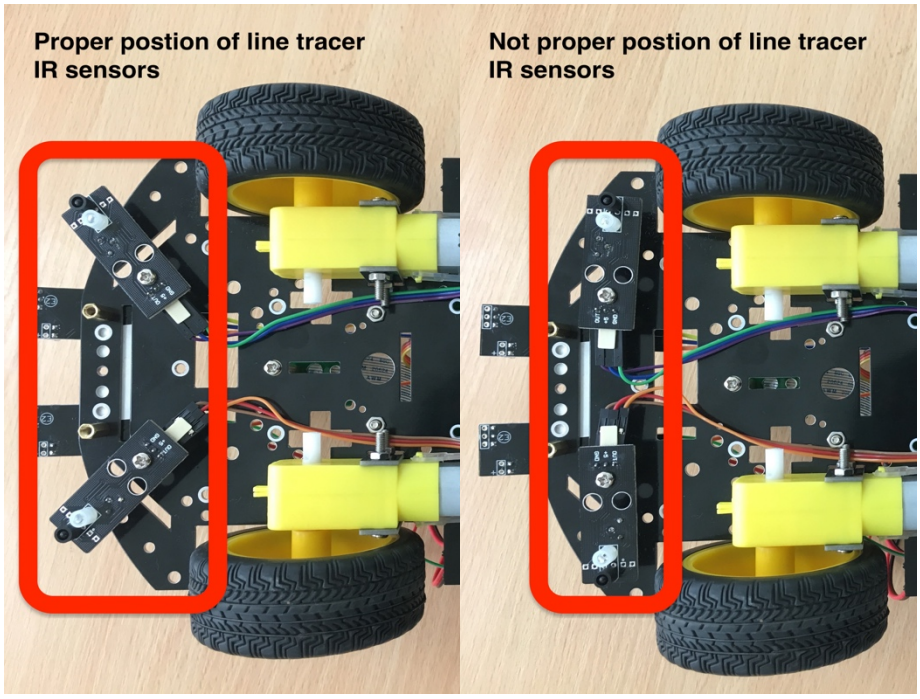


IR Line Tracer Sensors Usage:

Used to assist lane detection and prevent from leaving the road.

1. Read data from right and left pins.
2. Depending on the received signals return direction.

We tried several positions of IR sensors and found best design (position):



Traffic Light Algorithm

We implemented several algorithms for the traffic light detection. We continually improved it step by step by finding it's weak points.

Algorithm #1

1. Find red color using inRange function.
- Lower red value - HSV from (0, 70, 50) to (10, 255, 255)
- Upper red value - HSV from (170, 70, 50) to (180, 255, 255)

2. Find Hough circles inside the masked image.

Algorithm #2 (final)

1. Find red color using inRange function.
- Lower red value - HSV from (0, 70, 50) to (10, 255, 255)
- Upper red value - HSV from (170, 70, 50) to (180, 255, 255)

2. Apply cascade classifier to recognize the traffic light.

We decided to train cascade image classifier on red masked images to identify the traffic light.

Below is the table of comparison of both algorithms:

	Weakness	Strength
Algorithm #1	• False positive results (any red color recognized as a traffic light). • Highly dependent on light conditions. Requires continuous modification of color range values.	• Easier implementation. • Does not require large data set to implement.
Algorithm #2	• Complex implementation. • Requires well trained image classifier. Needs lots of data during different light conditions (dark, light).	• Independent of light conditions, given the classifier trained on different light conditions. • No false positive results.

Traffic Sign Algorithm



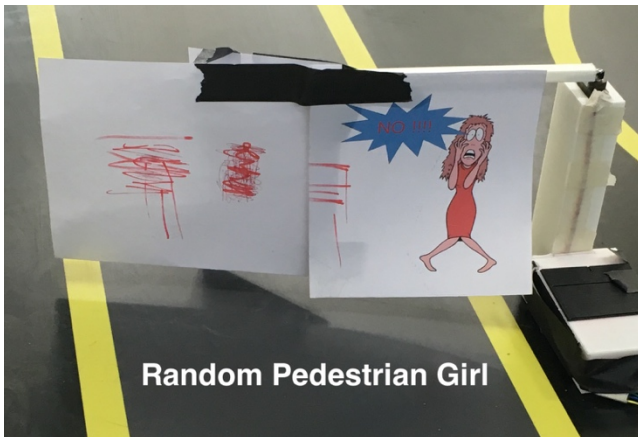
We recognized traffic signs using cascade classifier.
Cascade training details:

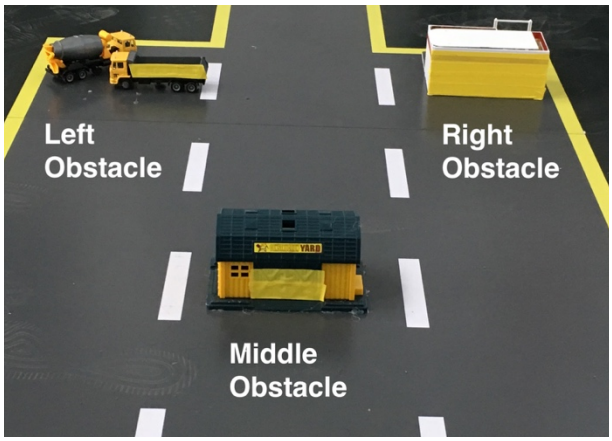
- LBP algorithm (since it is training speed if faster than HAAR as well as it provides less false positive values)
- 8,000 negative images (images of road, other signs), 1,000 positive images (images of dedicated sign)
- Signs trained: pedestrian, stop, left, right, parking.

Algorithm:

1. Get image from raspberry camera.
2. Use detectMultiscale function to recognize the signs
3. If sign detected, perform it's action:
    - Pedestrian: slow down, until the zebra.
    - Stop: stop for 5 seconds.
    - Left: turn left.
    - Right: turn right.
    - Parking: park a car and stop.
4. If sign is not detected, grab next image. Move to step #1.

Obstacle Avoidance Algorithm





On the road provided, we have 4 obstacles:

1. Random pedestrian girl.
2. End obstacles: middle, left, right.

We recognize the obstacles using 2 IR light sensors on the front of the car. If both sensors are activated, we determine the state as obstacle detected.
After several trials, we found the best design (positions), which is placing the sensors as close together as possible:

After recognition, specific action performed for each obstacle:

- Pedestrian: stop the car, while the obstacle is present.
- End obstacle - middle and right: perform the left maneuver.
- End obstacle - left: perform the right maneuver.

Pseudocode (main)

1. Initialize all flags.
2. Create permission objects to each sensor.
3. Run each sensor on separate thread.
4. Continuously check each flag and permission.
    a. Check traffic light flag: if the flag is on then stop.
    b. Check parking sign flag: if the flag is on then proceed parking action.
    c. Check turn sign flag: if the flag is on, get the direction from sign permissions and turn in this direction.
    d. Check IR tracers' flags: depending on their values change the direction.
    e. Check IR sensors' flags: depending on their values change the direction.
    f. Get direction from lane permissions and move.
5. After finishing stop all sensors and exit.