ARCUV: Autonoumous Robot Car based on Ultrasonic and Visual Sensors

Jui-Chieh Chang
Department of Computer Science and Informational Engineering
National Taiwan University
Taiwan, Taipei
b05902016@ntu.edu.tw

Chon-Seak Lai
Department of Computer Science and Informational Engineering
National Taiwan University
China, Macao
b05902091@ntu.edu.tw

Kuan-Wei Tseng
Department of Mechanical Engineering
National Taiwan University
Taiwan, Taipei
b05505008@ntu.edu.tw

***Abstract* — Autonomous automation is an active field of robotics research. Its research problem contains several subtopics that can be discussed particularly such as computer vision, control system, path planning etc. We developed an autonomous robot car based on ultrasonic and visual sensors as our term project in robotic courses (NTUCSIE5047) which applies the technology above. The robot car is designed, processed, and assembled by ourselves. It is composed of 4 DC motors, camera module, ultrasonic ranging sensor, 2 motor drivers, power source, and a Raspberry Pi 3 Model B+. All computation and decision making is done on the Raspberry Pi. With camera and ultrasonic sensors, it is capable of lane detection and tracking, obstacle avoidance, traffic light detection, and map navigation. When the visual or ultrasonic sensors detect the lane and obstacle inputs, Raspberry Pi will make new control decision and send control signal to motor drivers, where PID control is applied and implemented by PWM. The testing environment is made up of tape and black plastic board to simulate the real word environment. The experimental results show that the unstable quality of sensor input and power supply result in fluctuating performance of the robot car.**

Keywords — Autonomous Car, Mobile Robot, Computer Vision, Control Theory

I. INTRODUCTION

Autonomous automation is an active field of robotics research and a technological trend in the automotive industry. It is an integrated technology from different fields including mechanical engineering, electrical engineering, computer science, etc. Its research problem can be discussed furtherly in each field, including computer and machine vision, control theory, path planning and optimization, and artificial intelligence. In our term project, we made our own self driving robot car which can be considered as the testing platform of the mentioned technology.

ARCUV combines and improves previous works and successfully implement them on a Raspberry Pi robot car. Its feature can be summarized as follow:

- Homemade robot car and hardcore control system programming.
- Automatically controllable without external device for computation.
- Real time lane detection using color filters and Hough line transform and tracking with PID controller.
- Static and Dynamic obstacle detection and avoidance with ultrasonic ranging sensors.
- Map navigation and self-cruising.

The design, processing, and assembly of our own robot car and testing will be discussed in Section III. The lane detection and tracking and traffic light recognition will be elaborated in Section IV. The control logic and system structure will be given in Section V. We will give the complete experimental results and conclusion in Section VI and Section VII respectively.

II. RELATED WORKS

Scholar works on autonomous car related technolgy are extensive. One of the noticable is the The Duckietown Project[1]. It was conceived in 2016 as a graduate class at MIT. Their goal was to build a platform that was small-scale and cute yet still preserved the real scientific challenges inherent in a full-scale real autonomous robot platform.

For the computer vison part, we utilize OpenCV Libraries. Hough line transformation functions in OpenCV [2]. It is used to detect the road lines in the image. The returned linear functions are used to calculate the offsets and deviation for course fixing. [3] provided the concept of the necessary procedures for doing traffic light recognition by image processing. The more detailed implementation is described in their former paper [4], which contains all the algorithms they have used in each part.

III. ROBOT AND ENVIRONMENT SPECIFICATION

ARCUV is designed, processed, and assembled by ourselves. Fig. 1 below shows the 3D modeling and Real world photo of the ARCUV.

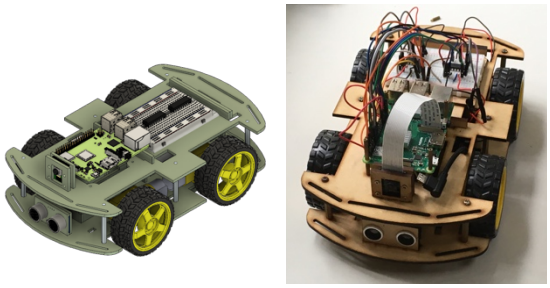

Fig. 1. 3D modeling and photo or ARCUV.

A. Design, Processing and Assembly

The primary materail of the struture is 3mm middle density fiber. We used AutoCAD to draw the 2D engineering graphics and Universal Laser Cutter to cut the board into the desired shape. During the design progress, we measure the size of each commercial component. As you can see, there are two layer in ARCUV, which are named as upper chassis and lower chassis respectively. The 2D CAD screenshot of the lower chassis is shown in Fig.2 to demonstrate the dimension.

There are 4 DC motors with reduction gear (1:48) will be fixed on the lower chassis. Then the toy wheel whose diameter is 65mm will be connected to the DC motors. The Ultrasonic

sensor (HC-SR04) is fixed on the front the lower chassis. Two 9V battery with a fiberboard box is installed behind the ultrasonic sensors. The assembly progress of the lower chassis is shown in Fig. 3 below.

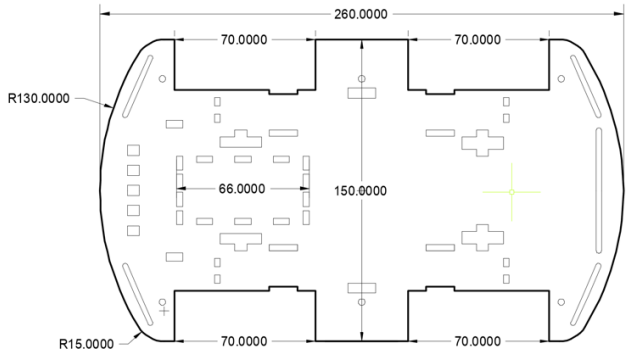

Fig. 2. 2D Engineering Graphics of Lower chassis.

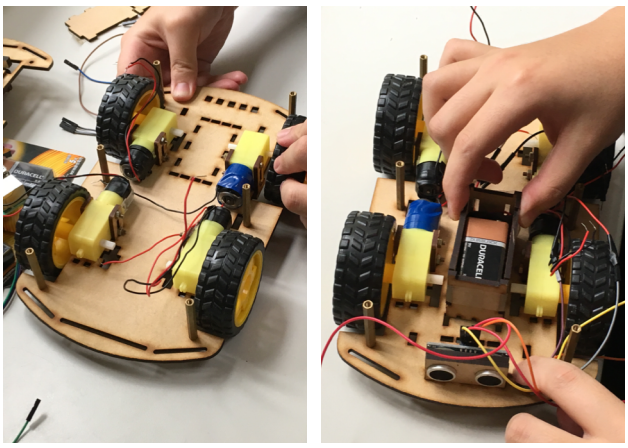

Fig. 3. Lower Chassis Assembly

On the other hand, the Raspberry Pi, Motor Driver IC, breadboard, and camera are installed on the upper chassis. In addition, there is a slim power bank under the upper chassis to provide electricity for Raspberry Pi. Fig 4. below shows the assembly progress of the upper chassis.

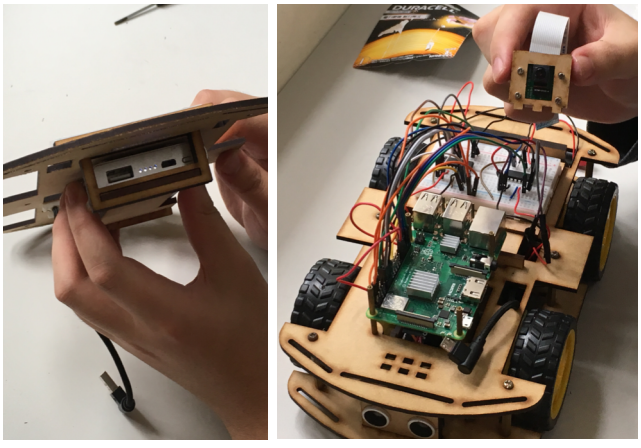

Fig. 4. Upper Chassis Assembly

B. *Hardware Devices*

We will introduce our development board: Raspberry Pi Model 3 B+, camera: Camera Module V2, DC motor driver IC: L293D, ultrasonic ranging sensor: HC-SR04, and power source: Verbatim Power Bank and Duracell 9V battery.

Raspberry Pi 3 Model B+: It is a small single-board computer, which can be driven by Linux operating system, developed in the United Kingdom by the Raspberry Pi Foundation. This model is the latest product in the Raspberry Pi range. It has 40 GPIO pins (General purpose input/output) which enable to control electronic devices or receive sensors input. The specification of Raspberry Pi 3 Model B+ retreived from its official data sheet is shown in Table 1 below [5].

TABLE I. RASPBERRY PI 3 MODEL B+ SPECIFIATIONS

CPU	1.4GHz Quad-Core ARM Cortex-A53 (64Bit)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0; hardware-accelerated OpenVG; and 1080p30 H.264 high-profile decode
Memory	1GB LPDDR2
Storage	Micro SD card (16GB~)
Wireless	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN
GPIO	40-pin 2.54 mm (100 mil) expansion header: 2×20 strip
Working Current	459mA(2.295W) when idle 1.13A(5.661W) maximum under stress
Size	85mm x 56mm x 17mm

Raspberry Pi Camera Module v2: The Raspberry Pi Camera Module is an official product from the Raspberry Pi Foundation. The original 5-megapixel model was released in 2013, and an 8-megapixel Camera Module v2 was released in 2016. For both iterations, there are visible light and infrared versions [6]. Here we use the visible light v2 module. It has a Sony IMX219 8-megapixel sensor, which supports 1080p30, 720p60 and VGA90 video modes, as well as still capture. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi [7].

Motor Driver IC L293D: It is a high current motor driver and controller Integrated Circuit. It is capable of driving load current up to 1A at the voltage ranging from 4.5V to 36V. It can receive a low current signal as an input and offer high current signal to the motor. Thus, it is commonly used as a current amplifier. In addition, it has two builtin H-Bridge driver circuits to make it possible to control two DC motors at a time in both clockwise and counter clockwise direction. Since there are 4 DC motors in ARCUV, we need two L293D to drive and control them .

Ultrasonic Ranging Sensor HC-SR04: The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package [8]. It emits ultrasound at 40000 Hz which travels through the air and if there is an obstacle on its path it will be reflected to the receiver. By considering the travel time and the speed of the sound, we can simply calculate the current distance between the obstacle and vehicle. However, the effective range of this module is narrow, it can detect 45 degree range but only 30 degree range is claimed effectual. Fig.5 shows the photo of HC-SR04 and its effectual range.

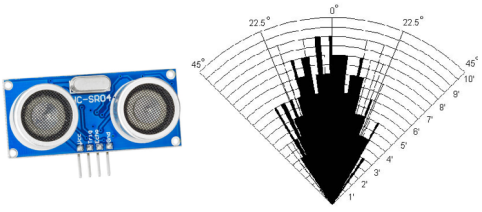

Fig. 5. HC-SR04 Ultrasonic Ranging Sensors

Power Sourcr: There are two 9V Duracell batteries on ARCUV, providing electricity to the L293D IC. As for the Raspberry Pi, we choose Verbatim 6000mAh Power Pack as input power since it is slim and able to provide maximum 2.4A current to the Raspberry Pi. Though 2.5A is recommended, we think that 0.1A result in few impact to our robot car.

C. *Testing Environement*

Our idea is to build a model of city which contains lanes and such infrastructure that is capable to be the test bench for our project. The city is constructed by corrugated board. The material we considered for the road was thick paper cardboard at first. However, due to the cost and the amount we need, we have substituted that for a plastic corrugated board, which is more reflective and has lower friction.

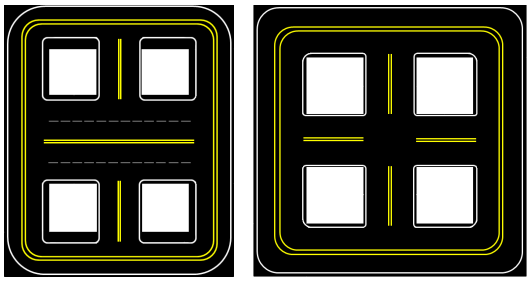

Fig. 6. The city design blueprint for the 1st and 2nd versrion

First Version: The city consists of four corners, four T-shape turn and a double two-way traffic intersection. In this version, the connection between the T-shape turn and the intersection is unnatural because such structure does not exist in real world. We have found that turning in this condition is complex and hard to be handled. Based on this design, we have made some modifications and produced the second version for our city.

Second Version: We have remained most of the design in the previous version but simplified the intersection into a single two-way traffic intersection. The ratio of the lane is also adjusted to make the environment more realistic.

Lane Details: The yellow and white tapes simply represented the white line and double yellow line for the lane. They are separated by 17 centimeters, which is enough for the car to pass through. Before every turn, intersection or corner, blue horizontal lines are added in order to help the car in detecting the distance and time needed to make the turn. It can also provide the timing to do the traffic light detection. Fig. 7 below demonstrates the lane details.

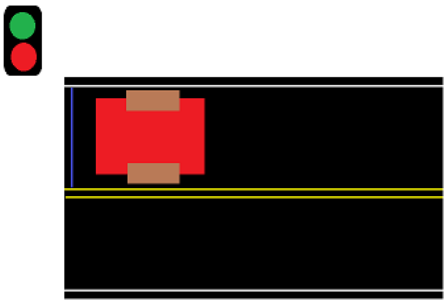

Fig. 7. Regulations for Colored Tape

Traffic Lights: We use 3D printer to make a traffic light's support, three kinds of LEDs (red, yellow, green) are inserted and fixed on the traffic light. Connected with wire are battery, we can manually control the traffic light.

Fig. 8. Photo of Real World Environment

IV. Vision and Image Processing

A. *Lane Detection using Hough Line Transform*

In order for the autonomous car to drive on the road safely, being able to detect road lines is a necessity. In our city, most of the roads are straight lines, so using Hough line transform to find out the road lines will be an efficient and precise method.

Road line detection is done by finding the intersection of two independently processed images, then use the intersection as the input of Hough line transform function.

On the one hand, we first use color filters to filter out all the yellow or white objects, since they are the color of road lines and thus are the only colors that we are interested in. The filtering is done by applying thresholds to the RGB input image.

$$y_{marked} = \{(x,y) \mid (x,y) \in img, R(x,y) > y_{R,Threshold}, R(x,y) > y_{G,Threshold}, R(x,y) < y_{B,Threshold}\}$$

$$w_{marked} = \{(x,y) \mid (x,y) \in img, R(x,y) > w_{R,Threshold}, R(x,y) > w_{G,Threshold}, R(x,y) > w_{B,Threshold}\}$$

On the other hand, Canny edge detector is applied to the grayscale image to extract the edges of the image.

After applying some noise removing functions to the two outputs, the intersection of these two images is taken. The intersection of y_{marked} and Canny edges $\mathbb{C}$ will be all the "yellow edges". The intersection of w_marked and Canny edges will be all the "white edges".

$$y_{edges} = \{(x,y) \mid (x,y) \in y_{marked} \wedge (x,y) \in \mathbb{C}\}$$

$$w_{edges} = \{(x,y) \mid (x,y) \in w_{marked} \wedge (x,y) \in \mathbb{C}\}$$

As for the last step of road line detection, Hough line transform is used. After applying Hough line transform on y_edges, the output will be all the "straight yellow lines". On the other hand, applying Hough line transform on w_edges, the output will be all the "straight white lines". Using the line equation of the white line and the yellow line, we can then easily calculate their separate offset and find the middle point of the road. With the middle point found, we can get the deviation by calculating the difference between the middle road point and the middle of the image, which can later be

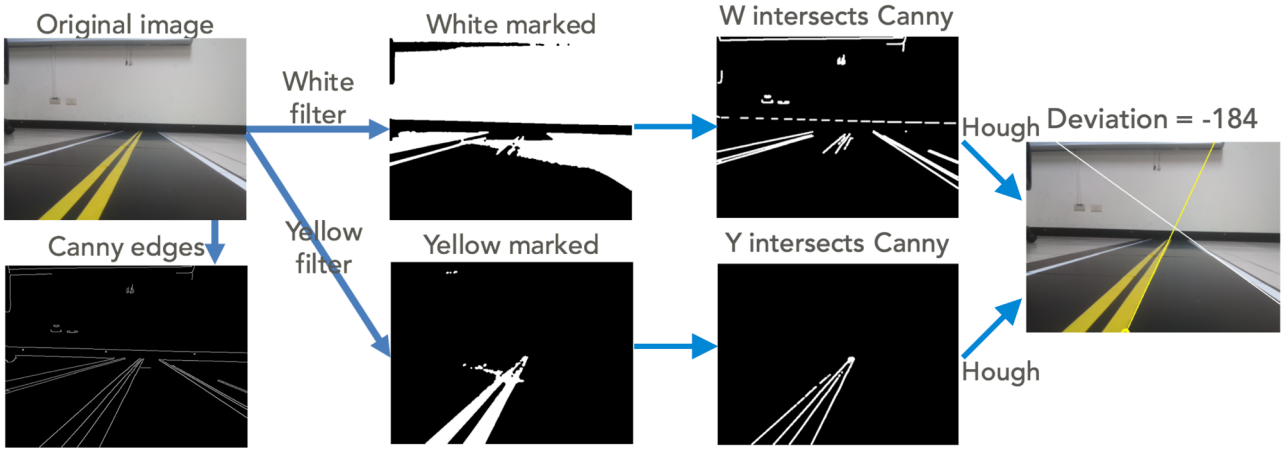

Fig. 9. Road Line Detection Process

used to revise the car's course for it to stay in the middle of the road.

B. Traffic Light Dectection and Recognition

The spot light is a common feature for traffic lights. The position of traffic light could be probably estimated by detecting the spot lights in the image.

Fig.1 below shows the skeleton of the algorithm we used for the spot light detection. The difficulty is to speed the computation time to give enough time for the car to react to the light after a turn or a crossroad is detected. Speed would be given first priority in consideration.

Fig. 10. Main Procedure for traffic light detection.

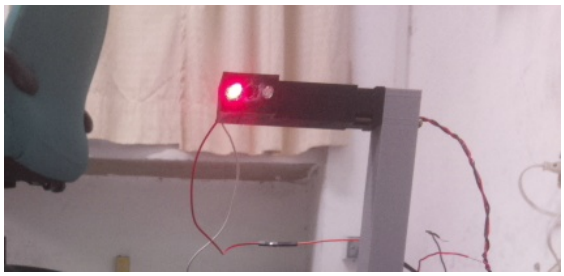

Fig. 11. Substitutional Traffic Light we used in our testing environment.

The traffic light we used is shown in Fig. 2. The position of the light is significantly the brightest part of the image. A simple algorithm is used to spot such region. A 5x5 Gaussian kernel is used to blur the image, which can be defined as:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

This is an alternative method for reducing the noise in the image, which helps for handling the discrete pixels. The difference could be seen clearly after thresholding. Fig. 12 shows the comporison of two conditions. The image after Gaussian blur is shown in the right picture. Compared to the image without blurring in the right, the number of connected objects is obviously less after blurring, which may affect the computing speed of the algorithm.

Fig. 12. Results of thresholding after blurring or not. (Left: raw image. Right: with Gaussian blur).

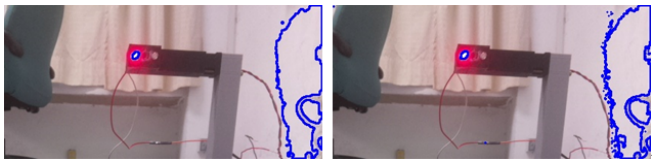

Fig. 13. Results of thresholding after blurring or not. (Left: raw image. Right: with Gaussian blur).

After receiving the binary image, we could start to get the contours of the objects in the image and select the one belongs to the traffic light. As mentioned above, the number of objects may elongate the time in finding contours. For poor noise removal, the spending time could be unpredictable. Fig.4 show the results of the images after find contours is done that with filtering or not respectively. The program is wasting time in processing such small pieces in the right picture. Moreover, different threshold values also causing the existence of chunks. A high threshold may increase the probability of breaking a large chunk into numerous pieces. In contrast, a low threshold may produce too many objects to be handled.

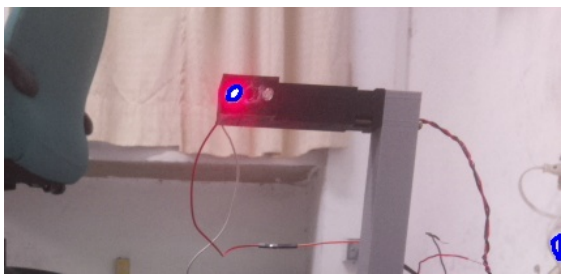

Fig. 14. The left contours after removing those which are out of boundaries

As the traffic light detection runs every time a crossroad or a turn is detected. The distance of the car to the light can also be estimated. For this reason, the area of the spot light

could also be estimated in a range of values. The leftovers are considered as the spot light and start the detection of color.

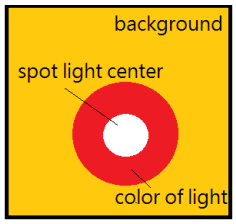

Fig. 15. The Layer of Spot Light Represented in the Image

The contours we received was the brightest part of the light. However, the emitting color of the light is contained by the aperture surrounding the white part. In order to retrieve the color, a buffer area should be retained for detection. Two color masks, one for green and one for red, are used to receive the color pixels which are satisfy in the range of HSV value in the cut out area. By counting the percentage of such color pixels, the light is defined as "detected" if it exceeds the threshold of green/red pixels' percentage.

V. Control Logic and Decision Making

In this section, we will elaborate on the control systems and the decision rule of ARCUV. ARCUV is 4-motrized wheel robot car which apply differential steering. The speed of wheel is altered by L293D which receives PWM signal from Raspberry Pi and Electricity from 9V batteries.

A. PWM signals

As mentioned in Section III, we use L293D to control the motors' speed. These command are sent by the Raspberry Pi with Pulse Width Modulation. By changing the duty cycle of the signal sent by the Raspberry Pi, we can change the output voltage at L293D to change the speed of car.

B. PID controllers

PID controller is a practical feedback control method in classical control theory. It calculates the "error" value as the difference between a measured input (feedback) and a desired target. The controllers then attempt to minimize the error by adjusting the output power. As for ARCUV, the error is deviation calculated in Section IV. The following equation is the standard form of the PID controller.

$$u(t) = K_p e(t) + \int K_I e(t)dt + K_D \frac{de(t)}{dt}$$

where K_p is the proportional gain factor, K_I is the integral gain factor and K_D is the differential gain factor.

There are three terms in above equation. They can be understood as the error of current (proportional), past (integral), and future (differential). PID tuning is an annoying process since we don't have the mathematical modeling of motors. Finally, we found that $K_p = 0.5, K_I = 0, K_D = 0.1$ is the optimal result. It can fix the deviation fast without too many overshoot. We still set saturation a duty cycle (maximum 80%, minimum 20%) to prevent acute modification which may cause ARCUV lose track of the road lines.

C. Map Constrution and Navigation

The autonomous car should know its position in order to decide what kind of choices it should make when it is at an intersection. Our city is in the shape of the Chinese character "田". For the carrying out its decision at the intersection, the car doesn't need to know its exact position in the city. It only needs to know which "road" it is on and in which "direction" it is going. Since our city is perfectly symmetric, we will only need several "states" to represent the position of the car. The transition of the states when an action (i.e. left turn, right turn, straight) is taken is called as an "update" function, which works pretty much like a finite state machine.

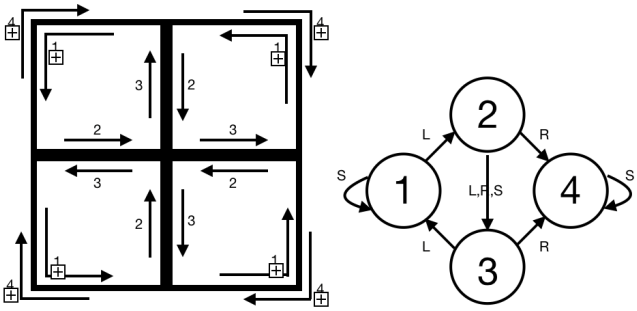

Fig. 16. City roads and the Corresponding States / Finite State Diagram

For example, if the car is currently at state 1, according to the city roads, it is going counterclockwise around the city. When it reaches a crossroad, according to the finite state diagram, it can either choose to go straight to stay at state 1 or turn left to enter state 2.

D. Obstacle Detection & Avoidance

An autonomous car should avoid traffic accident. We use ultrasonic sensor to detect the distance between the obstacle. Since ultrasonic sensors has higher frame rate and less calculation complexity, it is more suitable than camera to perform obstacle detection and avoidance task. If the distance is too close, ARCUV will stop. If the obstacle is leaving (increasing distance), ARCUV will try to resume its original plan. However, the effectual range is relatively narrow. Due to size limit, we cannot add more sensors. Thus, we can only detect the obstacle in the front of ARCUV.

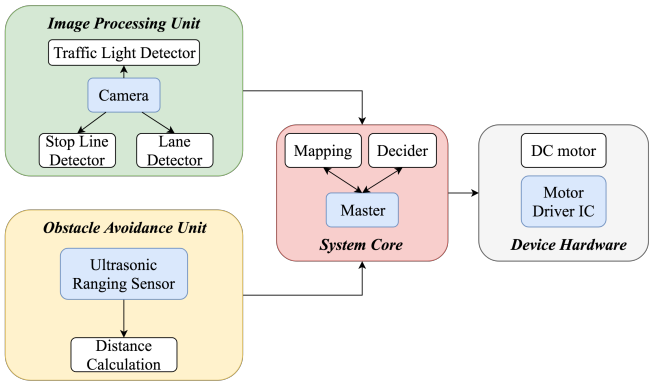

Fig. 17. System Struture

VI. Experimental Results

During our experiment, we found some notice problems that can and should be solved.

A. Battery Level:

We were using 9V batteries for our autonomous car, one for each two motors. During our experiments, we found out that the motors' power depends highly on the battery level.

The autonomous car's speed is managed through setting the duty cycles (0~100) of the left and right wheels. When we are using a new fully charged battery, the car can run on a medium speed with the duty cycle set to 30. After running the same battery for about ten minutes, in order for the car to maintain the same speed, the duty cycles might need be set to 80. The power of the motors drops drastically with the battery level.

When we are testing out the correct parameters to set, the power of the motors is also dropping. The inconsistency of the battery level thus leads to the difficulties of setting the precise parameters for turning and fixing the deviation. The power of the motors is the most important part of the control of our autonomous car since it's the only way decide the car's behavior.

The problem might be solved by using voltage regulators to stabilize the voltage output of the batteries and thus the motors can output the same power independent of the battery level.

B. Low Frame Rate:

The autonomous car's behavior is highly based on vision. By taking pictures using the front camera, the car is capable of detecting road lines and traffic lights. In our project, a picture is taken every 0.2 seconds (fps = 5) and the road lines and traffic lights are detected using image processing functions. Of course it would be better if we could set the frame rate to be higher since it would give the autonomous car a more precise and updated information about the environment. However, the time between two pictures taken should be at least longer than the processing time of the road line detection and traffic lights detection functions. If the frame rate is too high, there won't be enough time to process the image.

We have been trying to lower the processing time of road line detection and traffic light detection by using threads to parallelize the processes. However, 0.2 seconds is the best we can do for now. The long processing time might be the inevitable result of running image processing functions with python on a Raspberry Pi processor.

C. Image Quality:

During our experiment, we found out that the image quality of the camera was inconsistent. When the car is running at a constant speed and the brightness of the environment is the same, some frames are a lot darker than the others and some frames have serious motion blur issues. Poor image quality puts a burden on road line and traffic light detection functions, especially when it happens randomly. We couldn't predict if the picture currently taken is dark or blurred or not, so there is no way to recover from these kind of situations. When a defective image is given to road line and traffic light detection functions, they might output incorrect results and the car might behave erroneously.

The problem might be solved effectively by replacing the camera hardware with a better one which suffers from a lower probability of taking defective pictures.

D. Environment:

The environment may affect our autonomous car in two ways. First, it's about the vision of the car. Just like the problem about the image quality. If the background light is too strong (i.e. sun light), the image taken by the camera might be too bright. Since our "road" is made of reflective materials, the road itself would also appear to be a light source, overexposure thus would cause trouble to the road line and traffic light detection functions. Second, the physical features of the environment will also affect the car's behavior. Since our autonomous car is not steerable, turning is done purely by setting the speed difference between left and right wheels. The turning behavior of the car depends highly on the friction of the road its running on. Even a little difference of the friction can cause a huge difference to the outcome of a turn. Aside from the inconsistent motor output, the environment is also a factor that causes the unpredictable behavior of the car.

VII. CONCLUSIONS

We couldn't say that this project is a success since we didn't carry out all the goal that we promised, but we would say that it is a precious experience. In the progress of designing and implementing, we ran into a lot of problems, both in hardware and software. Our goal is to build a car on our own and write a code for it to run safely in our designed city. Normally, when the code is run a consistent and stable hardware, like a PC or server, the only thing we need to care about is whether the code carries out the correct result. However, in this project, our code is run on a self-constructed car. The compatibility of the software and hardware thus becomes a serious problem. There is no longer "correct" answer and the bug doesn't only come from the error of the code. Camera defection, low battery level, environment friction, all these things may cause problem to the car's behavior and add difficulties to our debug progress. Compared to other robots like Pioneer or ITRI-arm, the precision and robustness of our self-constructed robot still have a lot of room for improvement.

REFERENCES

[1] MIT: "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research" ICRA 2017

[2] "OpenCV 3.0.0 documentation - Hough Line Transform," https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html.K. Elissa, unpublished.

[3] Raoul de Charette, Fawzi Nashashibi: "Traffic light recognition using image processing compared to learning process" 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems

[4] Raoul de Charette, Fawzi Nashashibi: "Real time visual traffic lights recognition based on Spot Light Detection and adaptive traffic lights templates" 2009 IEEE Intelligent Vehicles Symposium

[5] "Raspberry Pi 3 Model B+ datasheet", https://raspberrypi-tw.s3.amazonaws.com/datasheet/rpi_3b_plus_spec.pdf, raspberrypi.org

[6] "Camera module v2", https://www.raspberrypi.org/products/camera-module-v2/, raspberrypi.org

[7] "Camera module",https://www.raspberrypi.org/documentation/hardware/camera/, Raspberry Pi Documentation

[8] "Ultrasonic Sensor HC-SR04 and Arduino Tutorial", https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/, Dejan, How to mechatronics

[9] Michael Sipser, "Introduction to the theory of computation_third edition," pp. 47-53, 2013.

[10] Farid Golnarsghi, Benjimin C. Kuo, "Automatic Control Systems_tenth edition", pp.707-712, 2017.

[11] Guo Yuqing, Guo Yanning, Xu Hang, Feng Zhen, "Vision-Based Detection and Tracking of a Moving Target for a Smart Car". IEEE 3rd International Conference on Image, Vision and Computing(ICIVC), pp. 586-591, 2018

[12] Rafael C.Gonzalez, Richard E. Woods, "Digital Image Processing_third edition", pp.627-645, 2008.