Nengo Empowered Self-Driving Car with Linux, ROS and OpenCV on Raspberry Pi 3 and PC

Advisor:
Prof. Shyh-Kang Jeng

Author:
Chung-Yuan Chen, Tzu-Wei Liu, Ruei-Kai Cheng

Course: Computational Cognitive Neuroscience 2017

National Taiwan University
2017/06/22

Abstract

In this work, a self-driving car with cognitive neuroscience is implemented. The car equipped with camera and other sensors as input. It could process surrounding information into perceptions with OpenCV, and then make decision on the fly with Nengo brain model. Finally, the brain sent out commands to Robotic Operating System to steer the car. The implementation of the system is well described in this report along with experiment condition, result and analysis. This work shows the possibility of incorporating cognitive neuroscience with automobile and robotics technology to develop more advance system.

Contents

Abstract
1. Introduction
   2.1 Cognitive Neural Network with Nengo
   2.2 Image Preprocessing with OpenCV
   2.3 Decision-Making Architecture
   2.4 Robotic Operating System
2. Implementation
   2.1 The Computational Platform
   2.2 The Desktop PC
   2.3 Remote connection
   2.4 Raspberry Pi 3 and ROS
   2.5 Additional Microcontroller
   2.6 Preprocessing with OpenCV
   2.7 Nengo SPA Model
3. Experiment
   3.1 Design of the Trial
   3.2 Experiment Result
4. Discussion
   4.1 Design of the SPA Model
   4.2 Difficulties in Visual Processing
5. Conclusion
6. Appendix

1. Introduction

1.1 Cognitive Neural Network with Nengo

A key feature of this project is adopting the Neural Engineering Framework (NEF) and its implementation, Nengo modeling and simulation tool, as the central cognitive behavior processor. Furthermore, the Semantic Pointer Architecture (SPA) of Nengo toolkit is used to focus the development of higher level cognitive tasks management. With the help of computer vision package, OpenCV, the images taken by the front camera get pre-processed and then ported into semantic pointers to excite the SPA brain model. The Nengo model not only provides ways of studying how human's brain performing cognitive tasks but also enable the possibilities of incorporating the essence of the biology with other applications, such as robotics here.

1.2 Image Preprocessing with OpenCV

The image captured by the camera is just an array of raw digital numbers which is not quite informative to the system. Thus, OpenCV is used to extract information the car concerned from the image. The output of the processed image are semantic pointers which describe the feature in the image. The tools of OpenCV used here are hsv color conversion, color filter and contour extraction.

1.3 Decision-Making Architecture

In the decision making, we use the Nengo SPA model and design the rule to handle various situation.

Figure 1 show our architecture, the vision part which is in charge of receive the data from the OpenCV's output, like RED, GREEN, NEAR and so on. After receiving the vision data, the SPA model will parse it and make a motion, then it will send to output which is a Nengo network Node, we use this Node is purpose to make the message call back, then we can take it to the ROS to control our self-driving car.

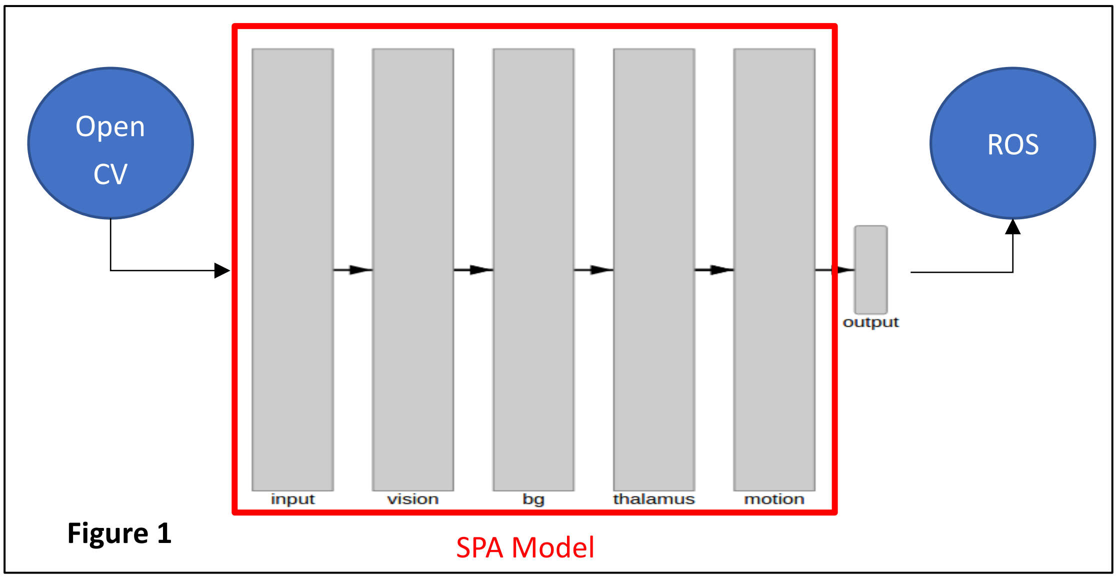


Figure 1

1.4 Robotic Operating System

In order to collectively manage the self-driving car's physical hardware, Robotic Operating System (ROS) architecture was chosen to take care of all the concurrent events. Besides to some of the core process of ROS needed, the custom functionalities are implemented by creating nodes of process where each node is only responsible for one specific task. These nodes exchange information via 'message' publishing and subscribing to different 'topics' held by ROS core, or 'service' method -- the direct connecting scheme. These architectural features of ROS reduce the difficulties of programing a robotic system and provide modular management of functionalities.

2. Implementation

Figure 2 reveals the physical construction of the Nengo empowered self-driving car from the front view. The detailed parts list and corresponding description are depicted in **Table 1,** they are cross referenced by the index number.

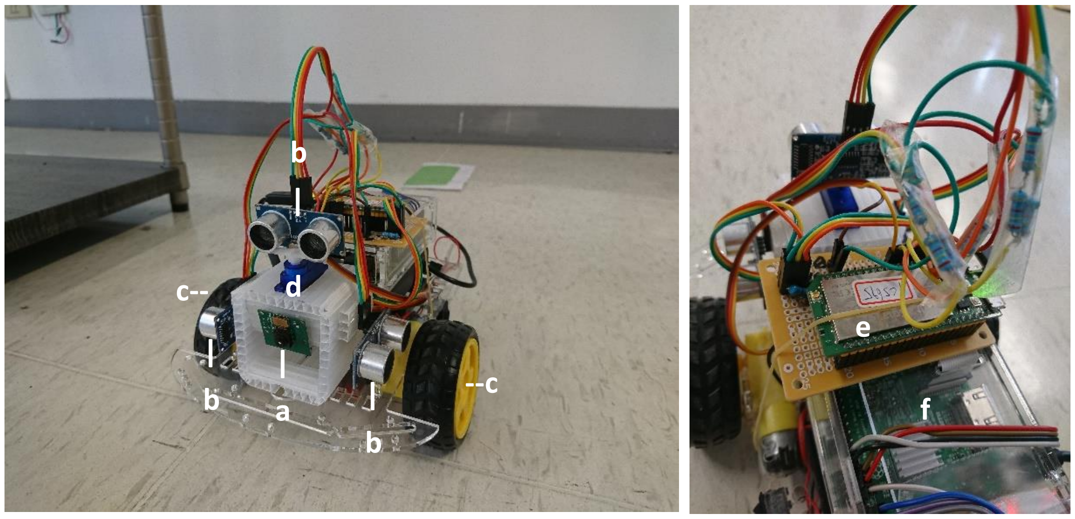


Figure 2, The pictures of the Nengo empowered self-driving car. (Left) picture taken at the slightly angled front. (Right) picture taken at the top shows the Raspberry pi 3 microprocessor and the ATmega32U4 microcontroller board.

INDEX	LABEL	DESCRIPTION
a	1* 5MP Camera	Capture images.
b	3* Ultrasonic proximity sensor	Detect proximity at left, front, right.
c	2* DC motor wheels	Drive the car.
d	1* Servo Motor	To pan the front proximity sensor.
e	ATMEL's ATmega32U4	Microcontroller.
f	Raspberry Pi 3	Microprocessor.
g*	2* IR optical blocking sensor	Wheel motion sensing.
h*	1* H-bridge DC motor driver	To drive the motor with control signal.

Table 1, The corresponding part list and the description to indexed item in the Figure.
*: not shown in the picture.

2.1 The Computational Platform

For the overall system to work, there is a PC with Ubuntu operating system as the 'brain' of the self-driving car, and a Raspberry Pi 3 microprocessor with Raspbian operating system and ROS as the controller of the car. The main reason of this setup is that it is required to have a powerful computing platform to enable neural network simulation of the brain, but the car also needed to be free to travel around. The block diagram of the overall computational platform is shown in **Figure 3**.

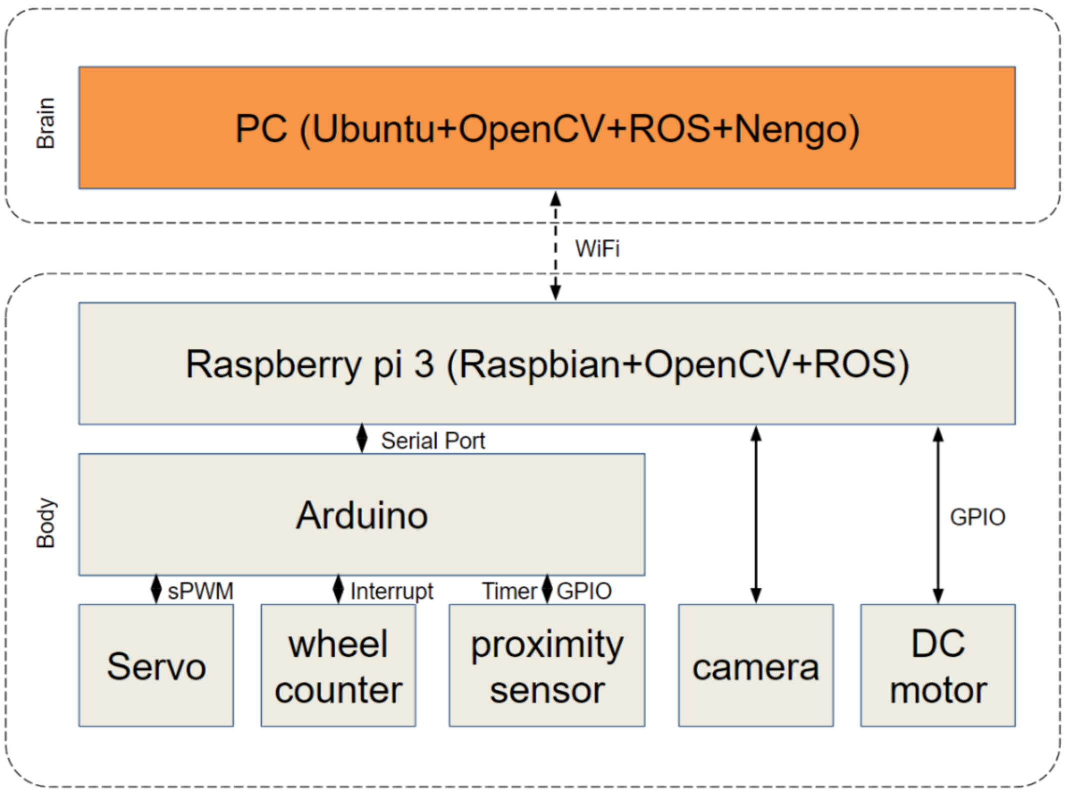


Figure 3, The block diagram of the Nengo empowered self-driving car system. The hierarchy of the signal flow is well demonstrated.

In addition, a full-fledged desktop PC equipped with eight logic core INTEL's CPU, NVIDIA's external graphics card and large memory is beneficial for development, debugging and real-time result visualization. To be specific, the 'brain activity' refers to the decision-making process of the car, which is modeled by a biologically plausible neural networks simulation tool, Nengo. The linkage between the two systems (the PC as powerful 'brain' and the Raspberry Pi 3 as 'spine') is over wireless local area network connection (WLAN, Wi-Fi technology).

2.2 The Desktop PC

The powerful desktop PC is installed with Ubuntu OS 16.04, OpenCV 3.2.0, Nengo 2.4 and ROS Kinetic, which were tricky to setup to work coherently. Once the system is ready, a main python script is developed to glue all the software module above and defines the Nengo brain model. The model here handles the decision-making of the self-driving car which will be further discussed in another paragraph. The script gets run in the GUI version of Nengo simulator which can run the neural network model with live data input, live result output and network status visualizations.

2.3 Remote connection

A User Datagram Protocol (UDP) socket receiver is setup to listen to the sensing data streamed from the remote car. The UDP packet of the connection contain the values of the proximity sensors, wheel movement sensors and image taken by the camera on remote car, these are the information for the brain model after some pre-processing. For instance, the image is first decoded from JPEG compression format and parsed into perceptions by OpenCV functions. The perceptions then excite the brain model. A Transmission Control Protocol (TCP) socket client is established to connecting to TCP server on the remote car for controlling. The usage of the TCP socket over UDP one is to ensure the transmission of the control commands and get the acknowledge of the operation conducted. In addition, Python's 'threading' package is utilized to deal with the concurrent task execution between the UDP receiver, TCP client and the Nengo model real-time simulation on the PC side.

2.4 Raspberry Pi 3 and ROS

As to the 'spine' of the self-driving car, the Raspberry Pi 3 microprocessor is installed with Raspbian Jessie OS as basis and get ROS installed and launched above it. The ROS system consist of one core service (roscore) and several 'nodes' which are the basic functional units run parallel in ROS architecture. They're programed in python script to perform various tasks. For instance, the socket connections to the remote desktop for brain simulation are handled by tcp_server and udp_server node, and the serial communication to the microcontroller is handled by mcu_bridge node. The motor_control node accesses the GPIO pins on Raspberry Pi board to control the H-bridge and then the motors. The full list of the nodes developed in this project and their description are depicted in **Table 2.** Information and control signals in side ROS are exchanged via 'message' publishing and subscribing or 'services' connection between nodes. The 'message' and 'service' available in this project are listed in **Appendix-Table 4**. To wrap the whole things up, a ROS launch file was created to collectively run the node scripts needed for the project.

Node	*Description*
camera.py	Capture image from camera at approx. 10Hz. Then publish the compressed image stream to 'RaspiCarCamera' topic.
mcu_bridge.py	Handle the serial connection to the external microcontroller and gather the sensing data from it. The collected information get pack into relative messages and publish to 'RaspiCarDistance,' and 'RaspiCarWheel' topics. This node also subscribes to ' RaspiCarServo' topic and transport the data.
motor_control.py	Serve 'RaspiCarMotorControl' requests and control the motor accordingly.
tcp_server.py	Serve command from the 'brain' of the car and transport it to action.
udp_server.py	Pack sensing data along with compressed image into UPD packet and send it to the 'brain'.
view_topics.py	Show the messages for debug purpose.

Table 2, The node processes run in ROS system on the remote car.

2.5 Additional Microcontroller

An ATMEL's ATmega32U4 microcontroller is connect to the Raspberry Pi 3 via serial-port over USB as co-processor to handle the timing-critical signal. For instance, the ultrasonic proximity sensor's echo signal is polled by hardware timer routine at 58 us interval for 1 cm resolution, and the wheel motion is captured by IR blocking switches which trigger the hardware GPIO interrupt. The microcontroller also generates PWM signal for the servo motor. There is a node script in ROS named 'mcu_bridge.py' to handle the serial port connectivity and data parsing.

2.6 Preprocessing with OpenCV

Figure 4 shows the input image. The goal here is to find (a) the color of the light, (b) the position of the light and (c) the distance from the car to the light. Then return a semantic pointer of these information.

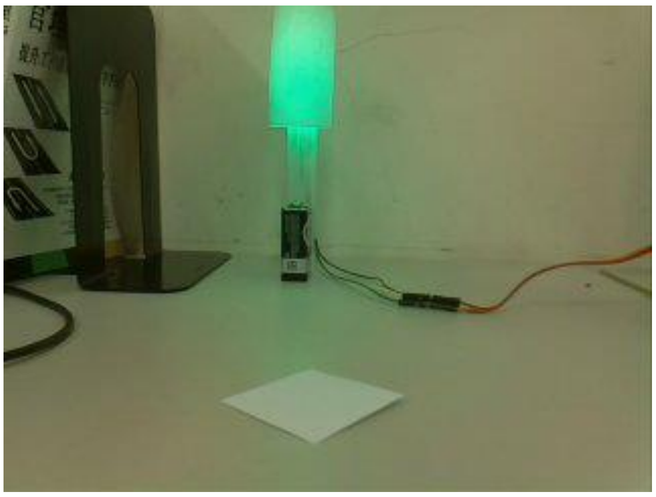


Figure 4, Input image to the OpenCV preprocessing.

First, since the picture is captured by the camera, it contains a lot of noises which will affect the results. To get rid of the noises, the image will be preprocessed, making it easier to extract features. Then we can use threshold to filter unwanted color and find candidate region where the traffic light located at. In our test condition, there is only one colored-light source. So, we choose the biggest region as the light position. Finally, the distance can be estimated by doing a linear transform to the height of the selected contour. The flowchart is shown in **Figure 5**.

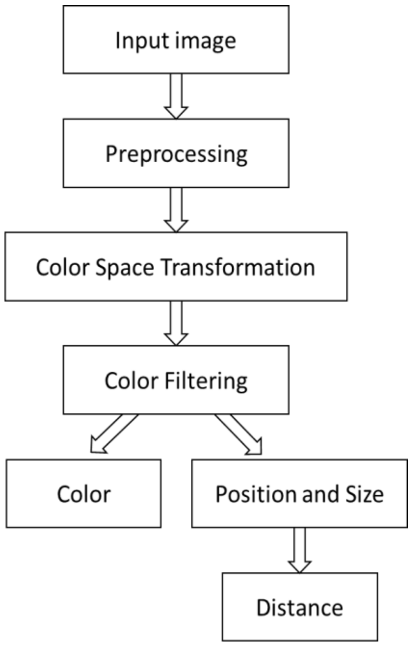


Figure 5, Image process flowchart.

i. Preprocessing

Our visual input is the image taken by a 5-million-pixel camera. But for the consideration of computing performance, we reduced the size of the image to 320 * 240. The compression of the image will induce some noises. To deal with it, we use Gaussian filter to smooth the image, making each color block in the image more consistent. In addition, since the traffic light won’t beneath the horizon, we only detect the part which is above the horizon.

ii. *Color Filtering*

Most of the time, the traffic light has the most intensive color in particular color range. We want to use this property to find the color and position of the traffic light. However, it is not intuitive for us to know how bright a pixel is in a RGB format image. So, we converted it into a HSV format image. The transformation is as followed:

$$\mathrm{V} = \max(\mathrm{R}, \mathrm{G}, \mathrm{B})$$

$$\mathrm{S} = \begin{cases} \dfrac{V - \min(R,G,B)}{V}, & if\ V \neq 0 \\ 0, & otherwise \end{cases}$$

$$\mathrm{H} = \begin{cases} \dfrac{60(G-B)}{V - \min(R,G,B)}, & if\ V = R \\ 120 + \dfrac{60(B-R)}{V - \min(R,G,B)}, & if\ V = G \\ 240 + \dfrac{60(B-R)}{V - \min(R,G,B)}, & if\ V = B \end{cases}$$

After the transformation, we can easily set the threshold by tuning the V value to find brightest part of the image. **Table 3** shows the threshold we set for each light signal. Because that we use 8-bit image, the range of H value is 0~180 instead of 0~ 360.

	H	S	V
Red	170~179, 0~15	>80	>170
Green	45~80	>100	>200

Table 3, The threshold values for color detection.

iii. *Position and Distance Estimation*

In the second process, we obtained the contour of the traffic light (by selecting the maximum candidate region). We can use this information to find the position of the traffic light. For that we don't want the light getting out of the sight. If the light is too left, it will return a 'LEFT' semantic pointer, vice versa. Moreover, the height of the bounding box and the real height of the light can be used to estimate the distance between car and traffic light. The formula is:

$$\text{distance} = \text{real height(pixel)} \times \frac{bounding\ box\ height(cm)}{320(pixel)}$$

If the distance is more than 30cm, it will return a 'FAR' semantic pointer which means the car is far from the traffic light. Otherwise, it will return 'NEAR'.

2.7 Nengo SPA Model

In SPA model, when we got the message from OpenCV, these messages will become a Sematic Pointer. Sematic Pointer is a high dimension vector that we can consider as a concept. In BasalGanglia, according to these vectors it can compute the Dot Product between the actions, after the dot operation it will choose the biggest one as its action. The actions we defined are using for control our self-driving car like FORWARD, LEFT, RIGHT and so on. After choosing the action, it will send to Thalamus which can take the action to the correspond state, in our case is motion. Then send it to output Node.

3. Experiment

3.1 Design of the Trial

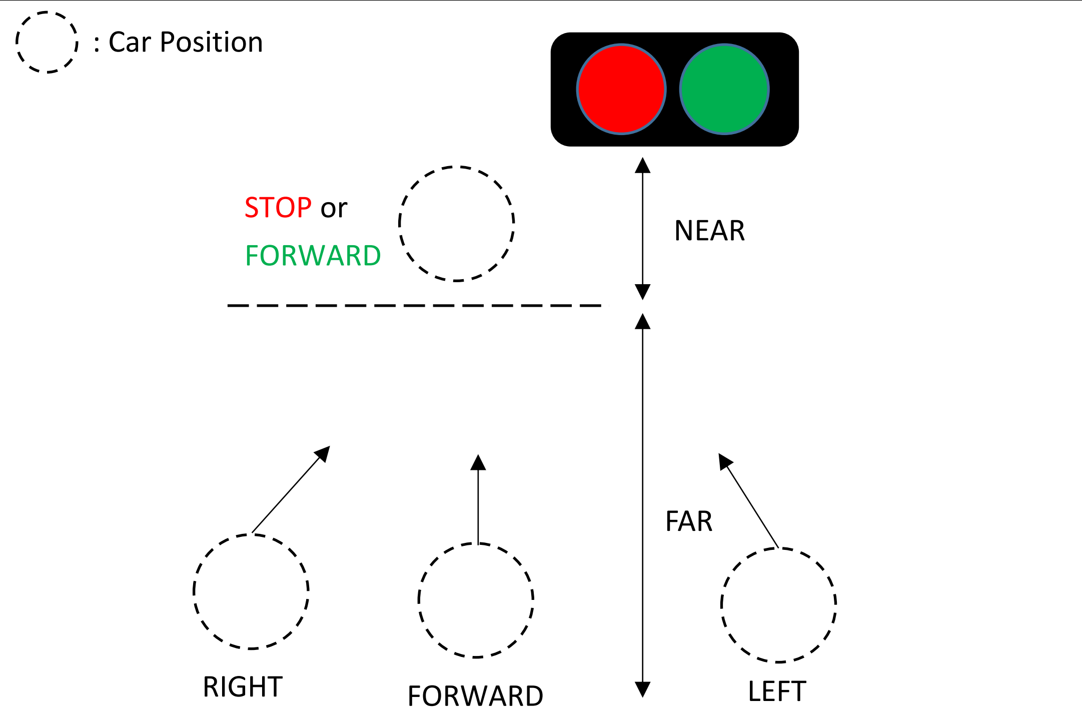
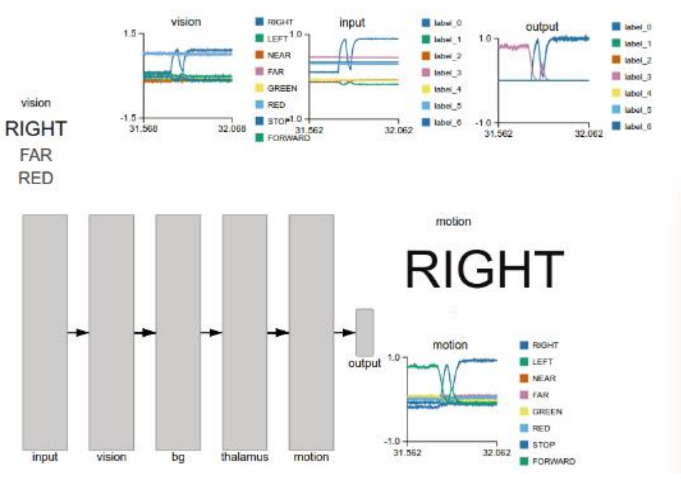
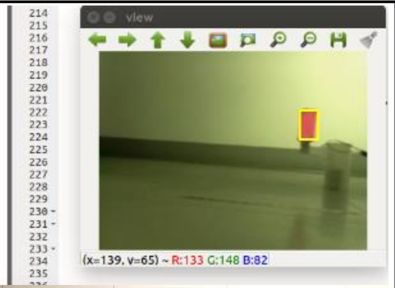


Figure 6, The design of the trial place. The site is divided into two areas according to the distance to the traffic light.

To test the Nengo empowered self-driving car, the main goal is to examine whether the system can react to the real world in real time. For instance, let the car to march to the traffic light when the distance is still far. On the way to the stop line in front the traffic light, the car would steer its direction if where it's heading is off. Once the car is around the place of the stop line, it would stop and check the traffic light, if the sign is red, the car stays, otherwise drives pass through. The initial position of the car is around 1.5m away from the traffic light, it front is facing the sign for the camera to capture the image.

3.2 Experiment Result





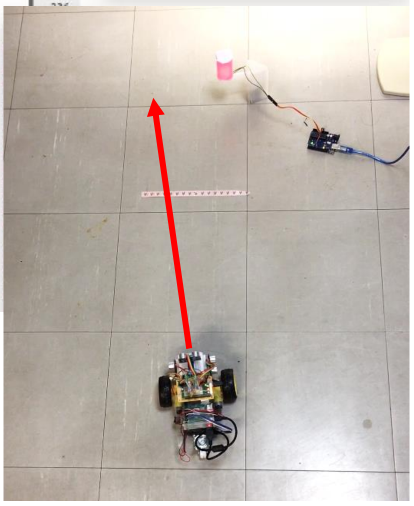


Figure 7, The inspection of the result at “RIGHT” decision. The left side of the figure shows the Nengo’s dash board and the top right corner is the image captured by the camera. The bottom right is a photo taken at the top of the trial place.

In **Figure 7**, the instant state of the system is captured. In this moment, the car saw the red traffic light at the right side of the view, so the input to the Nengo brain model is “RIGHT+FAR+RED” which can be observed at the word cloud graph of the ‘vision’ state object. According to the model design, the decision output of the model is “RIGHT” semantic pointer. Thus the ‘turn right’ command sent to the car.

The following **Figure 8, Figure 9 and Figure 10** demonstrate the other scenario where the car encountered the different conditions. They all followed same the procedure that the car process what it saw into semantic pointers, and the brain model made a decision based on visual input.

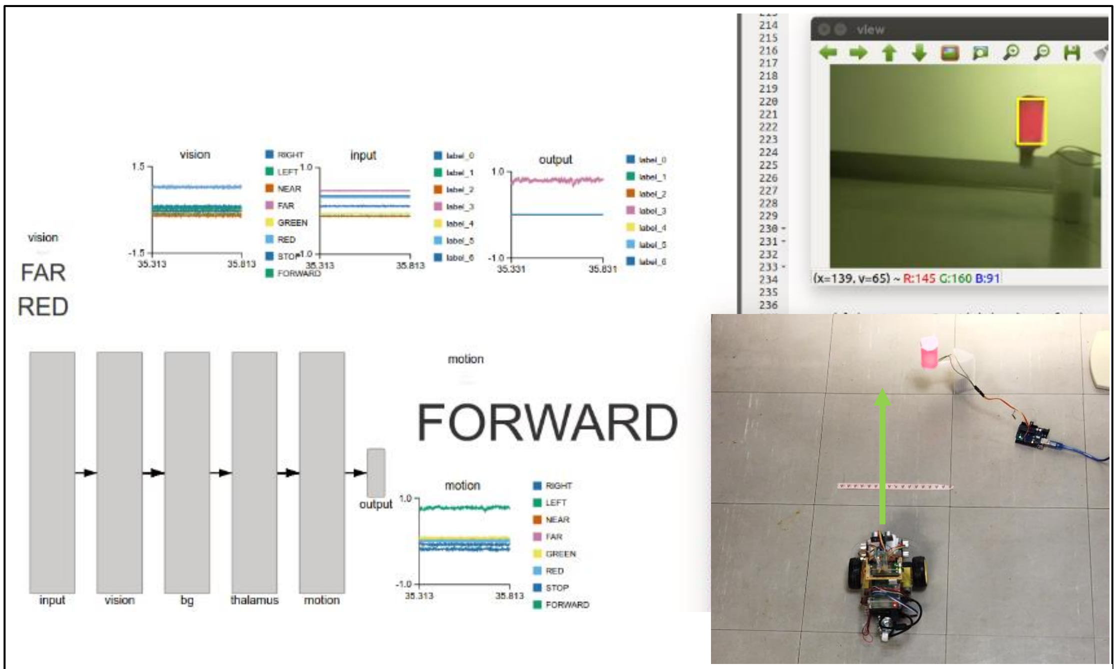


Figure 8, The inspection of the result at "FORWARD" decision. This figure shows that the car thinks the traffic light is far from itself, so it should keep going forward.

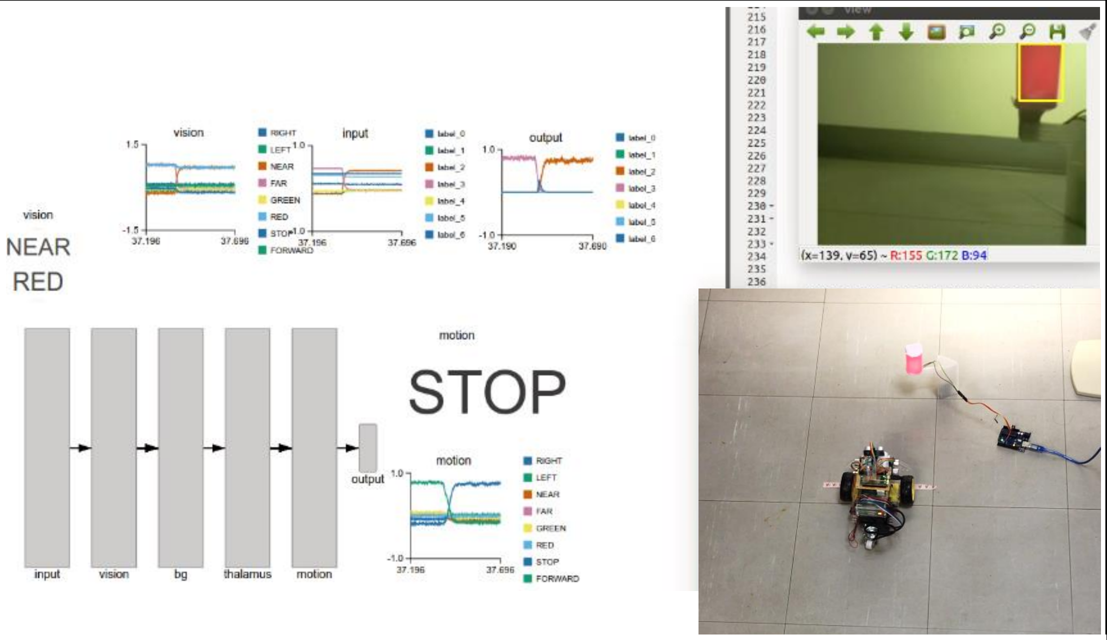


Figure 9, The inspection of the result at "STOP" decision when the car notice itself was around distance of the stop line and the traffic light is red.

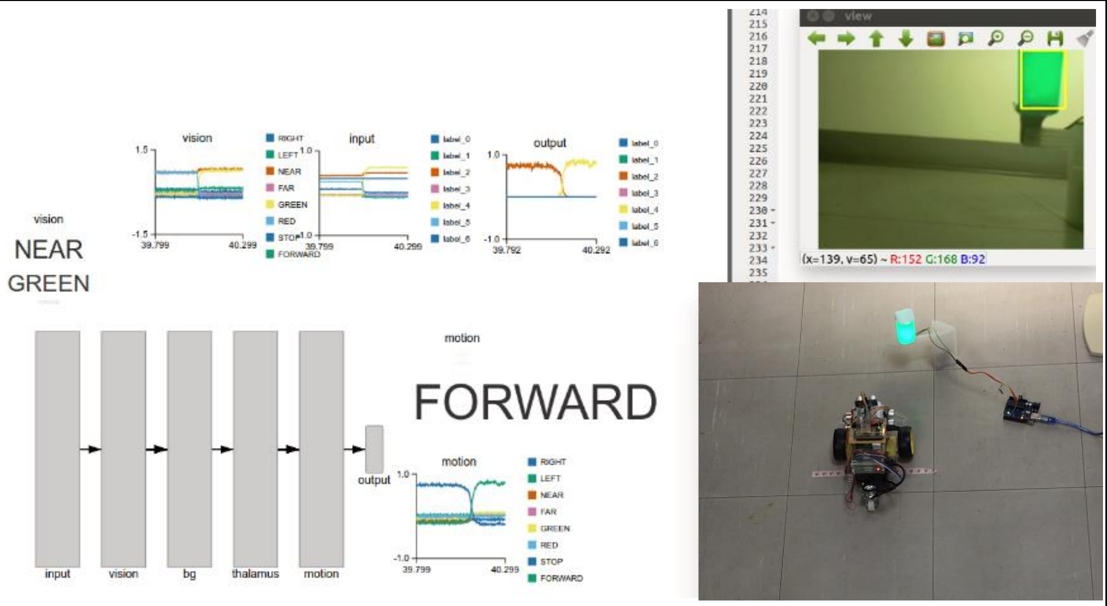


Figure 10, The inspection of the result at "FORWARD" decision when the traffic light turns green.

4. Discussion

4.1 Design of the SPA Model

The interesting part is designing the rules in BasalGanglia. We need to distribute the weight for every action. When we got the multiple input and set the same weight to every actions it will make wrong decision.

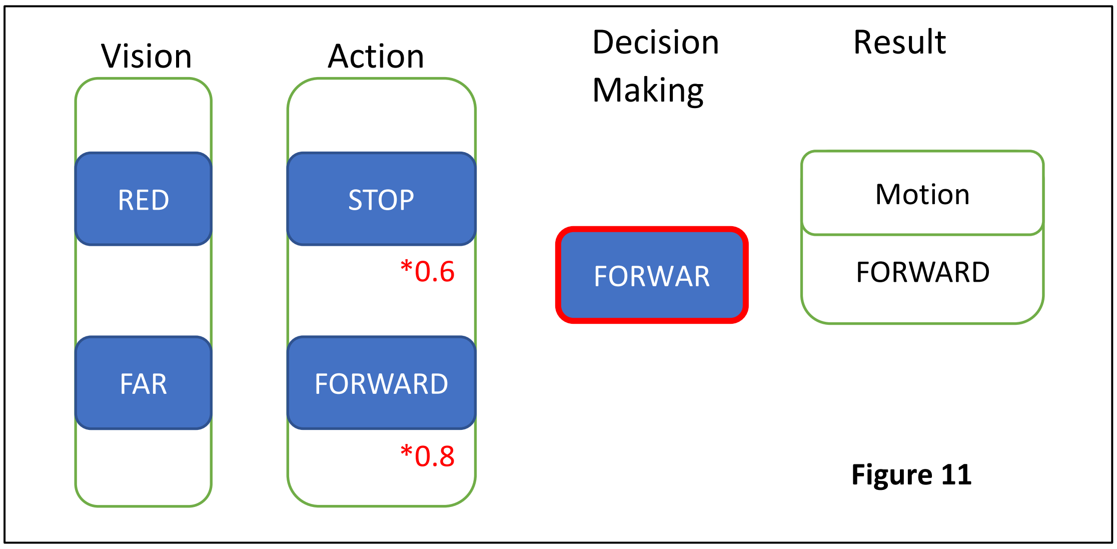


Figure 11

In **Figure 11**, we got the message from vision state, we have the traffic light(RED) and distance perception between the car and traffic light(FAR), so we have to decide that I get RED should be stop or still FAR from the traffic light should be forward. Obviously, the decision is the FAR one, so we need to set FAR the bigger weight than RED to deal with this scenario. If we didn't set the weight the decision will be unreasonable, because their action's weights are the same which will cause BasalGanglia be unable to make right decision.

4.2 Difficulties in Visual Processing

Since we use the size of traffic light in the image to estimate the distance, the car will get misunderstood in a very short distance. When the car gets too close to the traffic light, most part of the light will out of screen leaving only a small part in the sight. The car will think it still far from the light and keep moving forward. To prevent this from happen, we add one more constraint to check if the target is indeed far from the car. In general, farther object is closer to the horizon, the object which reaches the celling of the frame is probably nearer. So, we check the top-most coordinate of the target. If it exceeds 3 and its detected distance is more than 30cm, the car thinks the target is far from itself. Otherwise, the car thinks that it is close to the traffic light.

5. Conclusion

In vision part, the mission is to extract high dimensional (Color, Position and Distance of the traffic light) information from visual input (the image taken by camera). By using some image processing technique and powerful tool, it can be done efficiently. We first preprocessed the image to reduce the noisy effects. After that, the color space of the image has been converted which made it easy to handle in further process. Finally, the job can be done by setting some thresholds and doing linear transformation.

In the decision-making part, we want to let our self-driving car has some behavior like human in recognizing the traffic light. We use Nengo SPA model to reach this idea. In Nengo SPA, there is a state, Basal Ganglia, which is in charge of making the decision while it has multiple action to choose. After these experiments the result shows that during every different condition the self-driving car can make the right choice.

For the overall system, we demonstrated that the possibility of incorporating cognitive neuroscience into automobile and robotics system. In the implementation, we utilized socket connection, OpenCV, ROS, and Nengo in the system to deal with some engineering difficulties encountered. Finally, the Nengo empowered self-driving car successfully controlled itself based on the cognitive decision derived from sensing data, independently pass the trial.

6. Appendix

MESSAGE or SERVICE	*DATA STRUCTURE*
RaspiCarCamera.msg	string info uint64 time_stamp uint8[] camera_raw uint8[] camera_jpg
RaspiCarDistance.msg	string info uint64 time_stamp uint16[3] distance
RaspiCarServo.msg	string info uint64 time_stamp int8 servo_control
RaspiCarWheel.msg	string info uint64 time_stamp uint32[2] wheel_count
RaspiCarMotorControl.srv	string command string info --- string ack string info

Table 4, Definition files of ROS message and service used in this system.