Automated Robot (Car) using Artificial Intelligence

Satyam Mishra

International School - Vietnam National University

October 30, 2023

Abstract

This paper discusses about researching an automated robot car using artificial intelligence; training its neural network using AlexNet model, using YOLO (you only look once algorithm) for object detection phase and for practical deduction and judging component we have used Open Neural Network Exchange (ONNX) format. Our Robot Car model is agile and cost efficient. It detects objects efficiently in front of it and movement of it is smooth. It moves through sensors in motors which makes it different than other models in the world.
Automated Robot (Car) using Artificial Intelligence

Satyam Mishra  
Informatics and Computer Engineering,  
Vietnam National University – International School  
Hanoi, Vietnam  
satyam.entrprnr@gmail.com

Cao Sy Minh  
Informatics and Computer Engineering,  
Vietnam National University – International School  
Hanoi, Vietnam  
caosminh1@gmail.com

Trần Việt Long  
Management and Information System,  
Vietnam National University – International School  
Hanoi, Vietnam  
tranvietlong2k@gmail.com

Hoàng Thị Chúc  
Informatics and Computer Engineering,  
Vietnam National University – International School  
Hanoi, Vietnam  
hoangthichuc2001@gmail.com

Dr. Thanh Tung Nguyen  
Head, Department of Science and Technology  
Vietnam National University – International School  
Hanoi, Vietnam  
tungmt@isvnu.vn

Abstract—This paper discusses about researching an automated robot car using artificial intelligence; training its neural network using AlexNet model, using YOLO (you only look once algorithm) for object detection phase and for practical deduction and judging component we have used Open Neural Network Exchange (ONNX) format. Our Robot Car model is agile and cost efficient. It detects objects efficiently in front of it and movement of it is smooth. It moves through sensors in motors which makes it different than other models in the world.

Keywords— Deep learning, NVIDIA Jetson Nano, ISBot, AlexNet, YOLO (You Only Look Once), Open Neural Network Exchange (ONNX), Convolutional Neural Network (CNN), Stochastic Gradient Descent (SGD), Intersection over Union (IoU)

I. INTRODUCTION

In this research, we have developed self-driving robot car model based on deep learning to avoid collisions, injuries and provide output as path following and image recognition; provided with difficult and versatile situations around the robot, you can see in Figure 1. Elon Musk has devoted his efforts to this field and came up with an amazing model of the automated electric car. Google is also doing its best to enter the market with its automated car product model. All giant techs have devoted their time, money, and efforts to automation. What we did different is using Jetson Nano board, which is different from Tesla, much faster and efficient. Our device looks like an automobile-formed robot, ISBot, it has an NVIDIA artificial intelligence (AI) oriented board. Our ISBot can find numerous items and can avoid bumping into them. We did many experimental trials putting our ISBot in diverse conditions, we also made a city model to test all technical functionalities we added. We added Lego Worker as a man, trees, and some other stuff to check if it avoids it or not. Then in the same city model, we made our ISBot run to follow the road in the city model to check its functionality of path following. We also tested it with an area drawn on paper, bordered by black stripes as an avoiding factor, and we were successful. Our ISBot did stop on seeing the black stripe.

Each of the car-fashioned discern and the deep-learning based obstacles-avoidance function fluctuate from current techs and products with respect to their main objectives and structural capabilities.

II. PROPOSAL AND RESEARCH OBJECTIVES:

A. Proposals:

In the last few years, academic researchers, mechanical engineers, and software engineers have developed a wide range of automatic and mechanical technologies. Productivity, car costs, and other aspects have all improved as a result. With the merging of informatics, mechatronics, and electrical engineering, automation has continued to evolve. Practically, for the actual automation transportation field, being sustainable with avoidance of collision and detection of obstacles on the path is an important factor nowadays. We researched about Jetson Nano and ISBot and how we can utilize them to implement solutions for automated robot (cars) using Artificial intelligence. Jetson Nano and ISBot are two new, dependable infrastructures that allow for the execution of versatile deep learning-based experiments as well as the collection of useable, meaningful data on real-world aspects. We picked Jetson Nano board instead of raspberry pi and made ISBot for this research. ISBot is a two-wheeled car-like system that houses the Jetson Nano board as well as other components such as detectors and a camera. We created from scratch and managed to deploy a deep learning ISBot-based system using real-world (city simulation we created) visual input for everyday objects. We used a novel deep learning-based obstacle avoidance perspective for practical object detection training. Special data gleaned via the use of technologies and various sensors in this part could alter skill demand and achieve security levels. We built it from scratch and trained this AI-based jetson nano model to avoid collision and follow a given path by using deep neural network-based training. [1]
The following are some of the benefits of the proposed method:

1. Our robot car’s method is cheaper than others present in the market,
2. Our robot car system’s accuracy in object detection and avoidance is high,
3. Our robot car system can go on any kind of surface and any kind of situation provided minor changes,
4. Our robot car system can be modified to run on water as boat as well because object detection and avoidance will work on any surface,
5. Above point bring us to our future of this research, we will be building same automated robot for water and air taxi as well.

B. Research Objectives:

There are six stages to this research:
1. Setting up jetson nano, building whole ISBot from scratch,
2. Uploading/Updating OS/softwares and setting everything based on our research needs,
3. performing experiments on a real city model created by our team; you can see our city simulation model in later pages,
4. recording the performance of our robot and making it more efficient by experimenting,
5. make sure our robot ISBot follows a given path A to B to C to D and avoid any obstacle coming in between, and
6. when needed can be teleoperated means in case of emergency user can take control of driving.

III. RESEARCH METHODOLOGY:

TOOLS/TECHNOLOGY USED:

ISBot System:

We have used NVIDIA Jetson Nano board for our ISBot instead of raspberry pi in our experiment as you can also see in Figure 2. The system has never been used in an automatic automobile study before, however it has been used in certain educational methods studies. The Jetson Nano microcomputer board, it comes in two versions, the 2-GB and 4-GB developer kits and is one of the most popular boards to compete with the Raspberry Pi. After all, it comes with a full suite of ports and 40 GPIO pins on a relatively small form. The Jetson Nano runs on a quad-core ARM Cortex-A57 64-bit @ 1.42 GHz. The board has versatile features like we can utilize many deep learning algorithms, image classification, image recognition, voice recognition etc. The official operating system for the Jetson Nano is the Linux4Tegra, based on Ubuntu 18.04, with a Jetpack software development kit as its programming environment (SDK). The SDK includes mostly important needed tools like the operating system, libraries, interfaces, CUDA, and Robot Operating System (ROS). The CUDA Toolkit is a software development environment for creating high-performance GPU-accelerated applications. ROS is a set of software libraries that developers can use to create robot-based apps. It is open source and includes drivers, cutting-edge algorithms, and powerful developer tools. This is available via the included SD card image, which is designed to run NVIDIA hardware. [1] [2]

The Jetson Nano comes in a 4-GB LPDDR4 version or a 2-GB alternative. It supports HDMI 2.0 and DisplayPort (eDP 1.4). The 2-GB version only supports HDMI 2.0. Jetson Nano 4-GB has four USB 3.0 ports, one USB 2.0 Micro-B port, two MIPI CSI-2 DPHY lanes, an HDMI 2.0 port, and a DisplayPort. The 2-GB model has one USB 3.0 port, two USB 2.0 ports, one USB 2.0 Micro-B port, a MIPI CSI-2 D-PHY lane, and an HDMI 2.0 port. It also supports Gigabit Ethernet and M.2 Key E for Wi-Fi support. The NVIDIA Jetson Nano has a 128-core Maxwell GPU at 921 MHz. Compared side by side, the Jetson Nano has a much more capable GPU than the Raspberry Pi 4. This makes the Jetson Nano more suitable for AI and ML applications, which could be a specific advantage, and therefore much intended for our research end-use. [3] [4]

Deep learning-based Process:

We programmed the system code using Jupyter and Anconda platforms. For Image recognition we used AlexNet as neural network of deep learning with CLI argument as “alexnet” and NetworkType enum as “ALEXNET”. AlexNet is a Convolutional Neural Network which includes 5 convolution layers and 3 fully connected layers. Convolution layers (also known as filters) extract useful information from images. In any convolution layer there are usually many kernels of the same size. For example, the first convolution layer of AlexNet contains 96 kernels of size 11x11x3. Usually, the width and height of a kernel are equal, and depth is usually equal to the number of color channels as you can see in Figure 3.

Convolutional 1 and convolution 2 are connected via an Overlapping Max Pooling in between. Similarly, between convolution 2 and convolution 3.
Convolutional 3, convolution 4, convolution 5 connect directly with each other, without intermediaries. Convolutional 5 connects fully connected layer 1 through an Overlapping Max pooling, followed by another fully connected layer. And finally, a softmax classifier with 1000 layers of labels (you can see the image of the AlexNet network architecture above for a more general view).

ReLU nonlinearity is used after all convolutions and fully connected layers. In the past, the ReLU nonlinearity of convolution layers 1 and 2 was usually followed by a local normalization step followed by pooling. However, later studies found that using normalization was not really helpful. So we won’t go into detail on that. [8]

We started the execution of our neural network training with 30 epochs; set batch size = 16, shuffle = True, Optimizer = SGD (Stochastic Gradient Descent), lr = 0.001, and momentum = 0.9. SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features. [6] The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

Mathematically, the update rule of SGD is: [7]

$$\theta = \theta - \eta \nabla_\theta J(\theta; x; y)$$

For the activation function of neural network, we used sigmoid function. Then we loaded trained model on the system. We used PyTorch, torchvision, OpenCV2, and numpy in the beginning. Then after when we were ready to take images from camera as image data to train on our neural network, we used YOLO algorithm for detection of objects. You Only Look Once (YOLO) is an algorithm that uses neural networks to provide real-time object detection. This algorithm is popular because of its speed and accuracy. It has been used in various applications to detect traffic signals, people, parking meters and animals. [8]

![Figure 4 YOLO Architecture: Real-time Object Detection](image)

We applied Tiny YOLO version 2; this model is a real-time neural network for object detection that detects 20 different classes. It is made up of 9 convolutional layers and 6 max-pooling layers. [9] We utilized a Five Anchor Box-based method, we input image of shape (5 * 416 * 416). With five assumed anchor boxes, the output is (13*13*25*5). Because the output is a (13*13*25*5) tensor where 13*13 is the number of grid cells that the images get divided into. Each grid cell corresponds to 125 channels, made up of the 5 bounding boxes predicted by the grid cell and the 25 data elements that describe each bounding box (5*25=125). [10]

To put everything in summary, what a convolution layer calculates: if x is the pixels in the input image and w is the weights for the layer, then the convolution basically computes the following for each output pixel:

![Figure 5](image)

<table>
<thead>
<tr>
<th>Layer</th>
<th>kernel stride output shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(448, 448, 3)</td>
</tr>
<tr>
<td>Convolution 1</td>
<td>3x3 1 - (448, 448, 16)</td>
</tr>
<tr>
<td>MaxPooling 1</td>
<td>2x2 2 - (224, 224, 16)</td>
</tr>
<tr>
<td>Convolution 2</td>
<td>3x3 1 - (224, 224, 32)</td>
</tr>
<tr>
<td>MaxPooling 2</td>
<td>2x2 2 - (112, 112, 32)</td>
</tr>
<tr>
<td>Convolution 3</td>
<td>3x3 1 - (112, 112, 64)</td>
</tr>
<tr>
<td>MaxPooling 3</td>
<td>2x2 2 - (56, 56, 64)</td>
</tr>
<tr>
<td>Convolution 4</td>
<td>3x3 1 - (56, 56, 128)</td>
</tr>
<tr>
<td>MaxPooling 4</td>
<td>2x2 2 - (28, 28, 128)</td>
</tr>
<tr>
<td>Convolution 5</td>
<td>3x3 1 - (28, 28, 256)</td>
</tr>
<tr>
<td>MaxPooling 5</td>
<td>2x2 2 - (14, 14, 256)</td>
</tr>
<tr>
<td>Convolution 6</td>
<td>3x3 1 - (14, 14, 512)</td>
</tr>
<tr>
<td>MaxPooling 6</td>
<td>2x2 2 - (7, 7, 512)</td>
</tr>
<tr>
<td>Convolution 7</td>
<td>3x3 1 - (7, 7, 1024)</td>
</tr>
<tr>
<td>MaxPooling 7</td>
<td>2x2 2 - (3, 3, 1024)</td>
</tr>
<tr>
<td>Convolution 8</td>
<td>3x3 1 - (3, 3, 2048)</td>
</tr>
<tr>
<td>MaxPooling 8</td>
<td>2x2 2 - (1, 1, 2048)</td>
</tr>
<tr>
<td>Convolution 9</td>
<td>3x3 1 - (1, 1, 4096)</td>
</tr>
</tbody>
</table>

$$out[j] = x[i]*w[0] + x[i+1]*w[1] + x[i+2]*w[2] + ... + x[i+k]*w[k] + b$$

This is a dot product of the input pixels with the weight of the convolution kernel, plus a bias value b.

And here’s the calculation performed by the batch normalization to the output of that convolution:

$$bn[j] = \frac{gamma * (out[j] - mean)}{sqrt(variance)} + beta$$

It subtracts the mean from the output pixel, divides by the variance, multiplies by a scaling factor gamma, and adds the offset beta. These four parameters — mean, variance, gamma, and beta — are what the batch normalization layer learns as the network is trained.

To get rid of the batch normalization, we can shuffle these two equations around a bit to compute new weights and bias terms for the convolution layer:

$$w\_new = \frac{gamma * w}{sqrt(variance)}$$

$$b\_new = \frac{gamma * (b - mean)}{sqrt(variance)} + beta$$

Performing a convolution with these new weights and bias terms on input x will give the same result as the original convolution plus batch normalization. Now we can remove this batch normalization layer and just use the convolutional layer, but with these adjusted weights and bias terms w\_new and b\_new. We repeat this procedure for all the convolutional layers in the network. [11]

For the practical deduction and judging component, we used the Open Neural Network Exchange (ONNX) format, which is an open ecosystem that empowers AI developers to choose the right tools as their project evolves. ONNX provides an open-source format for AI models, both deep learning and traditional ML. It defines
an extensible computation graph model, as well as definitions of built-in operators and standard data types. Currently we focus on the capabilities needed for inferencing (scoring). ONNX is widely supported and can be found in many frameworks, tools, and hardware. Enabling interoperability between different frameworks and streamlining the path from research to production helps increase the speed of innovation in the AI community. [13]

In our research study, we have not used pre-learned pth format file, but we create an original file. Which we obtain after training our neural network and obtaining the best path (model) file i.e., pth-file. For deep learning, we have used two datasets for taking pictures from our ISBot’s camera:

1. first data set where our robot can move freely, and
2. second dataset where our robot should stop to avoid collision or take a turn from there to reach from path A to B to C to D and vice versa.

Our ISBot will execute deep learning processes with the above-mentioned picture datasets which it will get from camera installed on our ISBot then it will create a .pth file, meanwhile we also give versatile picture datasets for our perfect path outcome to neural network. In the ISBot kit’s SD card, Python, PyTorch, and CUDA based notebooks for obtaining picture data are pre-loaded for general users. We utilized them for our comfort with keeping up the modern technology trends. For the graphic computing, we used a GPU instead of a CPU.

IV. EXPERIMENTAL METHOD:

Steps:
Preparation for neural network training/Training:

Basic motion of ISBot: To train the ISBot’s basic movements, we started by importing the Robot class from the ISBot package in jupyter notebook. After that, we added many functions to make it turn left, right, move forward, move backward, and stop it. After that we linked its motor to traitslets to make it much easier to use. Jupyter Notebooks allow us to create graphical widgets that make use of traitslets. We can now attach our motors to widgets and control them from the browser, or simply visualize the value. From there we got a graphical button user interface on the browser to turn it left, right or move it back and forth or stop it.

Teleoperation: We'll use a gamepad controller connected to our web browser machine to control the ISBot remotely. The first thing we'll do is make an instance of the Controller widget, which will be used to control our robot. The Controller widget accepts an index argument that provides the controller's number. If you have many controllers connected, or if certain gamepads appear to be numerous controllers, this is handy. The motor control is the first and most basic control we wish to add. Using the dlink function, we'll connect it to the left and right vertical axes. Unlike the link function, the dlink function allows us to connect the source and target with a transform. We'll use a simple lambda function to negate the value because the controller axes are inverted from what we think is sensible for motor control. Then we will add an image widget using

```
to show our live camera feed. We'll keep the height and width to 224 pixels, so it doesn't take up too much room. We'd like to be able to save some photographs from our robot right now. Allow the right bumper (index 5) to save a snapshot of the current live image. Using the uuid python module, we'll save the images in the snapshots/ directory with a name that's guaranteed to be unique. We choose the uuid1 identification since it includes the date and MAC address, which we may need in the future. Before closing this notebook and shutting down the Python kernel for the notebook, we want to properly close the camera connection so that we can use the camera in another notebook.

Collision Avoidance: We developed the ISBot-based system to move and avoid various obstacles in our own build city model. Then, we executed practical experiments to test the system's performance, especially in relation to obstacle avoidance. Here, the system automatically moves through the nondirected path between positions A and B (via a path from A to B, or from B to A), assuming (simulating) the situation where the system is moving on its own and avoiding the conflicts on its path. ISBot has previously input a command to go to the right automatically if it is about to crash into walls or other non-directed items (i.e., one of the #1-5 obstacles). The obstacles (target objects to avoid collisions with); the obstacles are #1 Tree, #2 Black line, #3 Wall, #4 Tower, and #5 Building. ISBot is 190 mm long, 125 mm wide, and 105 mm high, excluding the two antennas to connect to Wi-Fi networks. It also has a rubber wheel that is 60 mm in diameter. The city model we build has roads, trees and a building to test and recognize its collision avoidance, image recognition and path follow algorithms.

![ISBot being trained for black stripe border](image=widgets.Image(format='jpeg', width=224, height=224, fps =5) display(image))

We have taken 400 picture input data in total from ISBot's camera of the five obstacles 80 times (40 times each for "no problem data" and "obstacle data"). We changed the camera angle at intervals of 4.5 degrees (360 ÷ 80 = 4.5), from starting position A to position B and same goes for C to D and vice versa. We split the image input dataset in two parts before training it for learning, first will be training and then testing. Finally, testing dataset will be used to find the trained model’s performance.

Figure 6 : ISBot being trained for black stripe border
Figure 7 Collision Avoidance Training Process

The Figure 7 shows the folder containing two picture datasets folder; blocked means ISBot should stop/take a turn from there, free means it can remain in motion.

Finally, we start the experiment session, by starting the power button, collecting data, setting parameter when needed for obtaining our research results, fixing the angle of camera, amount of light for picture capturing, doing multiple test trials and finally training collected data over network. When best path file obtained, we run our ISBot to see it was working perfectly and avoiding the objects put in its path. We also experimented sending our robot from Path A to B to C to D to A in the city model we designed as you can see in Figure 8.

Figure 8: Testbed environment for automated robot car

Path Following: In final trials ISBot moves from Path A to B, Path B to C, Path C to D and Path D to A successfully avoiding any obstacle being put in their path as you can see in Figure 9. When it recognizes any obstacle, it stops/turns instantly from that obstacle. We also noted experimental testing data in Table I for Success (From Different Position), Tests, Block Image Data (where robot will stop/turn), Free Image Data (where robot will be in motion) and Experiment Time (minutes).

Figure 10: A window concerning block and free picture-data collection:

<table>
<thead>
<tr>
<th>Success (From Different Position)</th>
<th>Tests</th>
<th>Block Image Data</th>
<th>Free Image Data</th>
<th>Experiment Time (minutes)</th>
<th>Collision Avoidance Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>From path A to B</td>
<td>40</td>
<td>200 images</td>
<td>200 images</td>
<td>120 minutes</td>
<td>Collision Avoided = 39/40 = 97.5%</td>
</tr>
<tr>
<td>From path B to C</td>
<td>40</td>
<td>200 images</td>
<td>200 images</td>
<td>120 minutes</td>
<td>Collision Avoided = 39/40 = 97.5%</td>
</tr>
<tr>
<td>From path C to D</td>
<td>40</td>
<td>200 images</td>
<td>200 images</td>
<td>120 minutes</td>
<td>Collision Avoided = 40/40 = 100%</td>
</tr>
<tr>
<td>From path D to A</td>
<td>40</td>
<td>200 images</td>
<td>200 images</td>
<td>120 minutes</td>
<td>Collision Avoided = 40/40 = 100%</td>
</tr>
<tr>
<td>Total</td>
<td>160</td>
<td>800 images</td>
<td>800 images</td>
<td>480 minutes</td>
<td>Average = (97.5+97.5+100+100) / 4 = 98.75%</td>
</tr>
</tbody>
</table>

V. RESULT EVALUATION:

We experimented and succeeded with the objectives of proposed research with accuracy of 98.75% as you can see the results in Table 1. Our ISBot based on NVIDIA Jetson Nano recognizes the obstacles and avoids them successfully. 1.25% error can be because of technical error or manmade mistakes. When needed it can be teleoperated as well or say can be operated manually in case of emergency. This is the first time we have done any research on some topic like Automated Robot (car) using Artificial Intelligence.
If 1.25% error occurs then it meant that the ISBot detected obstacle too early to detect and avoid it, it detects object in front of it faster so couldn’t avoid it, it doesn’t have enough light to get input and some of them might be some error while training the image input data in the neural network.

Our trained ISBot model was trained perfectly under given conditions of the city model as you can also see in figure 11. It stopped on the obstacles mentioned and even if someone put his/her hand in front of it. It perfectly followed from position A to B to C to D to A and recognized a tree or building or any obstacle being put into its path, making it stop/turn to avoid collision. This makes it a recognizable and sustainable automated robot car system. We are now looking forward to adding more features into upcoming time like text to speech and speech to action feature into it. [14]

**Following are some benefits of our model:**
- During Covid time, safe traveling, and transfer of covid patients
- 90% reduction in Traffic Deaths
- 60% drop in harmful emissions
- 10% improve in fuel economy
- 40% reduction in travel time

**VI. Conclusion and future work:**

We experimented with an actual city model and collision avoidance model with many physical objects and a dark black line (stripe) to create and verify basic testing phases of the ISBot and deep learning-based, car-shaped system and methodology. We gave important and useful input data for deep learning and neural network training, as well as output-based suggestions for physical object obstacles we made inside our city model. We discussed collision-avoidance functions and characteristics in the context of movement, as well as road following (path following). As this experiment was done in a small city model that we made, we are planning to experiment similar AI integrated system in real-world and add more useful functionality (e.g., text to speech, speech to action, automatic siren, and lightning functions). Our final research conclusion is that, we have largely achieved automated robot car training, particularly machine learning-based technologies in a city simulated environment. We hope industrial engineers and workers can further add more to our robot and make it even more task-oriented and secure.

**Future scope and research work:**
- To transform our automated Car model into an automated flying taxi.
- Produce an air traffic control plan for it.
- Add more features.

**References**
1. Jetson Nano vs Raspberry Pi 4: https://all3dp.com/2/raspberry-pi-vs-jetson-nano-differences/
   and https://en.nancedigital.com/article/nvidia_jetson_nano_vs_raspberry_pi
10. Tiny YOLOv2: https://github.com/omnis/models/tree/master/vision/object_detection_segmentation/tiny-yolov2
11. Tiny YOLOv2: https://gallery.azure.ai/Model/TinyYOLOv2
13. ONNX: https://github.com/onnx/onnx
14. Deep Learning-Based Self-Driving Car: http://dx.doi.org/10.24018/ejfood.2020.2.3.45

![Figure 11: Our ISBot being trained for road (path) following](image-url)

![Figure 12: Flowchart of the whole research in general](image-url)