Self driven Car

Ecole Nationale des Sciences Appliquée de Kénitra ,

Our project focuses on building an AI-driven robot using the Jetson Nano platform for autonomous navigation. By integrating advanced road following and collision avoidance algorithms, our JetBot system can navigate environments with precision and safety. Leveraging deep learning and computer vision techniques, the robot can detect road markings for navigation and dynamically adjust its path to avoid obstacles. This showcases the Jetson Nano's computational power in enabling real-time decision-making for intelligent robotics applications.

Keywords –jetson nano, Restnet18, I2C, TensorRt, jetbot, , AI model, AI, Image processing.

I. INTRODUCTION

JetBot is an open-source, educational AI robotics platform that combines the NVIDIA Jetson Nano with a set of hardware components to enable a wide range of projects, from autonomous navigation to object recognition.

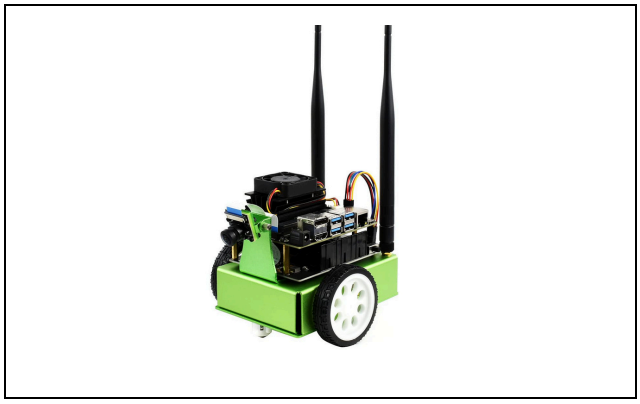

Fig. 1. Jetbot Ai Robot

2. Hardware and Software Configuration

The JetBot project integrates a sophisticated blend of hardware and software components to achieve its autonomous navigation capabilities. On the hardware side, JetBot utilizes the powerful NVIDIA Jetson Nano, a compact yet potent AI computing platform capable of processing complex algorithms in real-time. Complementing the Jetson Nano is a myriad of sensors including cameras, motor controllers, and collision detection sensors, all meticulously integrated to provide comprehensive environmental perception. Meanwhile, on the software front, JetBot leverages state-of-the-art deep learning frameworks such as PyTorch to implement neural networks for tasks like road following and collision avoidance. Additionally, custom software modules handle sensor data processing, motor control, and high-level decision-making, orchestrating the seamless operation of the robot in diverse scenarios. Together, these hardware and software components form the backbone of JetBot, empowering it to navigate autonomously with precision and intelligence.

A. Hardware Configuration

The hardware architecture of JetBot comprises a carefully curated selection of components designed to enable robust and efficient autonomous navigation. At its core lies the NVIDIA Jetson Nano, a high-performance AI computing platform that serves as the brain of the robot. This compact yet powerful device provides the computational horsepower necessary to process sensory data and execute complex algorithms in real-time. Surrounding the Jetson Nano are an array of sensors, actuators, and motor controllers, each playing a crucial role in the robot's perception and locomotion.

1- NVIDIA Jetson Nano (B01) : The central processing unit of JetBot, the Jetson Nano, is equipped with a quad-core ARM Cortex-A57 CPU and an NVIDIA Maxwell GPU with 128 CUDA cores. This combination of CPU and GPU allows the Jetson Nano to handle intensive AI workloads efficiently, making it an ideal choice for robotics applications.

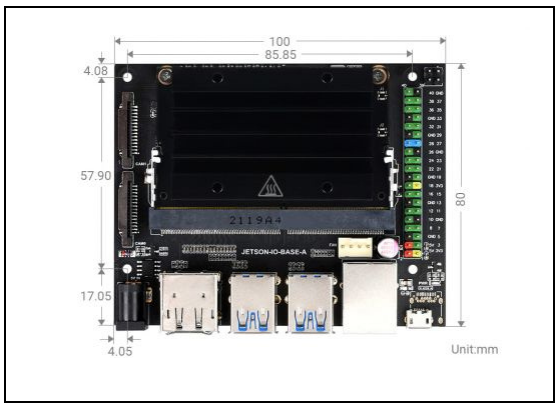

Fig. 2. Jetson Nano

2- Camera Module (IMX219-160) : JetBot typically incorporates a camera module, such as the Raspberry Pi Camera, to capture visual data from its environment. This camera serves as the primary sensor for tasks like road following and object detection, providing the robot with crucial information about its surroundings.

Fig. 3. IMX219-160 Camera.

3-**Motor Controllers:** To facilitate movement, JetBot utilizes motor controllers to regulate the speed and direction of its motors. These controllers interface with the Jetson Nano and translate motor commands generated by the software into physical motion.

4-**PiOLED display:** The OLED display on the JetBot is a compact yet powerful visual interface, offering advantages like high contrast ratios and low power consumption. Despite its small size, it provides clear visibility in various lighting conditions, thanks to its impressive brightness levels. With its ability to produce true blacks and vibrant colors, it ensures excellent readability of displayed content. The display serves as a dynamic information panel, offering real-time feedback on motor speeds, steering angles, battery voltage, and system diagnostics. Its compatibility with the JetBot Python library enables seamless integration and customization, making it an indispensable component for monitoring and controlling the JetBot's operation.

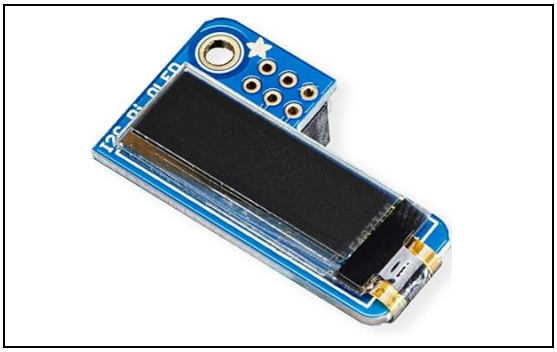

Fig. 4. PiOLED.

B. software Configuration

The software configuration of the JetBot involves setting up the operating system, installing necessary libraries, and configuring the neural network models for road following and collision avoidance.

1.Operating System: The JetBot typically runs on the Jetson Nano Developer Kit, utilizing NVIDIA's JetPack software development kit (SDK). JetPack includes the Linux-based Ubuntu operating system optimized for the Jetson platform.

2.**Python Libraries:** Various Python libraries are installed to enable JetBot functionality. These include:

- PyTorch: A deep learning framework used for training and deploying neural networks.
- torchvision: A PyTorch library providing tools and utilities for computer vision tasks.
- torch2trt: A library for converting PyTorch models to TensorRT (NVIDIA's deep learning inference optimizer) format for faster inference on the GPU.
- traitlets: A library for creating and configuring traits, enabling interactive control of JetBot components.
- ipywidgets: Interactive HTML widgets for Jupyter notebooks, facilitating dynamic user interfaces.
- torchvision.transforms: Transformation functions for preprocessing images before feeding them into the neural networks.
- cv2: OpenCV library for computer vision tasks such as image processing and manipulation.

3.Neural Network Models: Pre-trained neural network models for road following and collision avoidance are loaded onto the JetBot. These models are typically trained using PyTorch and converted to TensorRT format for optimized inference. The road following model predicts the direction in which the JetBot should steer to stay on the road, while the collision avoidance model detects obstacles and triggers evasive actions.

II.Road Following:

The road following capability of JetBot is a fundamental aspect of its autonomous navigation system. Leveraging a neural network trained on a combination of labeled real-world data and synthetic images, JetBot is equipped to autonomously follow paths, roads, or any designated trajectory. This functionality is vital for applications such as autonomous delivery, surveillance, or exploration, where consistent and accurate navigation is crucial.
JetBot's road following system utilizes a camera mounted on its chassis to capture real-time images of the environment. These images are processed through a deep neural network model, which has been trained to recognize and interpret road features, including lane markings, edges, and obstacles. The model generates steering commands based on its interpretation of the road scene, enabling JetBot to adjust its trajectory in real-time to stay aligned with the desired path.

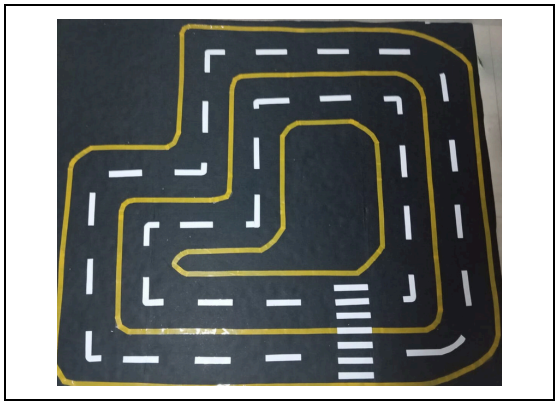

Fig. 5. Path.

The road following capability is achieved through a combination of hardware and software components, including the camera module, NVIDIA Jetson Nano, and custom-built neural network models. This integration allows JetBot to navigate complex environments with varying lighting conditions, road types, and obstacles while maintaining a high level of accuracy and reliability.
In the following sections, we will delve into the technical details of JetBot's road following system, including the neural network architecture, training process, and real-world performance evaluation.

JetBot's road following system operates through three primary processes: data collection, training, and deployment/live demo:

1.**Data collection** :

In this part of the activity, we'll collect data for JetBot's Path Following by capturing images through a Jupyter Notebook. we will use a widget to select a point on each image that identifies the part of the path that the robot should drive toward.

Note: The image is immediately saved with the x, y pixel coordinate position of the green circle as part of its file name

Fig. 6. data collection process.

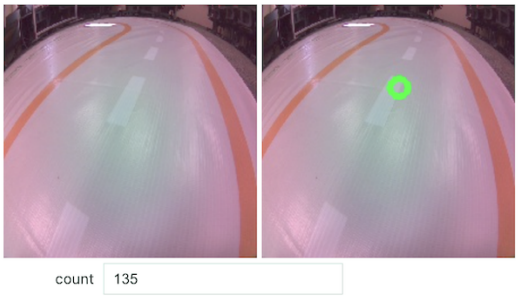

Fig. 7. Path Following Data Collection Camera Widget

As we collect data:
we Capture the Path at Multiple Distances

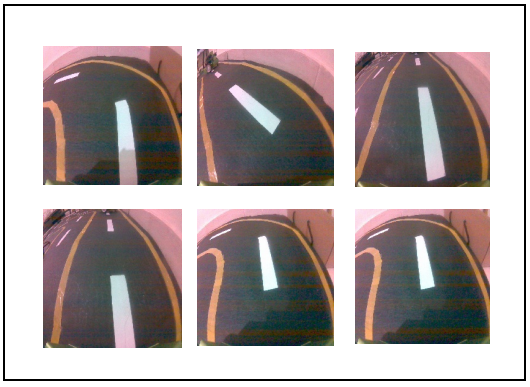

Fig. 8. Path Following Dataset Sample

2.**Training** :

the training process encompasses the development and optimization of the neural network model using the collected data. Through techniques such as data augmentation, transfer learning, and hyperparameter tuning, the model learns to accurately interpret road features and make informed navigation decisions.

TIn machine learning, regression is used to predict continuous values, unlike classification which predicts discrete categories. Predicting continuous values allows the robot to determine how much it needs to turn to stay on the desired path.

Regression can be represented by a line (or curve) that tries to fit through the points in a graph. The points represent the training data and the line represents the predicted continuous values. The goal of the regression model is to minimize the distance between the line and the points.

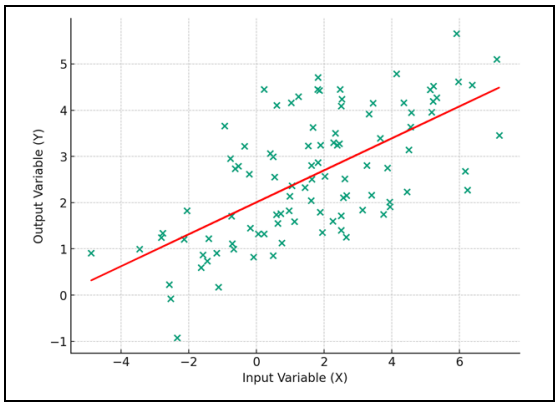

Fig. 9.Regression as a Graph

.

n the JetBot Path Following activity, regression is used to enable the robot to accurately determine the next position it should move toward along the path. The regression model processes your dataset of images with labeled path points to learn how to convert them into coordinates (x, y pixel positions as seen by the camera). As the JetBot moves, it uses the calculated coordinates from the trained model to smoothly navigate along the path and adjust its direction.

3.**live demo**:

Finally, the deployment/live demo process focuses on integrating the trained model into JetBot's software framework for real-time operation. This involves optimizing the model for inference on Jetson Nano's GPU, configuring input/output pipelines for camera data, and implementing control algorithms for steering and speed regulation. Once deployed, JetBot can autonomously navigate its environment, demonstrating its road following capabilities in real-world scenarios.

4.**Model Optimization** :

When we ran path following previously, our robot followed the path! But, very slowly and wobbly. In this part of the activity, We'll make the robot be able to follow the path much more smoothly!

n this part of the activity, we will optimize the JetBot's Path Following model using TensorRT. This involves converting the trained ResNet18 model to a TensorRT version for enhanced efficiency and speed. We'll use one Jupyter Notebook for the conversion process and then test the optimized model's performance through another, observing the performance improvements achieved.

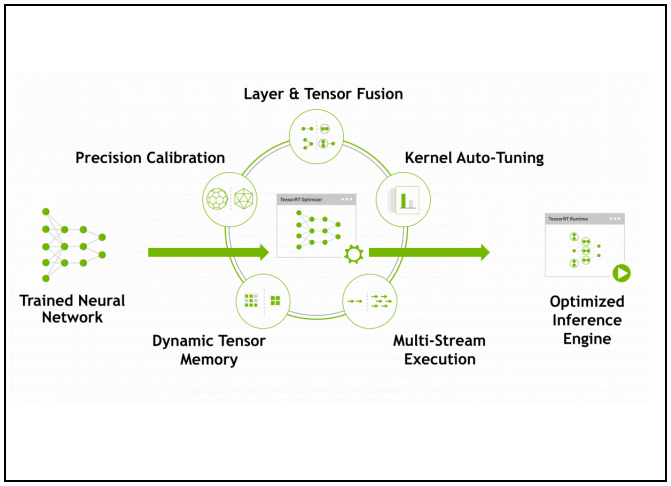

Fig. 10.TensorRT Optimizations

III.Collision avoidance :

Collision avoidance is a critical feature of JetBot's autonomous navigation system, ensuring safe operation in dynamic environments with obstacles and potential hazards. By leveraging advanced computer vision techniques and deep learning algorithms, JetBot can detect obstacles in its path and make real-time decisions to avoid collisions.
This technology is essential in autonomous navigation projects like the Cargo Unmanned Ground Vehicle (CUGV) and in even smaller devices like the JetBot
The collision avoidance functionality of JetBot involves three main components: data acquisition, model training, and integration into the navigation system.

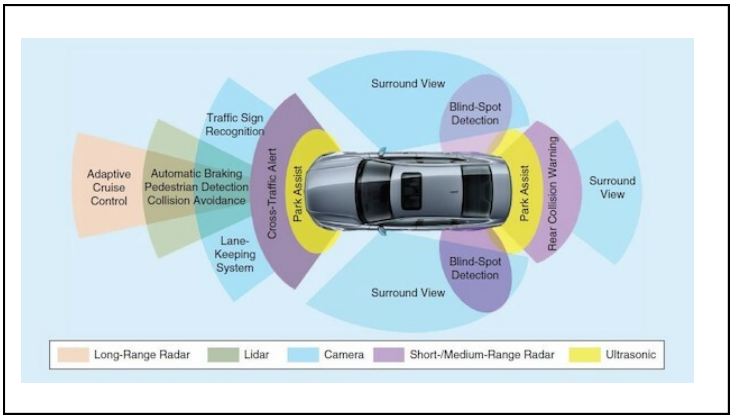

Fig. 11. Collision Avoidance Diagram

1.**Data collection** :

The first step in developing a robust collision avoidance system is to collect a diverse dataset of images representing various obstacle scenarios. This dataset includes images captured from different perspectives, under different lighting conditions, and with various types of obstacles, such as pedestrians, vehicles, and environmental obstructions. By collecting a comprehensive dataset, JetBot can learn to recognize and classify obstacles accurately during training.

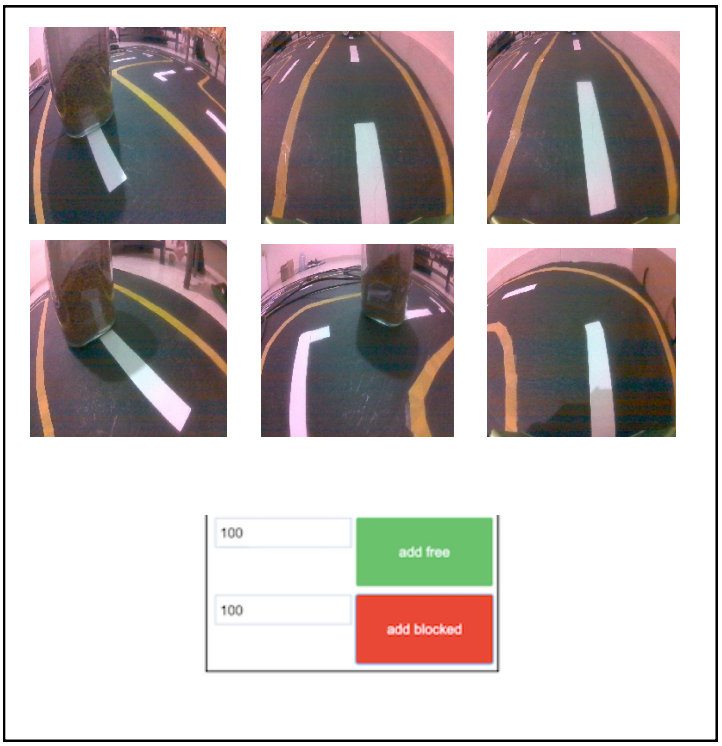

Fig. 11. Data Collection Widget and Sample Dataset

2.**Model Training**:

Once the dataset is collected, the next step is to train a deep learning model to detect obstacles in real-time. This involves using convolutional neural networks (CNNs) or similar architectures to process input images and predict the presence and location of obstacles. During training, the model learns to extract relevant features from the input images and make predictions based on learned patterns. Techniques such as transfer learning and data augmentation are often used to improve the model's performance and generalization ability.

First let's see Classification in machine learning , it is a process where a model is trained to categorize data into **distinct, predefined classes**. It's like training an algorithm to sort items into different groups based on their features.

Classification can be visualized as a line on a graph that separates points into different categories. The points on either side of the line belong to different classes.

n this part of the activity, we will optimize the JetBot's Path Following model using TensorRT. This involves converting the trained ResNet18 model to a TensorRT version for enhanced efficiency and speed. We'll use one Jupyter Notebook for the conversion process and then test the optimized model's performance through another, observing the performance improvements achieved.

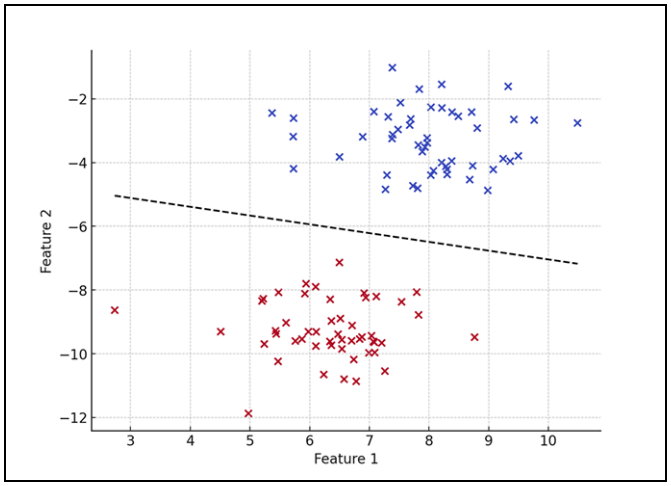

Fig. 12. Classification as a Graph

In the JetBot Collision Avoidance activity, classification is used to train the robot to identify 'blocked' and 'free' paths. The model uses your dataset to identify patterns or features that are indicative of each class. As the robot navigates, it will use its trained model to decide whether the path ahead is either blocked or free, and then turn or continue forward, respectively.

to give more context we will explain briefly what is Neural Networks and Transfer Learning
Neural networks are computational models inspired by the human brain, designed to recognize patterns and make predictions from data. They consist of interconnected nodes, called neurons, and have three main components: the **input layer**, **hidden layers**, and the **output layer**. The input layer receives raw data, hidden layers perform complex computations, and the output layer produces the final prediction or output.

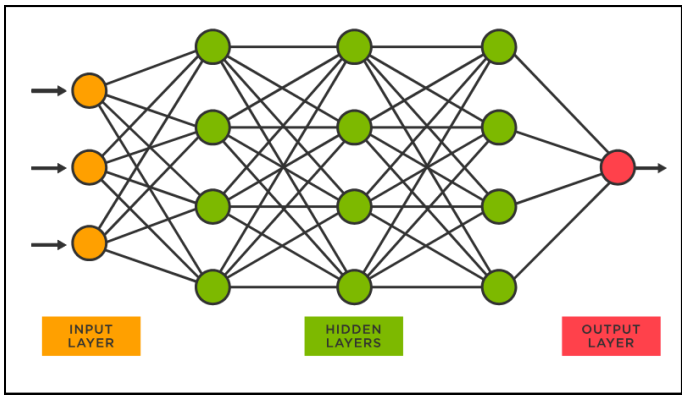

Fig. 12.Neural Network Structure

Transfer learning is a powerful machine learning technique where a model developed for one task is repurposed in a second, related task. In the Collision Avoidance activity, we are repurposing the ResNet-18 model (which was originally trained on a dataset with 1000 class labels!) for our dataset which contains just two class labels ('blocked' and 'free') by replacing its final layer. Transfer learning allows us to save time and resources by leveraging a pre-trained model, rather than training a new model from scratch.

4.**Model Optimization** :

When you ran collision avoidance previouslye, our robot avoided collisions! But, very slowly. In this part , we'll make the robot be able to avoid collisions much faster!

we will optimize the JetBot's collision avoidance model using TensorRT. This involves converting the trained ResNet18 model to a TensorRT version for enhanced efficiency and speed. You'll use one Jupyter Notebook for the conversion process and then test the optimized model's performance through another, observing the performance improvements achieved.

IIII.Combining Collission avoidance with Road following :

This project focuses on combining both optimized regression and classification models into one notebook to enable the Jetbot to follow a specific path on the track and at the same time also be able to avoid collisions with obstacles that come on it's way in real-time by bringing the Jetbot into a complete halt.

Note: For Collision Avoidance, the Blocked class should include images of obstacles such as vehicles, people, stop signs etc. captured on the track, meanwhile the Free class should include background images of the empty track where the Jetbot should be free to move around in.

Fig. 13.Road Following + Collision Avoidance

IIIII.Reference :

[1] https://www.cs2n.org/u/mp/badge_pages/3220

[2]https://github.com/abuelgasimsaadeldin/Jetbot-Road-Following-and-Collision-Avoidance

[3] https://github.com/NVIDIA-AI-IOT/torch2trt

[4]https://docs.nvidia.com/deeplearning/tensorrt/release-notes/index.html#rel_7-1-3

[5] https://jetbot.org/master/