LRIMa city: a Fog-Computing-Based Smart City with Integrated Self-Driving Cars

Jihene Rezgui, Enric Soldevila, Abderrazak Mokraoui
Laboratoire **R**echerche **I**nformatique **M**aisonneuve (LRIMa)

Montreal, Canada
jrezgui@cmaisonneuve.qc.ca

***Abstract* – The LRIMa city is a *continuous, expandable* and *polyvalent* project focused on IoT and IA solutions for research and learning. It serves as a testbed for exploring IoT architectures, including cloud centralized and fog on-device computation. Additionally, the city enables the development of AI models for autonomous vehicle navigation. Our smart city implementation encompasses various embedded systems such as smart motorized vehicles, a speed radar, smart parking systems, adaptive streetlights, and a remotely controlled bridge. To facilitate autonomous driving, we have created a Convolutional Neural Network (CNN) based on NVIDIA's model for predicting steering angles from input images. Moreover, we have developed a YOLOv8-based traffic light detection model and a cascade classifier for stop sign detection. For other components, we have employed diverse AI solutions, ranging from license plate detection and identification to hand gesture recognition with Mediapipe. We hope that our city will serve as a valuable resource for researchers and newcomers to explore and develop innovative IoT and AI solutions, promoting experimentation and advancements in these fields.**

Keywords: Artificial Intelligence, Internet of Things, smart city, Aliot, LRIMa, ALIVEcode.

I. INTRODUCTION

In the last few years, the creation of smart cities has been a well talked about subject [1-2], and it has become even more so since the rising popularity of AI technology in education [3]. The idea of having a self-driven city with wireless transmission of data between motorized vehicles resulting in direct changes of behaviors at a real-life scale may seem far away, but we are convinced that it can be sooner with the correct workforce. AI has never been more important as of today, that is why it is of the utmost importance that this field be understood by newcomers since they will be the next generation of scientists most likely to bring this idea to life. This is why we created the LRIMa City (see **Fig.1**), a long-term project used as a testbed for new ideas and algorithms to promote the development of smart cities. In our previous work, we created an IoT development kit for personalized smart ecosystems named aliot [4] which was used in a previous version of the LRIMa City and is still used to this day. This version was a cloud-based solution solely relying on cloud computing for data sharing and route navigation. This solution had many issues such as high latency, unpredictability, and faulty navigation due to preprogrammed driving. The new LRIMa City we propose fixes those issues and adds new functionalities such as a speed radar and a smart parking with hands-recognition controls.

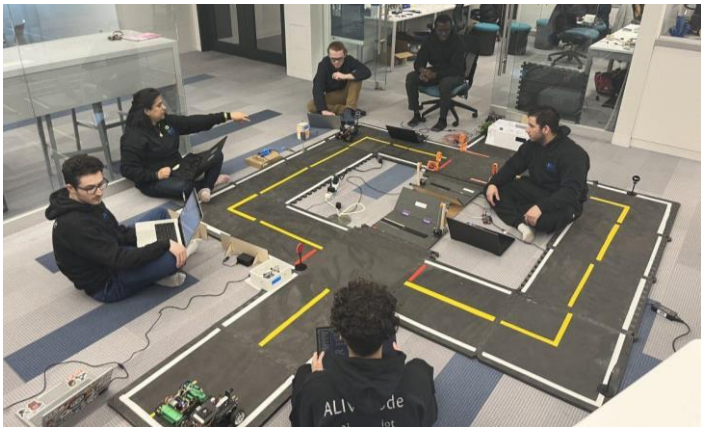

Fig.1. The LRIMa City being tested with the autonomous vehicles, the speed-radar and the smart bridge.

Our contributions in this paper can be summarized as follows: **(1)** We developed a CNN capable of predicting the steering angle from an input image. **(2)** We developed a traffic light detection model using YOLOv8 (You Only Look Once). **(3)** We created a fog on-device solution to decentralize data, increase security and reduce latency in the city. **(4)** We created a smart parking gate controllable using MediaPipe hand-tracking models. **(5)** We created multiple other enticing and interactive components in the LRIMa City. **(6)** We used our smart city as a testbed for autonomous vehicle navigation and recorded the process. **(7)** We showcased the LRIMa City in multiple workshops with newcomers to introduce AI and IoT.

Outline: Section II gives a brief overview of similar smart cities projects and compares them to the LRIMa City. Section III presents the different architectures for communications tested with the smart city. Section IV describes various smart components of the city. Section V shows the navigation system of our motorized vehicles throughout the city. Finally, section VI concludes the paper.

II. SIMILAR SMART CITIES

Numerous researchers have presented their solutions for smart cities, highlighting the growing interest in this field. In light of this, we compare in this section, the LRIMa City with other existing smart cities and elucidate the distinctive aspects of our own. By conducting this comparison, we aim to provide valuable insights into the innovative approaches and novel features employed in our smart city project, showcasing its potential to revolutionize urban environments. These comparisons can be seen in **Table 1** below.

To the best of our knowledge, the projects presented [5-7] in the table are the only ones offering a smart city as a playground for new experiments coupled with education.

Table 1. Similar smart cities projects compared to the LRIMa City

	Integrated learning	Integrated IoT Solution	Remote city control via web Platform	Self-driving cars	Traffic light detection	Real stop sign recognition	Modular hardware	Vehicle-to-vehicle communication	Infrastructure variety (city elements)
LRIMa City	✓	★	★	✓	✓	✓	✓	✓	****
DuckieTown [5]	✓			✓	✓		✓	✓	**
Micro:bit [6]	✓						✓		***
STEAM [7]	✓			✓					**

* 1 to 4 stars (few - many)

The ★ represent our key strengths

✓ Implemented Features

Empty cells indicate that the features are not implemented

For a more in-depth comparison, Duckietown and LRIMa City are both innovative projects that aim to explore and advance the field of autonomous vehicles, albeit with some key differences in their design and implementation.

Duckietown, first and foremost, is an open-source project that focuses on affordable and accessible education in the field of robotics and self-driving vehicles. It utilizes small "Duckiebot" vehicles that are equipped with a Raspberry Pi, and a camera. These bots navigate through a "Duckietown", a miniature town composed of roads marked with white and yellow tape, traffic signs, and obstacles. The Duckiebots primarily rely on simple computer vision techniques to navigate the town, interpreting the tape lines as roads and using colors and shapes to recognize signs and obstacles.

On the other hand, LRIMa City is a more technologically advanced and complex system. The self-driving cars in LRIMa City are equipped with a Raspberry Pi 4 and a front-facing USB camera. The LRIMa cars leverage more advanced AI models for navigation, including a steering angle prediction model based on NVIDIA's CNN, a traffic light detection model using YOLOv8, and a stop sign detection model. These models collaborate to ensure safe and efficient driving within the miniature smart city.

In addition, LRIMa City incorporates a wider range of smart city elements than Duckietown. While both have traffic lights, LRIMa City also has a speedometer and a smart parking, making it a more comprehensive testbed for exploring a variety of autonomous vehicle scenarios.

In summary, while Duckietown focuses on simple, accessible learning for autonomous vehicle concepts, LRIMa City steps up the technological complexity and incorporates more elements of a smart city to provide a more advanced testing ground for self-driving vehicle technologies.

III. The Smart City Connectivity Solutions

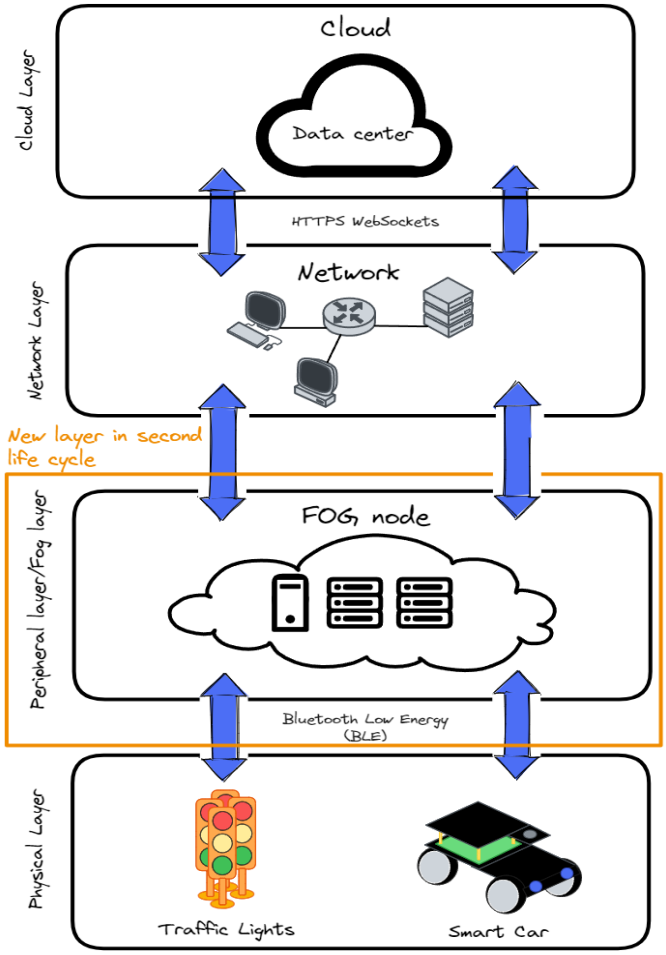

Fig.2. Fog-based approach

The LRIMa City underwent two distinct life cycles, each characterized by different approaches. The initial phase primarily emphasized establishing connectivity with the cloud. While this approach offered certain advantages, our team opted for a more decentralized solution due to encountered challenges pertaining to efficiency and reliability.

A. Cloud-based approach

During the first life cycle, the LRIMa City placed significant emphasis on achieving robust connectivity with the cloud. This approach aimed to leverage cloud-based technologies and services to enhance various aspects of urban life. However, as the project progressed, it became apparent that certain difficulties arose, particularly in terms of efficiency and reliability. These challenges prompted a revaluation of the initial approach, leading to a shift towards a more decentralized solution. Our old infrastructure is shown in **Fig.2** above except for the fog layer.

B. Fog-based approach

By adopting a decentralized approach, the LRIMa City project aimed to leverage distributed technologies and infrastructure.

This paradigm shift not only mitigated the shortcomings faced previously but also paved the way for new possibilities and opportunities. The subsequent life cycle of the LRIMa City represented a departure from the initial focus on cloud connectivity, emphasizing the advantages of a decentralized solution in terms of efficiency and reliability. A representation of the fog-based infrastructure is shown in **Fig.2** above.

IV. The Smart Components

One key factor that differentiates the LRIMa City and makes it unique from its competitors is the presence of multiple unique smart components designed purposefully for the city. Some of the components are shown in **Fig.3** below.

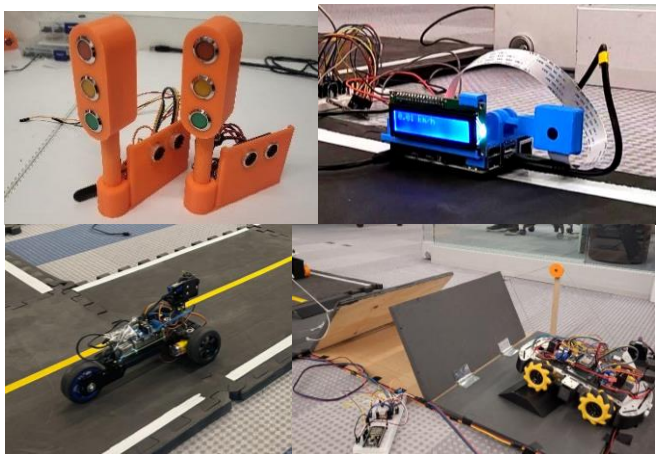

Fig.3. Some components used in the city

A. Self-Driving Vehicle

Our self-driving car, inspired from our previous work [8], equipped with a Raspberry Pi 4, two DC motors, a servomotor, a battery, and a single front-facing camera, navigates the LRIMa City using multiple AI models that work together to ensure safe and efficient driving within the miniature smart city. These key models include:

1. Steering Angle Prediction Model: SAPM

This model is inspired by and adapted from NVIDIA's end-to-end deep learning model for self-driving cars, as described in their paper [9] as well as a CNN for lane-recognition [10]. The model employs a CNN to predict steering angles from raw images captured by a camera mounted on the vehicle. By processing these images, the model can understand the current driving scenario and generate appropriate steering commands, ultimately enhancing the safety and efficiency of our smart city.

Our model, as illustrated in **Fig.4** below, consists of several layers, each designed to process input images in distinct ways. The normalization layer scales the input image to a fixed range, typically between -1 and 1, ensuring consistency in the data and improving model convergence during training. Following this, the convolutional layers perform core feature extraction by applying filters to the input images to detect patterns such as edges, corners, and textures. In our model, we use multiple convolutional layers with varying filter sizes and depths to capture both local and global features. For instance, the first convolutional layer has 24 filters with a size of 5x5 and a stride of 2, capturing low-level features like edges and corners.

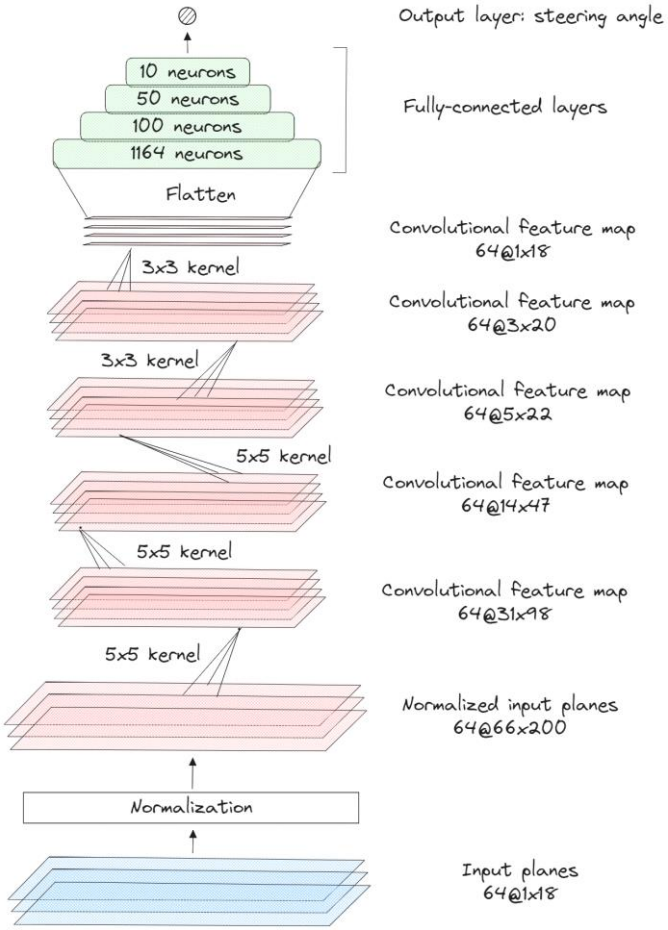

Fig.4. Steering angle prediction model architecture

The subsequent layers use different combinations of filter sizes, depths, and strides to learn increasingly complex patterns in the input images. Mathematically, the kernel convolution operation in the convolutional layers can be represented as:

$$G(m,n) = h * f(m,n)$$
$$= \sum_{j} \sum_{k} h(j*k) f(m-j, n-k)$$

Where $G(m,n)$ represents the output value at position (m,n) in the resulting feature map. The input feature map is denoted as $f(m,n)$, and the filter/kernel is represented by $h(j,k)$. The inner summation is taken over the values of j and the outer summation is taken over the values of k. This formula computes the convolution by sliding the filter over the input feature map and multiplying the corresponding elements of the filter and the local region of the input. The results of these element-wise multiplications are summed up to obtain the output value at each position (m,n) in the feature map. The convolution operation helps the model to learn spatial hierarchies by combining local features in the input to extract higher-level features in the output.

After the convolutional layers, activation layers introduce non-linearity to the model by applying activation functions such as the Exponential Linear Unit (ELU) function, enabling the model to learn complex, non-linear relationships between inputs and outputs. Each activation layer follows a convolutional layer, applying the ELU function element-wise to the output of the preceding convolutional layer. The activation function, in our case the ELU function, can be expressed mathematically as:

$$ELU(x) = x, \quad for\ x \geq 0$$

$$ELU(x) = \alpha(e^x - 1),\ for\ x < 0$$

Where x is the input to the activation function, and α $(alpha)$ is a hyperparameter (typically set to 1) that determines the slope of the function for negative inputs. The ELU function helps the model to mitigate the vanishing gradient problem, which can occur when training deep neural networks, by ensuring that the gradients do not become too small during backpropagation, thus facilitating the learning process.

Subsequently, a flatten layer is employed after the last convolutional layer to convert the multi-dimensional feature maps into a one-dimensional vector, enabling a seamless transition from the spatial feature extraction in the convolutional layers to the higher-level representation in the fully connected layers.

Finally, the fully connected layers serve as the last stage of the model, connecting the high-level features extracted by the previous layers to the output layer, which predicts the steering angle. The first fully connected layer has 1164 neurons, followed by a second fully connected layer with 100 neurons, and a third fully connected layer with 50 neurons. The reduction in the number of neurons across these layers enables the model to learn a compressed representation of the input features, focusing on the most relevant and crucial aspects for predicting the steering angle. The last fully connected layer has a single neuron that outputs the predicted steering angle. The computation in the fully connected layers can be represented by a matrix multiplication and an added bias term:

$$output = Activation(W * input + b)$$

In this equation, W represents the weight matrix, $input$ is the vector of input features, b is the bias term, and $Activation$ is the activation function (in our case, the Exponential Linear Unit or ELU). This equation highlights the linear combination of input features with weights and biases, followed by the application of the activation function to introduce non-linearity.

2. Traffic Light Recognition Model: TLRM

Our self-driving car also relies on a Traffic Light Recognition Model, which plays a vital role in ensuring the vehicle adheres to traffic rules and regulations. We utilized the YOLOv8 object detection architecture, building upon the pretrained YOLOv8l.pt default weights to train our custom model specifically for our miniature smart city's traffic lights.

YOLOv8 (You Only Look Once) is a state-of-the-art object detection framework that is known for its high accuracy and real-time processing capabilities. The architecture divides the input image into a grid, and each cell in the grid is responsible for detecting objects within its boundaries. This single-shot approach enables the model to efficiently process images and make predictions with minimal computational resources. The YOLOv8l.pt default weights provide a starting point for training the model, which has already been pretrained on a large-scale dataset, allowing for faster convergence and improved performance on our custom traffic light dataset.

3. Stop Sign Recognition Model: SSRM

Our stop sign detection model was sourced from an existing GitHub repository [11]. This pre-trained model was ideal for our purposes, as it had already been trained on various types of stop signs and exhibited excellent performance. The model is an XML cascade classifier, which is a popular approach for detecting specific objects in images.

Cascade classifiers are a type of machine learning model that use a cascade function trained from positive and negative images to detect objects. In this case, the cascade function is trained to recognize stop signs. The model operates by scanning an image in a sliding window fashion, analyzing different regions of the image at various scales to detect stop signs. If a stop sign is detected, the cascade classifier returns the coordinates of the bounding box around the detected stop sign.

By utilizing this pre-trained stop sign detection model from the GitHub repository, we were able to save time and resources that would have been spent on collecting and annotating a dataset, as well as training a model from scratch. The model's high performance and compatibility with our custom traffic infrastructure in the LRIMa City made it an ideal choice for our stop sign detection needs.

B. Smart Parking

The smart parking system in LRIMa City offers a seamless and efficient parking experience. It incorporates various technologies to enhance convenience and optimize parking space utilization.

Fig.5. Recognizing hand gestures to automatically open the parking barrier

One of the key features of the smart parking system is the barrier control mechanism shown in **Fig.5**. The parking

entrance is equipped with a camera that utilizes AI-powered MediaPipe technology [12]. This camera captures hand gestures or specific movements made by users. By recognizing these gestures, the system can automatically open the parking barrier, allowing smooth entry without the need for physical tickets or manual operation. This hands-free access control not only enhances user convenience but also promotes contactless interactions, which is especially important in today's context.

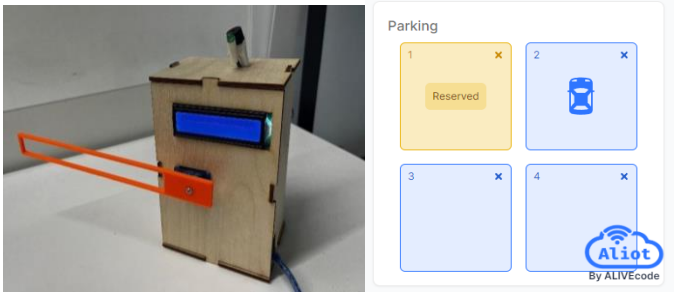

Fig.6. The LRIMa City's Smart Parking

Furthermore, the smart parking system utilizes a high-positioned camera located in a 3D printed tower to detect available parking spots as shown in **Fig.6**. This camera provides a panoramic view of the parking area, offering a better vantage point to detect all the parking spots. Through advanced computer vision algorithms, the camera scans the parking area in real-time, analyzing the occupancy of each parking space. The system then identifies and highlights the open spots on our web platform, providing users with an accurate and up-to-date view of parking availability. This feature eliminates the frustration of searching for parking spaces and allows users to easily locate an open spot from a high-level perspective.

By combining barrier control with real-time parking spot detection and display, the smart parking system in LRIMa City offers a comprehensive solution to optimize the parking experience. It simplifies access to the parking area, minimizes the time spent searching for parking spaces, and enhances overall user satisfaction.

V. Training & results - AI models

The following sub-sections will present the results of our numerous algorithms used in the LRIMa City. Alongside the quantitative results, we recorded a video showcasing the smart city with examples of interactions between the self-driving vehicle and the other smart components [13].

A. SAPM

The training process for our Steering Angle Prediction Model involved the use of a large dataset that was constructed by collecting data while remotely driving the car using a controller. During this process, images were captured, and their corresponding steering angles were recorded, providing the model with examples of the "appropriate way" to drive. To enhance the robustness and diversity of our dataset and prevent overfitting—a phenomenon where a model learns the training data too well, reducing its ability to generalize on unseen data—we employed data augmentation techniques, such as rotations, translations, flipping, and adjusting brightness or contrast. These techniques not only increased the size of our dataset but also helped our AI model generalize better across various driving scenarios and conditions. The augmented dataset was then divided into training and validation sets, with the former used to train the model and the latter to evaluate its performance. This separation allowed us to monitor and adjust the model to ensure it maintained good generalization capabilities, further minimizing the risk of overfitting.

Once the architecture of the neural network has been established and the dataset prepared, the model is trained using the augmented dataset of images and corresponding steering angles collected from actual driving scenarios. The training process involves minimizing the error between the predicted steering angles and the actual steering angles from the training data. To quantify this error, we use the Mean Squared Error (MSE) loss function, which calculates the average of the squared differences between the model's predictions and the ground truth steering angles:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Here, $\hat{Y}_i$ represents the predicted steering angles, Y_i denotes the ground truth steering angles, and n is the number of samples in the dataset.

To minimize the MSE loss, we employ the Adam (Adaptive Moment Estimation) optimizer, which is an adaptive learning rate optimization algorithm. Adam combines the advantages of two other popular optimization algorithms, AdaGrad and RMSProp, by dynamically adjusting the learning rate for each weight and bias parameter in the model based on the first and second moments of the gradients. This approach allows for faster convergence and improved performance during training.

During the training process, the model's parameters (weights and biases) are iteratively updated based on the gradients of the loss function with respect to the parameters. The Adam optimizer adjusts these parameters in a way that minimizes the MSE loss, ultimately fine-tuning the model to predict steering angles accurately from input images. The choice of the optimizer and loss function plays a crucial role in the model's ability to learn the mapping between images and steering angles effectively, ensuring the self-driving car can navigate safely and efficiently in various driving scenarios.

Fig.7 displays the training and validation loss as a function of the number of epochs. In the initial stages of training, we observed a substantial decrease in both training and validation loss within the first 10 epochs. Subsequently, the model's loss continued to decrease, albeit at a slower rate, eventually reaching a plateau around 55 epochs. Beyond this point, the decrease in loss became less significant, indicating the potential onset of overfitting if training were to continue for a few dozen more epochs. The model converged with a validation MSE of 0.037, indicating its effectiveness in predicting steering angles. To evaluate the robustness and generalization capabilities of our model, we subjected it to various driving scenarios and

conditions within the LRIMa City. In most situations, the model demonstrated strong performance, accurately predicting steering angles under different lighting conditions and environmental factors. However, we encountered challenges in scenarios with sharp turns, where the field of view (FOV) of the front-facing camera wasn't wide enough to capture both lane lines. This observation suggests potential areas for improvement and further model refinement.

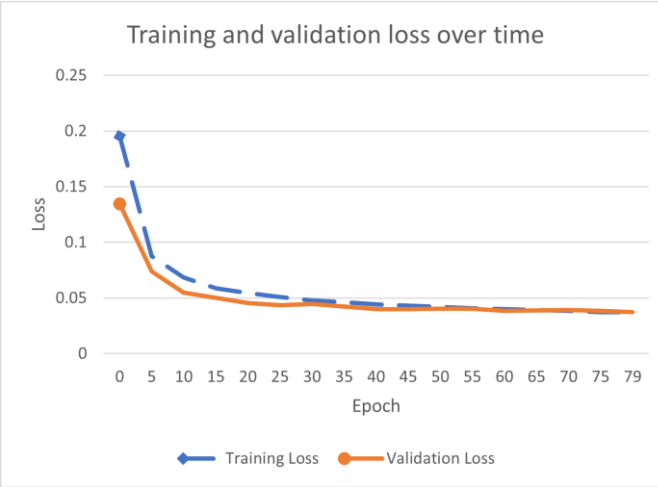

Fig.7. Training and validation loss over time (epochs)

Fig.8. Qualitative evaluation of steering angle predictions

Fig.8 presents a qualitative evaluation of our model, showcasing its predictions on real-world images alongside the ground truth steering angles. In most cases, the model accurately predicted the steering angles, allowing the self-driving car to navigate the city efficiently. However, a few instances revealed discrepancies between the predicted and ground truth angles, especially in complex driving situations. These cases provide insights into the model's limitations and can guide future refinements to enhance its performance.

Overall, the results indicate that our Steering Angle Prediction Model is effective in predicting steering commands for our self-driving car in the LRIMa City. The model demonstrates good generalization capabilities across various driving scenarios and conditions, making it a valuable component of our miniature smart city's autonomous driving system. Future work will focus on improving the model's performance in challenging situations and further optimizing its architecture for increased efficiency and accuracy.

B. TLRM

To create a suitable dataset for training the Traffic Light Recognition Model, we collected images of our custom traffic lights in various states (red, yellow, green) and under different lighting conditions. These images were then manually annotated, with bounding boxes drawn around each traffic light and labeled according to its current state. This process of annotating and labeling the images ensured that our model could learn to accurately detect and recognize the different traffic light states in the LRIMa City.

After assembling the dataset, we divided it into training and validation sets. The training set was used to fine-tune the YOLOv8 model with our custom traffic light data, while the validation set was employed to evaluate the model's performance during the training process. This allowed us to monitor the progress of our model and make any necessary adjustments to prevent overfitting and optimize its generalization capabilities.

Fig.9 depicts the model's Mean Average Precision (mAP@50-95), a widely utilized metric in object detection tasks. This metric assesses the model's accuracy in detecting objects and their corresponding classes across various Intersection over Union (IoU) thresholds, ranging from 0.5 to 0.95 with a step size of 0.05. During the training process spanning 50 epochs, we observed that the mAP reached a plateau around the 20th epoch, indicating that further training beyond this point would

yield diminishing returns and potentially lead to overfitting. Notably, our model achieved an impressive mAP@50-95 score exceeding 80%, demonstrating its robust performance in accurately detecting and classifying traffic lights.

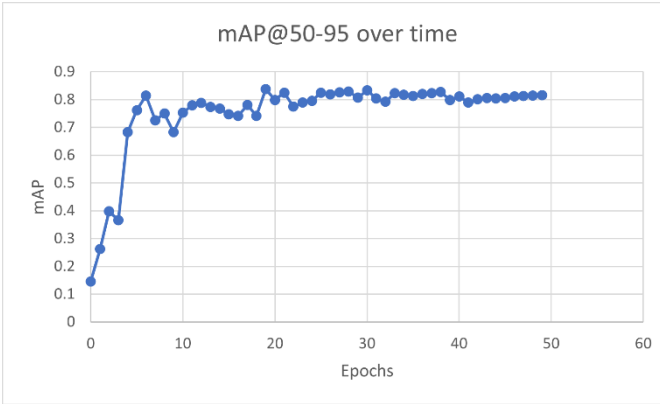

Fig.9. mAP@50-95 over time (epochs)

To provide further insight into the model's performance, **Fig.10.A** showcases a selection of real-life images from the validation set containing labeled objects, representing the actual ground truth values, allowing for a direct comparison with the model's predictions depicted in **Fig.10.B** below. These images demonstrate the model's ability to detect and classify traffic lights under varying conditions and scenarios, revealing that the model can accurately recognize the vast majority of images with only a few instances of misclassification (false positives and false negatives), especially with red and yellow lights.

Fig.10.A. Images presenting labeled traffic lights

Fig.10.B. Images presenting the model's predictions with their corresponding confidence scores

By analyzing these metrics and visualizations, we can conclude that our YOLOv8 Traffic Light Recognition Model exhibits strong performance in detecting and classifying traffic lights within the LRIMa City. However, there may still be areas for improvement, as highlighted by some of the graphs and visualizations.

VI. CONCLUSION

In conclusion, the LRIMa City project has undergone two distinct phases, each contributing to its evolution and development. The initial phase centered around implementing an IoT solution utilizing a cloud-based infrastructure, while the current phase has shifted the focus towards AI-powered navigation and enhanced interactions between components within the fog layer. The project has proven to be an ideal platform for innovation, encouraging the exploration of novel ideas and technologies.

Looking ahead, our future endeavors aim to advance the LRIMa City even further. One key objective is the development of a robust traffic navigation system, enabling multiple cars to communicate and cooperate with each other seamlessly. This collaborative approach will enhance the efficiency and safety of the self-driving cars within our smart city, facilitating smooth and reliable transportation in complex scenarios.

Additionally, we aspire to integrate an obstacle recognition and avoidance model into our self-driving cars. By leveraging cutting-edge AI algorithms, our vehicles will possess the capability to identify and respond to potential obstacles in real-time, ensuring enhanced safety and mitigating risks during navigation within the LRIMa City.

The LRIMa City serves as a living laboratory for ongoing innovation, and we are committed to continuously pushing the boundaries of AI, IoT, and smart city technologies. Through our ongoing research and development efforts, we strive to create a future where autonomous systems can navigate urban environments efficiently, safely, and seamlessly.

ACKNOWLEDGMENT

We would like to thank FRQNT, Énergie Scolaire, PIA and Innovations ALIVEcode inc. for financially supporting this research. We would also like to thank Kassem Kandil, Makhlouf Hennine, Ramy Naffati, Yao Kounakou and Francis M. Gosselin who greatly contributed to the *Smart City* project.

REFERENCES

[1] Anthopoulos, Leonidas. (2017). The Rise of the Smart City. 10.1007/978-3-319-57015-0_2.

[2] Javed, A.R.; Shahzad, F.; ur Rehman, S.; Zikria, Y.B.; Razzak, I.; Jalil, Z.; Xu, G. Future smart cities requirements, emerging technologies, applications, challenges, and future aspects. Cities 2022, 129, 103794.

[3] Chen, Xieling & Xie, Haoran & Zou, Di & Hwang, Gwo-Jen. (2020). Application and theory gaps during the rise of Artificial Intelligence in Education. Computers and Education: Artificial Intelligence. 1. 100002. 10.1016/j.caeai.2020.100002.

[4] J. Rezgui and E. Soldevila, "Novel IoT Development Kit for Personalized Smart Ecosystems: Aliot," 2022 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT), Alamein New City, Egypt, 2022, pp. 54-59, doi: 10.1109/GCAIoT57150.2022.10019206.

[5] L. Paull et al., "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 1497-1504, doi: 10.1109/ICRA.2017.7989179.

[6] Clarinval, A.; Simonofski, A.; Henry, J.; Vanderose, B.; Dumas, B. Introducing the Smart City to Children: Lessons Learned from Hands-On Workshops in Classes. Sustainability 2023, 15, 1774. https://doi.org/10.3390/su15031774.

[7] Ruiz Vicente, F.; Zapatera Llinares, A.; Montés Sánchez, N. “Sustainable City”: A Steam Project Using Robotics to Bring the City of the Future to Primary Education Students. Sustainability 2020, 12, 9696. https://doi.org/10.3390/su12229696.

[8] J. Rezgui, F. Jobin, S. Beaulieu and Z. Ardekani, ‘Autonomous Learning Intelligent Vehicles Engineering in a Programming Learning Application for Youth: ALIVE PLAY, accepted IEEE ISNCC 2021, Dubei.

[9] M. Bojarski et al., “End to End Learning for Self-Driving Cars”, arXiv preprint, arXiv:1604.07316, 2016.

[10] K. Yassin, G. Souhir, L. Lew, J. Maher, M. Mehrez, A. Mohamed. (2022). Deep embedded hybrid CNN-LSTM network for lane detection on NVIDIA Jetson Xavier NX. Knowledge-Based Systems. 10.1016/j.knosys.2021.107941.

[11] Eshgin Guluzade’s stop sign detection github repository: https://github.com/EshginGuluzade/stop_sign_detection [last visited 11 May 2023].

[12] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C., & Grundmann, M. (2020). MediaPipe Hands: On-device Real-time Hand Tracking. ArXiv. /abs/2006.10214.

[13] The Smart City feature video: https://www.youtube.com/watch?v=WLPV0lk77A0. [last visited 29 May 2023].