Implementation of Unified Local-Cloud Decision-Making via Reinforcement Learning on a mobile platform

Kote Batonisashvili

***Abstract*— Mimicking real-life scenarios to the best of our abilities has always been the goal when designing and operating within a simulator. Unfortunately, we can never incorporate enough uncertainty in our programming to fully accommodate for all unknowns and failures which can occur. Building on the work from Unified Local-Cloud Decision Making via Reinforcement Learning (UniLCD) [16], our proposal is to introduce this framework to a physical mobile system for a sim-to-real comparison and analysis. We begin this process through instantiating a baseline imitation learning (IL) algorithm for both the local system and the cloud system. For the design of the reward function as well as the training, a functional IL algorithm must be in place. Utilizing the available sensors on the vehicle, we develop an obstacle avoidance policy for two separate models, one more involved and robust which is stored on the cloud, and a local lightweight model designed for quick, albeit simple, analysis. Our policies were tested against collected data and showed significant promise in active obstacle avoidance. Our local and cloud models were successful at avoiding obstacles in over 90% of the test cases, with the latter performing significantly better at the expense of energy used.**

I. INTRODUCTION

A. Motivation

Sim-to-real challenges are an ongoing battle among researchers. With simulations providing conditions that are near perfect, data alterations, and random noise are frequently added for additional information. An abundance of information is a necessity for all possible events or actions, but regardless of how much injection is performed into our collected data, real-life situations will always come as a new surprise. Simulations also come with the added benefit of hardware and software stability under different conditions, which is never the case for the real world. Storage space, sensor accuracy, unexpected item degradation, lower efficiency tied to a draining charge, and many other limitations must be taken into consideration. Converting simulations to real-world scenarios is therefore fundamental to ensure feasibility in the presence of uncertainty.

To address hungry energy demands during experimentation, UniLCD provides the framework for a lightweight model with little energy consumption on deployed systems, and offloading energy-intensive computations to the cloud. This policy allows a mobile agent to utilize its energy capacity efficiently and prolong the operational duration.

Our contributions offer the implementation of this framework on a physical system assembled together with embedded sensors and hardware. We develop a lightweight Imitation Learning (IL) based policy to be utilized locally on the vehicle, and a similar, but significantly more involved, model to be used on the cloud. We adjust these pre-trained policies to accommodate to our sensors and collected data, as well as alter training parameters to suit our goals more effectively. With a limited amount of sensors and relatively small dataset, we train effective models which perform obstacle avoidance with a success rate higher than 90%.

B. Related Works

Implementation of learning models on physical systems has always been somewhat of a challenge due to the inherently high risks that are associated with it. Crashes in simulations are simple statistics and data, but crashes in the real world result in setbacks, delays, or sometimes serious injuries. With several decades of research, path finding challenges and navigational training has been studied across many different works [4], [6], [13]. Researchers in these works discuss the societal and environmental challenges that autonomous systems encounter when working in the real world, taking into consideration the uncertainty and randomness of human behavior, confusion with localization from dynamic obstacles, and the difficulty of RL policy implementation due to reward function design. Our implementation of UniLCD luckily avoids most concerns discussed through a combination of simple path planning assisted by basic obstacle avoidance, and the curation of the reward model in IL policy choosing as opposed to path planning focused.

Imitation learning for navigation is not recent either, with some papers dating as far back as the early 1980's with robots like ALVINN performing supervised imitation learning [15]. We see the extensions of this work all across the world in the modern day, with experiments in simulations and physical systems achieving remarkable results. Many works [1], [18]have seen a preference towards the CARLA simulator, having robust capabilities of recording data to parse for training, developing models which specialize in end-to-end learning with deep imitation learning. We see others [17] utilize open source software such as OpenAI's gym or more involved simulators like TORCS [9], where they focus on more advanced techniques like DAGGER and generative adversarial learning respectively. While the comparisons and survey's proposed in these show advantages of different policies in different environments, they neither provide any physical experiments nor deviation towards combining multiple training methods together.

We see more involved and nuanced learning integrations in [2], [3], [5], [7], [10]–[12], implementing multiple layers in how their agents generate rewards and policies based on those. Reinforcement learning seems to be a prominent component in most of these works, allowing the researchers

to craft intricate systems which have major differences and yet contain underlying similarities in structure. The following works [5], [7], [10]–[12], all share the same architecture of developing a reward function or hierarchy which proceeds to pass an action to an imitation learning model, with [12] sharing relatively similar goals to UniLCD in focusing on energy optimization for multiple microgrid agents during interruptions or power outages.

With unique implementations of RL policies dictating further IL action, the closest to our work can be seen in [2], [3], [14], in which these policies are directly guiding a physical agent for the purpose of autonomous path planning and driving. [14] implements a successful imitation learning model on a rally car which demonstrates incredible traversed distances of up to 350+ meters without crashing. Introducing RL into the equation, [2] presents a model named deep imitative reinforcement learning (DIRL) which trains from expert policies, alongside its own RL reward function which is based on speed, collision, and uncertainty. While similar to UniLCD, it does not take energy considerations nor does it offload computation to a cloud policy, but instead retains both in a single model pipeline. The closest to our work in terms of physical experimentation may be seen in [3], in which a deep imitation learning policy is developed on a platform that is identical to ours with a few changes in sensor and control inputs. Codevilla notes in their setup the importance of data augmentation and noise injection for their imitation learning model. Data adjustments were also made in our contributions for good, diverse, quality of training.

II. Methodology

To implement such a framework on a physical system, a suitable base was procured for its modularity. The Secure Computing Systems Lab at Boston University had an RC car that was available to use as our modular base. The outer shell of the vehicle was discarded and only the internal components were used for modification and integration. An Nvidia Jetson running Ubuntu, 10 inch LCD display, and an RGB depth camera were the only embedded devices during data collection. We powered the hardware devices with a car battery jump-starter. The wheel motors were powered by built-in battery packs that came with the RC car. Figure 1 shows the vehicle used for data collection.

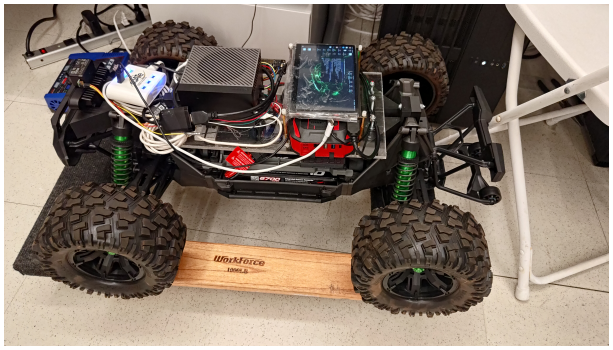


Fig. 1: Our mobile vehicle with custom embedded hardware which was used for data collection.

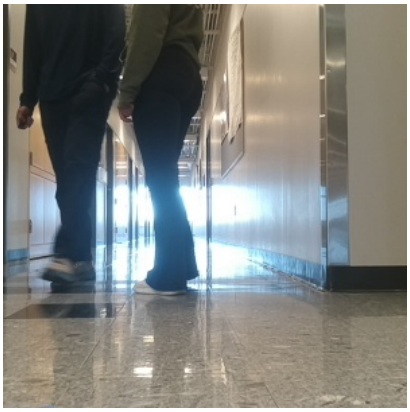
Connection to the camera and the wheels was established through a pre-existing Python script. Wheel control was made possible by ”warming up” the motors through an increasing angle command of 94 through 99, where 99 would correspond to full throttle, and any previous value would disengage them entirely. This warm-up was a necessity before any further command could be given and would likely need to be addressed in further research. Once the start-up procedure was complete, a data collection script was coded through asynchronous commands passed to the sensors and the wheels. We utilized a Bluetooth keyboard connected directly to the Jetson as a remote controller. The downside of connecting to the motors through a keyboard is the the complete lack of sensitivity to the throttle command. Holding down the **W** key mapped the motor angle to the ”99” value which would engage the throttle at 100%, forcing the vehicle to accelerate to its top speed, which was also capped for safety reasons. Keys **A** and **D** were mapped to turn left and right respectively. The reverse functionality of the vehicle was completely disabled for simplicity, safety, and to avoid unnecessary direction changes. We are thus left with three direct commands which can control the vehicle; steer left, throttle, steer right. An implicit fourth command of *stop* also exists by avoiding any actions altogether.

A. Data Collection

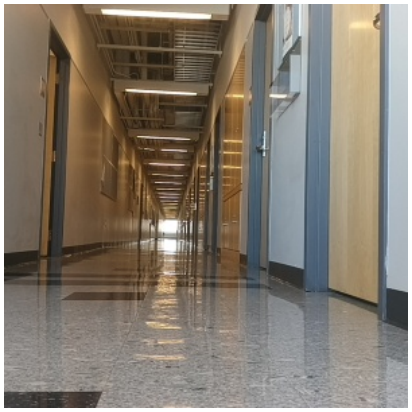
Gathering data for any physical experiment can usually come at a challenge, gathering good data even more so. Our vehicle was limited to just a single sensor during data collection through an RGB + depth camera. While keyboard command inputs combined with their respective RGBD frame may provide some information on the environment, it is difficult to train a robust policy on just those alone. Therefore to create a feasible objective which the agent could follow, a simple obstacle avoidance task was our focus during data collection.

Training for obstacle awareness is relatively simple, if there is an obstacle in the way of our vehicle, no commands were given, to simulate avoidance. If there is no obstacle detected directly in front of the vehicle, it may proceed forward with regular throttle commands. Approximately 1000 images were used for model training that were split 80 to 20 for training and validation. Another 1000 images were used for model comparison between predicted action versus the ground truth which is later discussed in the results section.

Gathering images consisted of several test subjects walking in front of the vehicle to simulate pedestrians crossing hallways or simply standing in front of the vehicle for prolonged periods of time before finally moving out of frame. Several different environments and environmental obstacles were chosen for recording, such as an empty hallway with two pedestrians, a room intersection with a single person wandering around the vehicle, bright sunlight coming from windows, reflected hallway lights from ceiling lights, reduced visibility close encounters with pedestrians, populated plazas with rushing students, and many more. Figures 2a and 2c show examples of obstacles in front of our vehicle, simulating



(a) Hallway example with multiple people and bright sunlight. Difficult environment with STOP action.



(b) Empty hallway example. Simple environment with GO action

(c) Different hallway example with a single person directly walking towards the vehicle. Simple environment with STOP action.

Fig. 2: RGB examples taken during data collection to visualize complex and simple environments for training.

difficult environments for analysis and simple environments respectively. Appropriate frames were also added to simulate the full throttle response of the vehicle when stopping was no longer necessary, as can be seen by the empty hallway in figure 2b. Data diversity for training was essential to tackle the issues of generalization as all data was recorded indoors. Several setbacks were encountered during collection which significantly reduced the size of our usable data for training. This is further discussed in the limitation section.

B. Problem Formulation

Revisiting the problem formulation from [16], our framework consists of 3 policies working hand in hand. Our reinforcement learning (RL) policy must actively monitor the current state and provide appropriate weights on which IL policy must be chosen for optimal actions. Figure 3 shows the high-level overview of the process.

Before starting RL training, a local and cloud IL must be in place. The local and cloud policies must operate under the assumption that the latter will have better performance at the cost of computational speed and energy consumption. Each policy must consist of its own design, training, and testing. One key detail to note when testing our policy is the clipping factor used for our throttle response. Our actions are limited to a binary STOP or GO command, where they are respectively given 0 or 1 as the input. This design is determined by the sensitivity of our throttle by holding down the W key on our keyboard. With the use of a joystick, we would see smoother action that we could map for increased or decreased speed, but since keyboard inputs are binary, so were our actions. Our neural networks returned normalized values between 0 and 1, and since these are not valid inputs, we perform a basic clip, where any value under 0.5 is evaluated as a STOP command, and all actions above are determined as a GO command.

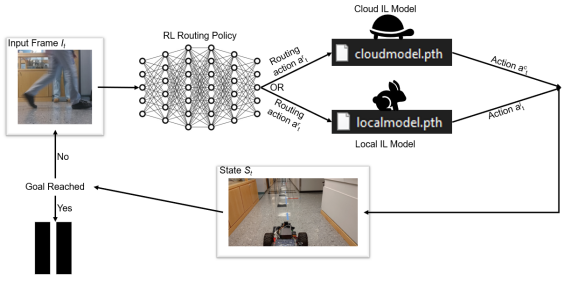


Fig. 3: Visualization of UniLCD. For every input frame, the RL model is provided with relevant data which subsequently gets passed to the local or cloud model depending on computational needs. Once the IL policy provides an action to the agent, it checks the current state to see if the object is reached or not.

1) Local Policy: To receive quick decision making for non-complex inputs, the local IL model was designed specifically with lower computational power in mind. With great results coming from [8], we build on this neural network architecture with slight modifications to accept our input data. Since we have an RGB + Depth camera installed to monitor the front of our vehicle, the neural network was modified to accept a fourth parameter to include depth for the parsed images. Consulting the training recipe included in [8], the Monte Carlo dropout (dropout) rate was kept at 0.2 to encourage adaptability within the model to deal with uncertainty. We reduced the batch size to 16 and the number of epochs to 200 to ensure the model remained relatively lightweight. Learning Rate (LR) and weight decay was adjusted in accordance to the reduced data size and more epochs, with both set to $1e-4$ and $1e-5$. As a result of our limited dataset during data collection, LeakyReLu was chosen for our neural network in an attempt to avoid overfitting. Thus the overall design of our neural network can be summarized as a modified MobileNetV3, which takes in 4 inputs and is reduced down to a single output which dictates when to use throttle, and when not.

2) Cloud Policy: The cloud policy while sharing some setup and training parameters with the local, had fairly large differences in architecture. Since we were no longer constrained entirely by energy consumption of our vehicle, the cloud model was capable of running through more layers and deeper networks for better results. We resorted to using ResNet101, a large classifier with a deep architecture that is well established in its implementation within Pytorch. This model was also modified in similar ways, where we changed the input parameters to accept four channels to incorporate depth, with a single output to simulate throttle response. The same optimization techniques were used, such as but not limited to: LeakyReLu, dropout rate of 0.2, 80 to 20 training and validating dataset, and a weight decay of $1e-5$. With the objective of being a robust, accurate policy, we modified the learning rate to $5e-5$, number of epochs to 2024, and batch size of 64. To ensure training is performed continually, we created a variable which tracks average loss. Once the model goes through 300 epochs without significant improvement, the training is marked as complete and the policy is generated.

III. Results

A. Local Model

We evaluate the performance of our model through a Mean Square Error loss during training, and a direct comparison between model prediction and actual input. We can see the training loss and validation loss in figure 4, where as mentioned previously, 80% of the data was used for training and the rest for validation.

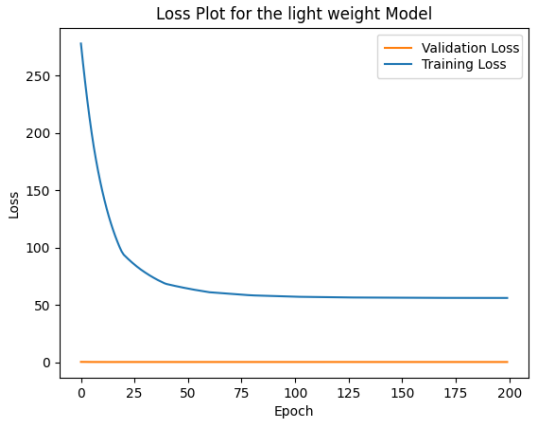


Fig. 4: Plot comparing training and validation loss of the lightweight model.

As we can see from training, the model converges to a loss of approximately 60 from the gathered data and does not improve further. Since our output is a simple binary command of 1 or 0, these results are indicative of either insufficient amount of training data, or the policy is underfitting due to a lack of diversity in training. We could take the lackluster performance into consideration as a result of clipping occurring post training.

Our comparison of predicted action versus ground truth can be seen in figure 5, however this time, our previously discussed clipping for throttle input is taken into consideration. We see significant improvement in the lightweight model immediately, where it can accurately predict the appropriate response over 90% of the time. We see from the results that it prefers to predict GO more often than stop, indicating a need for more safety measures to be in place for collision avoidance if an incorrect prediction is passed.

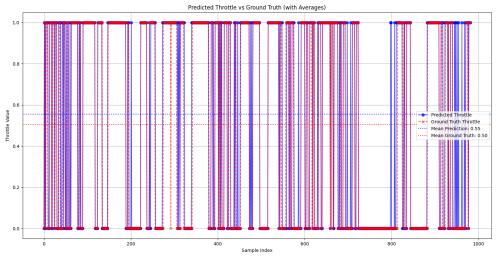


Fig. 5: Predicted actions vs ground truth for our lightweight model. Our data is split where half of the frames require a throttle response, and the other half do not.

These results for the lightweight model fall within acceptable bounds, since it is not designed for perfect predictions and instead must make quick actions.

B. Cloud Model

Our cloud based model followed the same standard for performance evaluation as the local policy. Using a deep model like ResNet101 on a small dataset seems counter-intuitive, especially for the simple task of STOP and GO throttle response. Incorporating such a large classification network within training was by design, as our collected data was exposed to several complex environments. We see the consequence of this large model in figure 6, where the plot reflects poor model training.

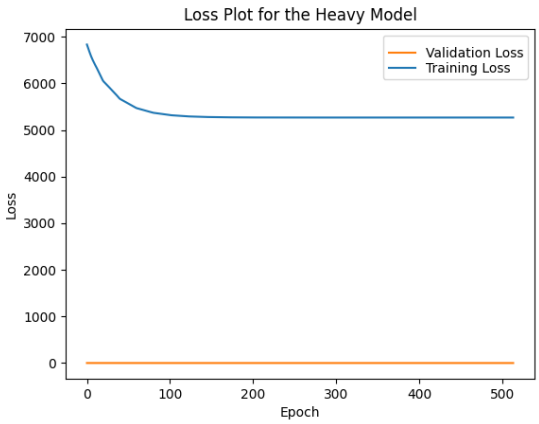


Fig. 6: Plot of validation loss and training loss against epochs.

With an extremely high loss during initialization and convergence, using this model for predictions would not be

feasible in the slightest. Recalling the local model analysis, these results are quite similar and are expected, especially before the clipping parameter is introduced when comparing predictions to ground truth.

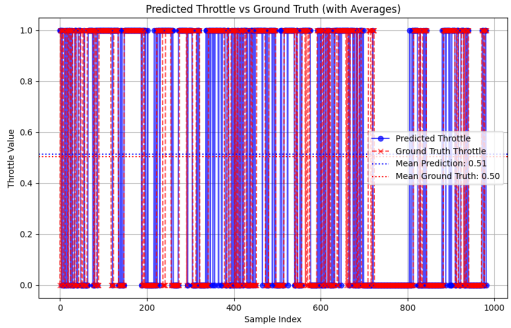


Fig. 7: Predictions vs ground truth when using our modified ResNet101 trained model.

When looking at figure 7, we see incredible results when applying our 0.5 clipping value. The larger model outperforms the lightweight local model significantly and provides incredible results. Working with significantly more convolutional layers, this policy has a smoother gradient for feature extractions in comparison, resulting in more accurate predictions. Having an incredible 98% accuracy for throttle prediction provides us with an operational model that would perform well in the real world. We would still need secondary fail-safes as this policy would provide mis-inputs, something that physical experiments must avoid.

IV. Limitations

As with all physical experiments, we ran into several setbacks which significantly hindered our progress at almost every step of the way. Our first workaround was operating with a single sensor on board that did not have access to ground truths for vehicle localization. At the time, we did not have access to accurate IMU or GPS information which simplified our approach to focus only on obstacle avoidance. Without access to other mapping methods, implementation of the RL cloud switching policy was not possible as we did not have access to trajectory mapping and vehicle speed, two important components of the reward function presented in [16].

Adjusting our workflow to accommodate the limited number of sensors, our data collection periods suffered from battery power discharge every half hour or so. With so many hardware devices connected to the jumpstarter, it was unable to maintain charge for longer periods of time, reducing our collection duration significantly. Wheel motor efficiency entirely depended on the battery packs the vehicle came with.

The largest setback which we were unable to resolve was latency delays between provided commands and agent response time. With prolonged used of the vehicle, the commands provided to the motors were significantly delayed and at times were not registered entirely. In most of our sessions, the vehicle would not register our commands which to the best of our knowledge was a result of quick battery discharge. Capturing good data became time limited during every session. Rushing against a time constraint resulted in significantly fewer frames gathered for training and other unexpected errors.

We would like to recall the warm-up process that our vehicle must go through before it interprets keyboard commands properly. The warm-up process must be completed with the vehicle wheels lifted above the ground to avoid full throttle before such a command is issued. Due to the battery discharge time constraint, we had to undergo the warm-up process multiple times in a day to resume data collection. Recall also the explicit removal of the reverse functionality that was mentioned previously. To the best of our knowledge, the vehicle's warm-up process encountered an error as a result of the battery discharge, which allowed the vehicle to use full throttle in reverse and crash into a nearby wall, cracking the hardware platform and putting the vehicle out of commission for an extended period of time.

V. Conclusion

In this paper, we develop an operational lightweight imitation learning policy which resides on a physical mobile platform, alongside a more robust and deeper imitation learning model which would be stored on the cloud. Our goal was to develop a working baseline for the implementation of UniLCD through utilizing a custom built vehicle with embedded hardware and sensors for data collection. We leverage currently available models as a baseline on the basis of energy consumption and tailor them based on our input and output needs. With obstacle avoidance as our primary objective, our trained models operate with a 90% or higher success rate in accurate predictions when compared to actual inputs. These results provide a solid foundation for full implementation of UniLCD.

A. Future Works

Working with a limited number of sensors and multiple workarounds for hardware setbacks, the full implementation of UniLCD would not be feasible. Through a recent acquisition of a GPS/IMU sensor, direct continuation and implementation of RL is entirely possible. To ensure efficient use of time and proper results, modifications to fine motor control and addressing the power-hungry battery consumption is a must. The latency delay which is introduced by both of those issues creates an unnecessary hurdle which delays and complicates data collection. Once that is resolved, extending the work to accommodate path planning and variable speed is only a matter of time.

Keeping energy efficiency in mind, an interesting extension to this work would be an implementation of a low-level, "power saving" mode for simple and generic tasks. Our devices at home such as personal laptops, smart phones, refrigerators, and many others devices typically come with a specific battery focuses mode which has a reduction of energy consumption through changes to its behavior. If our vehicle

for example, is following a straight path for an extended period of time with little to no obstacles, it can follow a power saving policy which prioritizes energy consumption as the main reward generator. This energy efficient mode would be different than our current local policy as it would only activate during simple tasks as mentioned previously, such as a straight route for several hours.

Another possible extension to the work can revolve around environmental disturbances during transit. While most vehicles operate under relatively stable conditions where road changes are not encountered, the size and off-road capabilities of the base vehicle grant it the ability to traverse more adversarial terrain. An example of this can be something as simple as cracks in the asphalt due to road degradation, or changes in terrain type from sidewalk to gravel or mud. Separating obstacles from these changes would provide more robustness to our agent in path planning.

B. Acknowledgments

This work was possible through the continuous help of Adwait Kulkarni, who was a fantastic teacher on the operation of our physical agent. Adwait provided historical context on how the vehicle was developed, the startup procedure, and what would constitute as good data during collection periods. Without his guidance, implementation of the imitation learning model would not have been possible. We would like to thank Dr. Ohn-Bar for his recommendations on the choice of topic alongside his permission to utilize the vehicle.

REFERENCES

[1] M. Abdou, H. Kamal, S. El-Tantawy, A. Abdelkhalek, O. Adel, K. Hamdy, and M. Abaas. End-to-end deep conditional imitation learning for autonomous driving. In *2019 31st International Conference on Microelectronics (ICM)*, pages 346–350, 2019.

[2] P. Cai, H. Wang, H. Huang, Y. Liu, and M. Liu. Vision-based autonomous car racing using deep imitative reinforcement learning, 2021.

[3] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning, 2018.

[4] A. Gaydashenko, D. Kudenko, and A. Shpilman. A comparative evaluation of machine learning methods for robot navigation through human crowds. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 553–557, 2018.

[5] K. Hamahata, T. Taniguchi, K. Sakakibara, I. Nishikawa, K. Tabuchi, and T. Sawaragi. Effective integration of imitation learning and reinforcement learning by generating internal reward. In *2008 Eighth International Conference on Intelligent Systems Design and Applications*, volume 3, pages 121–126, 2008.

[6] C. He, W. Zhang, and T. Wang. Reinforcement learning with auxiliary localization task for mapless navigation. In *2020 Chinese Automation Congress (CAC)*, pages 3069–3073, 2020.

[7] J. Ho and S. Ermon. Generative adversarial imitation learning, 2016.

[8] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for mobilenetv3, 2019.

[9] G. Lee, W. Oh, J. Oh, S. Shin, D. Kim, J. Jeong, S. Choi, and S. Oh. Semi-supervised imitation learning with mixed qualities of demonstrations for autonomous driving. In *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, pages 20–25, 2022.

[10] J. Li, S. Huang, X. Xu, and G. Zuo. Generative adversarial imitation learning from human behavior with reward shaping. In *2022 34th Chinese Control and Decision Conference (CCDC)*, pages 6254–6259, 2022.

[11] Z. Li. A hierarchical autonomous driving framework combining reinforcement learning and imitation learning. In *2021 International Conference on Computer Engineering and Application (ICCEA)*, pages 395–400, 2021.

[12] Y. Lin, Z. Ni, and Y. Tang. An imitation learning method with multi-virtual agents for microgrid energy optimization under interrupted periods. In *2024 IEEE Power Energy Society General Meeting (PESGM)*, pages 1–5, 2024.

[13] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh. Core challenges of social robot navigation: A survey. *ACM Transactions on Human-Robot Interaction*, 12(3):1–39, 2023.

[14] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots. Agile autonomous driving using end-to-end deep imitation learning, 2019.

[15] D. A. Pomerleau. Alvinn: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, page 305–313, Cambridge, MA, USA, 1988. MIT Press.

[16] K. Sengupta, Z. Shagguan, S. Bharadwaj, S. Arora, E. Ohn-Bar, and R. Mancuso. Unilcd: Unified local-cloud decision-making via reinforcement learning. *arXiv preprint arXiv:2409.11403*, 2024.

[17] X. Sun, M. Zhou, Z. Zhuang, S. Yang, J. Betz, and R. Mangharam. A benchmark comparison of imitation learning-based control policies for autonomous racing, 2023.

[18] M. Vijayalakshmi, D. Suvitha, and P. Sivaguru. End-to-end autonomous vehicle navigation using imitation learning. In *2023 12th International Conference on Advanced Computing (ICoAC)*, pages 1–6, 2023.