Impact of Delays and Computation Placement on Sense-Act Application Performance in IoT

Pragya Sharma, Mani B. Srivastava
Dept. of Electrical and Computer Engineering
University of California Los Angeles
Los Angeles, USA
{pragyasharma, mbs}@ucla.edu

***Abstract*—This paper investigates the challenges posed by delays in *Closed-Loop Sense-Act Systems* in the context of Adversarial Internet of Things (IoT) applications. Prior work focused on studying the impact of delays on a single resource-constrained platform. To capitalize on the capabilities of different computing platforms, this work investigates the adaptation of control placement to optimize application performance in distributed settings. An Adaptive Control Placement (ACP) strategy is introduced, which dynamically switches between a local controller with lower accuracy and a cloud controller with higher accuracy based on network dynamics, optimizing overall application performance. The effectiveness of the ACP strategy is evaluated using a simulated Vehicle Following application in the PyBullet simulator. The results demonstrate that in terms of a time-to-complete (TTC) metric, the ACP strategy consistently outperforms strategies that use a fixed combination of controller type and location (e.g., PID at Local and MPC at Cloud) across various deadline scenarios.**

***Index Terms*—Sense-Act system, Adversarial IoT, PID, MPC, Adaptive Control**

I. Introduction

Sense-Act systems are fundamental to many Adversarial Internet of Things (IoT) applications. Examples include radar or camera-based tracking, intruder interception, and guided missile interception. These systems primarily consist of two modules: a sensing module responsible for measuring the state of the environment and an actuation module that utilizes this measurement to take suitable actions to achieve desired results. The feasible methods for sensing and actuating vary based on the specific nature of the application and its associated performance metric.

For instance, in radar-based intruder tracking scenarios, utilizing a sensing module with high latency and high accuracy would be deemed acceptable if the primary goal is to achieve the most precise representation of the environment possible. Conversely, a goal of obtaining the most accurate representation of the environment as quickly as possible would require sensing modules with lower latency or those capable of predicting the environment's state at the time of inference.

The trade-off between accuracy and latency is particularly evident in Closed-loop Sense-Act Systems (Figure 1). Consider the example of a guided missile interception system. The Sensing module records the state of the environment, namely the current positions of the target intruder missile and the interceptor missile. Within the sensing module, there may be additional sub-methods (Perception, Inference) that assist in deriving the final state information from the raw sensor data. This state information is leveraged by the Control module to estimate the required trajectory of the interceptor missile and is subsequently translated into an action, specifically updated linear and angular velocities, by the Actuation module.

Research reported in this paper was sponsored in part by the DEVCOM Army Research Laboratory.

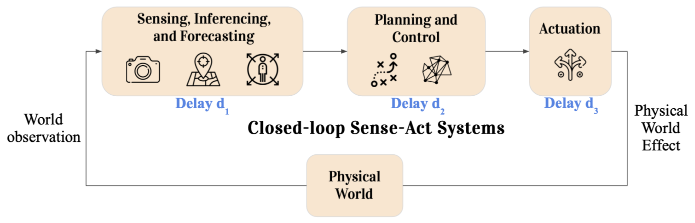


Fig. 1: Visualization of the Sense-Act system with module delays d_1, d_2, and d_3

Each module within the Closed-loop Sense-Act System incurs its own computation delay (d_1, d_2, and d_3), which, when integrated with the system design, governs the application performance in terms of the accuracy-latency trade-off. In the majority of instances, [13] [14], the system design is tailored to account for the worst-case delays, wherein sensing and actuation occur at fixed intervals with the interval being greater than the worst-case cumulative of d_1, d_2, and d_3. Alternatively, some cases, [3] [15] [16], configure the system design such that sensing occurs at fixed worst-case intervals, while actuation is executed as soon as the action is ready.

Prior research has examined the impact of delays on various components of the Sense-Act systems. For instance, Sela et al. [1] investigated the influence of model completion times on the performance of the perception module. The authors illustrate that by the time the more complex, accurate models finish processing, the ground truth has changed significantly. Furthermore, if the application performance is determined by accurately representing the state of the world at the time of inference, then in certain scenarios it is more beneficial to choose a simpler, less accurate model that finishes faster.

Sandha et al. [2] demonstrate that systems cognizant of the overall delays within the loop yield superior application performance. For complex environmental settings, their approach of training an end-to-end perception to control policy showed 23.8% performance improvement when compared with the state of the art techniques.

However, these works focus on Sense-Act systems operating on a single platform, typically a resource-constrained device. As the models employed within these systems evolve, they become increasingly computationally complex. For instance, TinyYolov4 (a lightweight object detector for embedded devices) offers higher accuracy at the cost of 66% higher inference latency than its predecessor, TinyYolo3x [17]. To optimize the utilization of these advances, we posit the need to leverage other platforms such as Edge and Cloud in addition to the resource-constrained devices.

In this work, we focus on the Control module in the Sense-Act system and investigate the adaptation of control placement to optimize application performance in distributed settings. We assess the impact of different controllers and their placement on application performance under varying delays. Using a representative Vehicle Following example where the ego vehicle trails a target vehicle on a specified trajectory, we demonstrate that systems with immediate execution of action outperform those designed for worst-case delays. Further, we highlight the benefit of a distributed system architecture on application performance by implementing an Adaptive Control Placement strategy.

This paper is structured as follows: We first introduce the underlying principles of our controllers of choice in Section II and outline the system design and architecture in Section III. Section IV provides an overview of the simulation process, including the setup and performance metrics, as well as elaborates on simulation results. Finally, we summarize our findings in Section V and discuss potential future directions in Section VI.

II. Background

In this section, we provide a brief overview of the two control methodologies we employ, namely PID (Proportional-Integral-Derivative) and MPC (Model Predictive Control).

A. Proportional-Integral-Derivative Control (PID)

The PID controller is the most prevalent form of feedback control algorithm [4], a popularity that can be attributed to its simplicity and ease of use. The mathematical expression of the PID Controller is given as:

$$u(t) = K_{\mathrm{P}}e(t) + K_{\mathrm{I}}\int e(t)dt + K_{\mathrm{D}}\frac{de(t)}{dt} \tag{1}$$

Here, the generated control signal *u(t)* is a summation of three components: Proportional, Integral and Derivative with gains K_P, K_I, and K_D respectively. Each term produces a control signal with respect to the error *e(t)* which is calculated as the difference between the desired set point *r(t)* and the measured state *y(t)*, i.e. *e(t) = r(t) - y(t)*.

As the name suggests, the proportional term produces a control signal proportional to error. The integral term corrects for any steady-state errors while the derivative term uses the rate of change of error to anticipate its future behavior. The gain of each term (K_P, K_I, and K_D) can fine-tuned to achieve desired controller performance. However, this fine-tuning can be challenging [6] and the performance of the controller degrades with significant time delays [5].

B. Model Predictive Control (MPC)

MPC is an advanced control strategy that computes control inputs by solving an optimization problem at each time step. To understand MPC, let us consider the example of a vehicle with the state space representation defined by the following set of differential equations:

$$\begin{aligned} x'(t) &= v(t) * sin\phi \\ y'(t) &= v(t) * cos\phi \\ v'(t) &= a(t) \\ \phi'(t) &= v(t) * tan\delta/L \end{aligned} \tag{2}$$

where $x(t)$ and $y(t)$ denote the current x,y coordinates, $v(t)$ and $\phi(t)$ give the current linear velocity and orientation of the vehicle. Additionally, $a(t)$, δ, and L represent the acceleration, steering angle and the vehicle wheel base (distance between the front and rear wheels) respectively.

The Model Predictive Controller utilizes the mathematical model (2) and the state of the system (in this case, the vehicle) denoted by $[x, y, v, \phi]$ to forecast its future behavior over a finite horizon T and generates the control output $[a, \delta]$ to minimize a cost function over this horizon. This approach allows MPC to handle multi-variable systems and constraints effectively. However, the requirement to solve an optimization problem at each time step makes MPC computationally expensive, and the selection of the prediction horizon, cost function weights, and constraints requires careful tuning to achieve the desired controller performance [7].

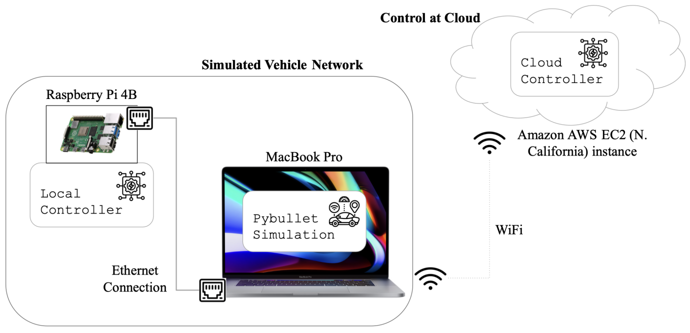


Fig. 2: System Setup

III. System Design

In this section, we first describe the system architecture for controller type and location. We then elaborate on our implementation of this architecture.

To illustrate placement of control in a distributed setting, we employ two locations of executing the control algorithm: Local and Cloud. Local control refers to on-device controller which may be connected to the device's sensors using a very low-latency link and has limited computational resources by the virtue of being hosted on the resource-constrained device. Cloud control can be thought of as a remote controller, equipped with larger computational resources but connected to the device via a high-latency link.

We implement this architecture (Figure 2) for a Vehicle Following application by simulating a F1/10 racecar [8] and its environment in the Pybullet simulator [9]. The simulator is operated on a MacBook Pro [10] which is connected to the local controller i.e. Raspberry Pi 4B [11] via Ethernet. The simulator together with the local controller is representative of the vehicular network. The cloud controller is hosted on an Amazon AWS EC2 (N. California) [12] instance and is connected to the vehicular network via Wi-Fi. The capabilities of both control locations are listed in Table I.

	Local	**Cloud**
Computation	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz; 4GB RAM	t2.micro with Intel Xeon Scalable Processors @ 3.3 GHz; 1 GiB RAM
Communication	Ethernet	Wi-Fi

TABLE I: System Capabilities of Control Locations

IV. SIMULATIONS

In this section, we first outline the application performance metrics and the simulation scenarios. Subsequently, we examine the impact of various combinations of controller type and placement on the performance. Finally, we introduce the adaptive control strategy and assess its performance in comparison to that of individual controllers.

A. Application Performance Metrics

We evaluate the application performance using two metrics:

- $Time\ to\ Complete\ Trajectory\ (TTC)$: The time the vehicle takes to reach $Goal$ destination from $Start$ location.
- $Tracking\ Error$: The average deviation of followed trajectory from expected trajectory. It is calculated by sampling the trajectories at fixed intervals and measuring the deviation at each sample.

For an ideal controller, we expect a low TTC and minimal tracking error.

B. Simulation Scenarios

To study the impact of delays on application performance, we evaluate the system under a variety of actuation deadlines:

- $As\ soon\ as\ possible\ (ASAP)$: There is no actuation deadline. Action is applied as soon as it reaches the device (in our case, simulated vehicle).
- $Static\ (worst-case)\ deadline$: Action is applied at a specified worst-case actuation deadline ($\Delta_a = \Delta$) where the interval between two deadlines is greater than worst-case delays within the loop.
- $Static\ (high)\ deadline$: Here, the actuation deadline is slightly less than that of Static (worst-case) ($\Delta_a = 0.9\Delta$)
- $Static\ (moderate)\ deadline$: Here, the actuation deadline is slightly less than that of Static (high) ($\Delta_a = 0.8\Delta$)
- $Static\ (low)\ deadline$: This is the case of very strict deadlines where the actuation deadline is close to the summation of delays in the loop ($\Delta_a = 0.7\Delta$)

In each case, sensing happens at fixed sensing intervals ($\Delta_s = \Delta$) which are greater than worst-case delays within the Sense-Act loop of our simulation.

C. Controller Type and Location Compatibility

To identify the most optimal combination of controller type and location, we first analyze the total latency from sensing to effect, i.e. one cycle of the Sense-Act loop. Here, the ASAP (As Soon As Possible) deadline is used wherein the action is implemented immediately upon becoming available. Table II provides a detailed breakdown of total latency for each combination of controller type and location.

We observe that the local control location offers approximately 8 times lower communication latency than cloud control location. Futher, controllers when hosted on the cloud benefit from a significant computational speed-up. As shown in Section II, the PID controller is less complex than the MPC, and therefore, receives a more substantial computational boost. Additionally, we note that the resource-constrained nature of the local control location has an adverse effect on MPC's computational latency, rendering it infeasible for systems with sampling period less than 0.6 seconds.

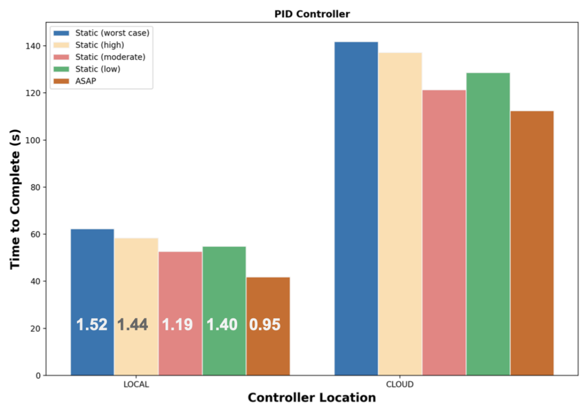
Controller Loc./Type	**Communication Latency (ms)**	**Computation Latency (ms)**	**Total Latency (ms)**
Local PID	(9.54, 44.9)	(56.8, 96.89)	(69.34, 111.24)
Local MPC	(7.92, 14.8)	(580.5, 773.21)	(580.62, 773.7)
Cloud PID	(81.73, 263.9)	(0.25, 0.34)	(81.98, 264.18)
Cloud MPC	(50.45, 117.53)	(157.6, 185.03)	(208.12, 269.8)

TABLE II: Breakdown of total latency with (average, worst-case) metrics in ms

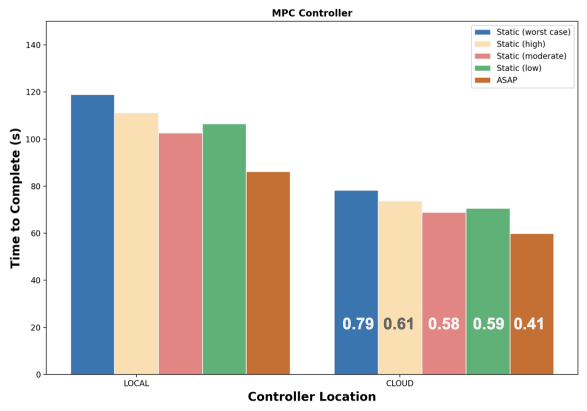
D. Controller Performance

We evaluate the performance of PID and MPC algorithms when hosted at local and cloud control locations using the metrics we identified in Section IV-A. Figure **??** shows the TTC to as function of controller type and location for each deadline as outlined in Section IV-B. Average deviation is also noted on the bars for Local PID and Cloud MPC which are the best performing combinations.

Executing PID at the Cloud results in the longest completion time for each deadline because the effect of lower accuracy, leading to deviation from the trajectory, is magnified by the high round-trip communication latency. Additionally, as highlighted in Section IV-C, the implementation of MPC at the Local control location is computationally infeasible. Although PID at Local yields the lowest Time-To-Complete (TTC), it



(a) Performance of PID controller at Local and Cloud locations. Local PID finishes faster than Cloud PID.



(b) Performance of MPC controller at Local and Cloud locations. Cloud MPC finishes faster than Local MPC.

Fig. 3: Application performance of PID and MPC controllers are Local and Cloud placement locations under ASAP deadlines. Cloud MPC has less tracking error (average deviation from expected trajectory) than Local PID controller.

incurs a higher tracking error compared to MPC at Cloud, which, despite having a higher TTC, yields more precise trajectory tracking.

Furthermore, we observe that Static (worst-case) deadlines offer the poorest performance for both time-to-complete (TTC) and tracking error. This is attributed to the vehicle having a prolonged wait time for the actuation deadline, resulting in a significant deviation from the expected trajectory. This deviation is amplified if the actuation deadline is missed due to computational or network latency. As the deadlines decrease (Static (high) and Static (moderate) cases), we notice a decrease in TTC and average deviation because the actions are implemented more promptly, affording the vehicle less time to deviate from the path. However, in the case of strict deadlines (Static (low) case), we see a deterioration in performance since the likelihood of missing deadlines increases, subsequently leading to increased deviation. It is, however, rapidly rectified at the subsequent actuation deadline. Finally, the absence of additionally delays in ASAP deadline scenario enables it achieve the best performance.

E. Adaptive Control Placement Performance

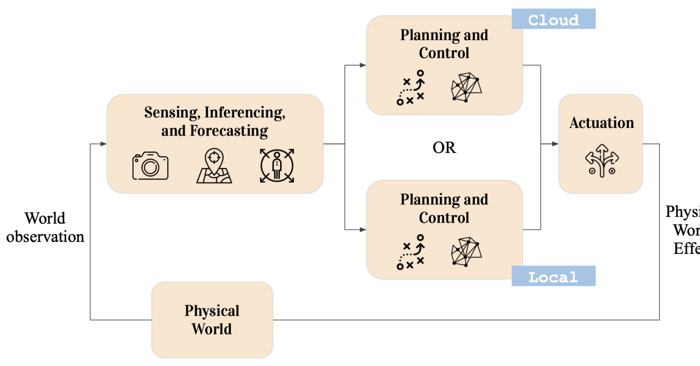
The findings from the preceding sections collectively suggest that PID, when hosted on the device, and MPC, when hosted on the Cloud, yield the best performance. We leverage this by designing a distributed system with both Local and Cloud control locations and employ an Adaptive Control Placement (ACP) strategy as shown in Figure 4(a). MPC, being a more complex control methodology, is hosted on computationally faster but higher communication latency control location, i.e. the Cloud, while the simpler PID controller is hosted on the computationally slower Local control location with low communication latency. Although preference is given to the more accurate MPC controller, we switch to less accurate but always available PID controller in the event of high network delays.

Figure 4(b) illustrates the application performance of the Adaptive Control Placement strategy compared with the best-performing controllers for each location, i.e., PID at Local and MPC at Cloud. ACP outperforms both the individual controllers in terms of time-to-complete (TTC) across all deadline scenarios. This can be ascribed to the absence of deadline misses in the case of adaptive control placement. Specifically, during substantial network delays, the PID controller assumes control, albeit with a less accurate control signal, to prevent the vehicle from significantly deviating from the trajectory. While this maneuver incurs a tracking error cost, it contributes to reducing the overall trajectory completion time.

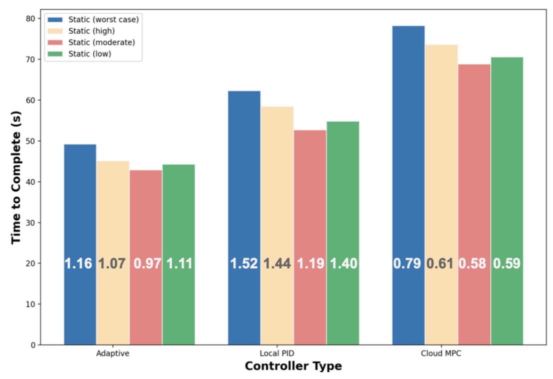
On the other hand, the tracking error of the ACP lies between that of the individual controllers since a combination of control signals from the low-accuracy PID and high-accuracy MPC controllers are utilized. Additionally, as expected, Static (worst-case) deadlines have the highest time-to-complete and it canonically decreases as the deadlines becomes shorter. However, for very strict deadlines (Static (low) case) we observe a slight increase in TTC due to increase in the number of invocations of the PID controller. This leads to greater deviation from expected trajectory and results in increased time to complete as well.

V. Conclusion

In this paper, we examined the influence of delays within the Closed-Loop Sense-Act Systems on application performance. While prior research in this domain focused on optimizing performance on a single platform, our work shows that better optimization is achieved by harnessing a distributed architecture. With the wide accessibility and adoption of computing tiers such as Edge and Cloud, the importance of challenges posed by computation and communication delays increases. Focusing on the Control sub-module within the Sense-Act system, we demonstrate that variations in delays leads to corresponding variations in application performance. Moreover,



(a)



(b)

Fig. 4: Architecture and performance of Adaptive Control Placement strategy. (a) Visualization of the distributed architecture of the Adaptive Control Placement strategy (b) ACP outperforms Local PID and Cloud MPC in terms of time-to-complete. Tracking error of ACP (denoted by bar labels) lies in between that of PID and MPC.

we present an Adaptive Control Placement strategy that optimizes overall application performance by switching between a lower-accuracy local controller and a higher-accuracy cloud controller based on network dynamics.

VI. Discussion and Future Work

This work focused on analyzing the effect of delays on the closed-loop version of Sense-Act Systems (for example, a guided missile interception system) where the sensing and actuation happen in a repetitive manner until the application objective is fulfilled. However, as we briefly discuss in Section I, there exist other variants of the Sense-Act systems such as radar-based tracking (situational awareness), and intruder interception (open-loop sense-act system) with different system architectures and objectives. A potential avenue for future research could be extending our delay impact analysis to such systems thereby facilitating a more comprehensive and generalized understanding of the relationship of delay with Sense-Act systems.

Additionally, the most prominent delays arise from complex sensing, inference and perception algorithms. Although we model these delays as a variety of deadlines in our work, integrating these algorithms with Adaptive Control Placement strategy would lead to a more refined overall system architecture, more closely aligned to real-world deployments. It would be valuable to study the application performance within complex, multi-modal Sense-Act systems and evaluate the generalizability of our findings.

Acknowledgment

The research reported in this paper was sponsored in part by the DEVCOM Army Research Laboratory under cooperative agreement W911NF1720196, and by the National Science Foundation under award # CNS-2211301. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

References

[1] Sela, Gur-Eyal, Ionel Gog, Justin Wong, Kumar Krishna Agrawal, Xiangxi Mo, Sukrit Kalra, Peter Schafhalter, et al. ”Context-aware streaming perception in dynamic environments.” In European Conference on Computer Vision, pp. 621-638. Cham: Springer Nature Switzerland, 2022.

[2] Sandha, Sandeep Singh, Bharathan Balaji, Luis Garcia, and Mani Srivastava. ”Eagle: End-to-end Deep Reinforcement Learning based Autonomous Control of PTZ Cameras.” arXiv preprint arXiv:2304.04356 (2023).

[3] Ma, Yehan, Chenyang Lu, Bruno Sinopoli, and Shen Zeng. ”Exploring edge computing for multitier industrial control.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39, no. 11 (2020): 3506-3518.

[4] Padula, Fabrizio, and Antonio Visioli. Advances in robust fractional control. Cham: Springer International Publishing, 2015.

[5] Atherton, Derek P., and Somanath Majhi. ”Limitations of PID controllers.” In Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251), vol. 6, pp. 3843-3847. IEEE, 1999.

[6] Cominos, P., and N. Munro. ”PID controllers: recent tuning methods and design to specification.” IEE Proceedings-Control Theory and Applications 149, no. 1 (2002): 46-53.

[7] Darby, Mark L., and Michael Nikolaou. ”MPC: Current practice and challenges.” Control Engineering Practice 20, no. 4 (2012): 328-342.

[8] O’Kelly, Matthew, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam et al. ”F1/10: An open-source autonomous cyber-physical platform.” arXiv preprint arXiv:1901.08567 (2019).

[9] E. Coumans and Y. Bai, ”Pybullet a python module for physics simulation for games robotics and machine learning”, 2016–2019.

[10] MacBook Pro (M1 2021), https://support.apple.com/kb/SP854

[11] Raspberry Pi 4B specifications, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications

[12] Amazon AWS EC2, https://aws.amazon.com/ec2/instance-types/t2/

[13] Ma, Yehan, Cailian Chen, Shen Zeng, Xinping Guan, and Chenyang Lu. ”Data-driven Edge Offloading for Wireless Control Systems.” IEEE Internet of Things Journal (2023).

[14] Liu, Liangkai, Zheng Dong, Yanzhi Wang, and Weisong Shi. ”Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles.” In 2022 IEEE Real-Time Systems Symposium (RTSS), pp. 305-317. IEEE, 2022.

[15] Xu, Hansong, Jun Wu, Jianhua Li, and Xi Lin. ”Deep-reinforcement-learning-based cybertwin architecture for 6G IIoT: An integrated design of control, communication, and computing.” IEEE Internet of Things Journal 8, no. 22 (2021): 16337-16348.

[16] Yi, Saehanseul, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. ”Energy-Efficient adaptive system reconfiguration for dynamic deadlines in autonomous driving.” In 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC), pp. 96-104. IEEE, 2021.

[17] Yao, ZhengBai, Will Douglas, Simon O'Keeffe, and Rudi Villing. "Faster yolo-lite: Faster object detection on robot and edge devices." In Robot World Cup, pp. 226-237. Cham: Springer International Publishing, 2021.