A Study of Real-time Computer Vision Tasks in 5G-enhanced Environment

Roman Juránek[1], Petr Klepárník[1], Michal Kapinus[2], Petr Dobeš[2] and Pavel Smrž[1,2]
[1] Cognitechna s.r.o., Brno, CZ
[2] Brno University of Technology, Brno, CZ
Email: r.juranek@cognitechna.cz

***Abstract*—Computer vision is an integral part of many robotic applications which are often executed on limited hardware. In this paper, we study applications which execute the vision tasks remotely in a cloud or edge environment instead of local execution. Such a structure of applications (called Network Application within the 5G-ERA project) helps to overcome many limitations of robotic platforms stemming from less powerful hardware. The one requirement is that each robot must have a sufficient network connection. In the future, we believe, this will be solved easily using new generations of networks which will offer low latencies and high bandwidth for connected clients. In this paper, we discuss the particular case of vision tasks and show their properties in a use case of visual collision warning and train detection systems for autonomous robots. In particular, we compare local execution with the execution in a cloud-based system using the automatically deployed container. The Network Application along with experimental data is available as an open source on GitHub.**

I. Introduction

Robotic platforms nowadays employ vision applications like object detection and tracking, environment mapping and localization, image recognition, etc. This is especially prominent e.g. in autonomous driving, where advanced and robust algorithms must be used in order to ensure reliability and safety. Such algorithms come with a price which includes hardware and energy demands and, ultimately, increased cost of the system. However, the computing power of robots is often limited, especially in cases where robots must be kept simple and cheap, the reason being that many units must be built and deployed (e.g. drones or service robots in medical care). Such robots, even when equipped with a GPU, cannot often afford the execution of advanced algorithms. Moreover, their performance cannot compare to massive computing power, which can be easily deployed in the cloud, not to mention energy demands which can be satisfied easier in a data center than on an autonomous robot running on battery.

The advantages of using cloud computing for autonomous robots include – almost unlimited computing power, low energy requirements for the robot, a simpler robotic platform, and simple updates of algorithms in the cloud. The one strong requirement is the connectivity. Almost unlimited computing power – limited only by the availability of computing nodes which can be added or removed transparently. Moreover, resources like memory or GPU can be allocated with respect to the needs of the robot. Low energy requirements – Due to offloading, the energy consumption can be "moved" to the data center, not draining the battery with expensive local computations. Simpler robotic platform - the hardware can be constrained to sensory inputs, necessary local processing and connectivity. Simple updates of algorithms – A new version of an algorithm used by the robot can be simply deployed in a container; no changes in the robot itself are required since the whole process is managed centrally.

The connectivity required for cloud application deployment can be satisfied by ordinary WiFi (and its recent specifications [1]), which can be used on local networks covering buildings and small areas. For autonomous robots, e.g. in transportation, mobile networks offering 4G, LTE or 5G connection have to be used. Especially 5G seems promising for such applications since it specifies network slicing, which can provide extremely low latency or transfers of massive data. In the future, the 5G network (and its potential successors) has the potential to become the standard not only for IoT communication but for robotic communication as a whole.

In the 5G-ERA[1] [2–4] project, a framework for the management and deployment of robotic applications is being developed. In this framework, a robot makes a request specifying the task it wants to use (like detection, mapping, etc.). The placement and life-cycle of deployed applications are managed by the Middleware [2] – a central service of the 5G-ERA framework. The Middleware decides where the application is executed – either the edge system near the robot for extremely low latency, or a remote data center for massive computations with the lower required bandwidth, and connects the robot to the deployed application. The important point is that from the robot's perspective, this process is transparent, performed by the Middleware in the background. Once the application is deployed, the robot starts sending inputs to the application and collects results which it uses for its own ends.

In this article, we employ the framework from the 5G-ERA project and implement applications for transportation within it – a vision-based collision warning system and train detection. Such a system requires instant feedback (e.g. low processing latency) and have to employ advanced algorithms based on deep learning – robust object detection and tracking. So far, such systems were usually implemented directly in robotic platforms using other sensory inputs than cameras (e.g. radar, lidar, and IMU). The main properties of our algorithms are as follows. They uses only a camera as the input. They are executed completely in the cloud managed by 5G-ERA Middleware, and they can be reused by multiple robots. The algorithms and the packages using the 5G-ERA framework

[1] 5G-Enhanced Robot Autonomy https://5g-era.eu
[2] https://github.com/5G-ERA/middleware

were made available as an open source on GitHub[3,4].

II. Network application framework

The 5G-ERA framework defines a simple protocol for connecting the robot and the Network Application. It combines the simplicity of HTTP requests with the versatility of WebSockets to deliver a bidirectional communication channel. This protocol enables various types of deployments. The network application can be deployed locally on the robot, remotely on an edge device, or in the cloud, using the same protocol and client library. Combined with the 5G-ERA Middleware, this framework provides high scalability and versatility in network application deployment across multi-domain and multi-administration environments.

The protocol requires the robot first to call the `/register` endpoint. As a result, the network application creates a session for the client (robot). After that, the client initiates the WebSocket communication to establish the bidirectional data transfer channel. The current implementation supports three image transport channels:

- JPEG-compressed image sent over HTTP endpoint
- JPEG-compressed and base64-encoded image sent over WebSockets
- h.264 video stream sent using GStreamer pipeline

Each channel has different demands for computing power (encoding h.264 video stream is more demanding than JPEG compression) or network bandwidth (individual JPEGs, encoded in base64, will be much bigger than h.264 stream). Therefore, each transport channel suits different use cases.

The framework consists of two main parts, the python client library `era-5g-client`[5], and the interface library `era-5g-interface`[6], which serves as a base for the implementation of network applications. Alongside these two libraries, a reference implementation of a network application is provided[7].

III. Network applications for transportation

As example applications for testing the framework, we consider two vision tasks that are part of the use case of the 5G-ERA project within the domain of transportation. Both applications are required for autonomous driving of an advanced delivery robot. The first task is train detection – as the robots are deployed in an industrial area with many train tracks without a fixed train schedule, one problem to be solved is detecting trains on crossings. The second task is collision warning system – another problem is detecting obstacles and dangerous objects in the robot route. Usually, these are cars and people moving freely in the area. Both applications require complex vision algorithms and instant feedback, and thus they are often considered as candidates for local execution, which also requires powerful hardware. We analyze the properties of these applications when executed both locally and remotely as Network Applications under a variety of conditions.

[3]https://github.com/5G-ERA/CollisionWarningService
[4]https://github.com/5G-ERA/TrainDetectorService
[5]https://github.com/5G-ERA/era-5g-client
[6]https://github.com/5G-ERA/era-5g-interface
[7]https://github.com/5G-ERA/Reference-NetApp/

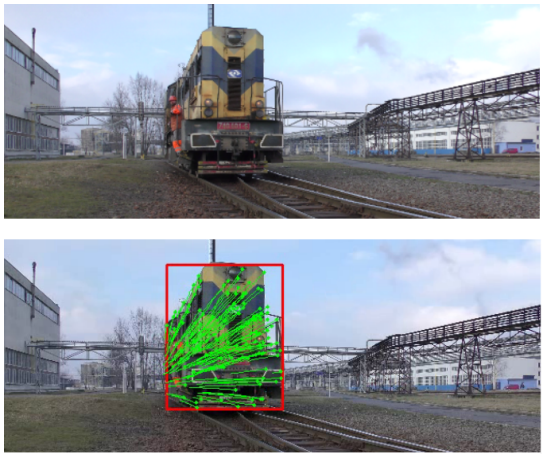

Fig. 1. Visualization of the train detector application. Train instances are detected with YOLO network and grouped together across the time domain using SORT tracker. Subsequently, keypoints are detected and tracked for each instance, in order to determine its movement. Green dots represent the keypoints, while green lines show their tracks over time. The bounding box of the train and the resultant information about its motion is then sent to the robot.

A. Train Detector

The train detector application is a service for obtaining information about possible train movement in a video stream. It is intended to be used on robotic vehicle that operates inside an industrial area with a railway branch line (industrial spur).

The robot can utilize this service each time it is about to cross railway tracks on its route. Firstly, the robot must stop before the tracks, in order to allow the motion of the train to be distinguished from the movement of the video stream that is induced by the robot itself. Subsequently, train detector service is called and it searches for any trains in the camera view, and evaluates whether they are still or moving. This information can then be used by the robot to decide when it is safe to proceed with crossing the track. Implementation of the train detector service encompasses several processing steps. YOLO object detector from MMDetection Toolbox [5] is first used to find train instances in a video frame. SORT tracker [6] is then employed to track the movement of the individual objects in the video. Subsequently, optical flow is estimated for each object. The final information about whether the train is moving is then established based on the magnitude of the optical flow. Visualization of the detection process is shown in Figure 1. The structure the service follows the Network Application framework described in Section II and it is made available on GitHub[8].

B. Visual collision warning

Collision warning works with a calibrated front-facing camera. It detects dangerous objects in the robot's path (e.g. persons, other cars, etc.), estimates their relative location and motion, and predicts their movement on the road plane, see Figure 2. The predicted movement is analyzed with respect to the defined safety zones, and alarm signals are issued in case of dangerous behavior.

[8]https://github.com/5G-ERA/TrainDetectorService

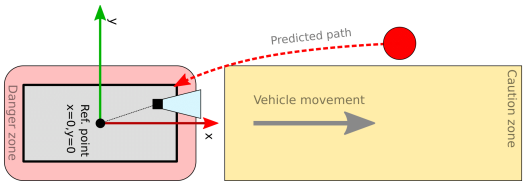

Fig. 2. Conceptual view of collision warning. The camera captures objects (the red dot) in the robot's path. The Network Application predicts the path of objects relative to *caution* or *danger* zones, and issue signals.

Camera calibration is defined by the intrinsic matrix – projection parameters, and extrinsic matrix – location and rotation of the camera w.r.t. the robot coordinate frame. We estimate intrinsics simply using OpenCV [7] tools, extrinsics can be estimated from one known vanishing point in the direction of the robot movement and known horizon line. The input image from the camera is rectified to get the ideal pinhole projection. For the detection of dangerous objects we use pre-trained YOLO detector [8, 9]. We detect common dangerous objects – vehicles, persons and animals. And we track the objects with the SORT tracker [6]. Each object is then represented by its identity, class and bounding box in the image. We leverage camera calibration information in order to estimate the 3D location of the objects w.r.t. the vehicle, the assumption here is that the objects stand on the road plane. The representation of each object is enriched with its location in the world by projecting the center of the bottom edge of the bounding box to the road plane ($z = 0$). Each object is tracked in the world coordinate system using a 2D Kalman filter [10] with a constant acceleration model (omitting z coordinate). The Kalman state is used for prediction over a short time span (about 1 s) to get the object location in the near future. We define two zones – *caution zone* and *danger zone*. The caution zone is defined by a polygon in front of the robot, generally speaking, where the robot will be in the near future. The danger zone is a tight polygon around the robot – i.e. anything penetrating the danger zone, collides with the robot. The movement of each object is analyzed, and four states are recognized:

- SAFE - Object is outside caution zone.
- CAUTION - Object will enter caution zone.
- WARNING - Object is inside cation zone.
- DANGER - Object will collide with the danger zone.

The Figure 3 shows a example visualization and signals issued by the algorithm. The robot must react appropriately to the obtained signals. The simplest case is to start breaking. The more advanced strategy involves path planning – this is, however, left to the developers using the Network Application which provides the signals.

The architecture of the Network application is visualized in Figure 4. The Python package is divided into `core`, `service` and `client` modules. The `core` module handles object detection, collision prediction and visualization. As an input to the core algorithm, it is necessary to specify the video source (file or video stream) and the configuration files for setting the algorithm parameters. The `service` module is used on a server that can run outside the robot environment and uses

Fig. 3. Visualization of the collision warning. The bottom left picture shows rectified image with predicted object track. The top bar shows the signals issued by the system.

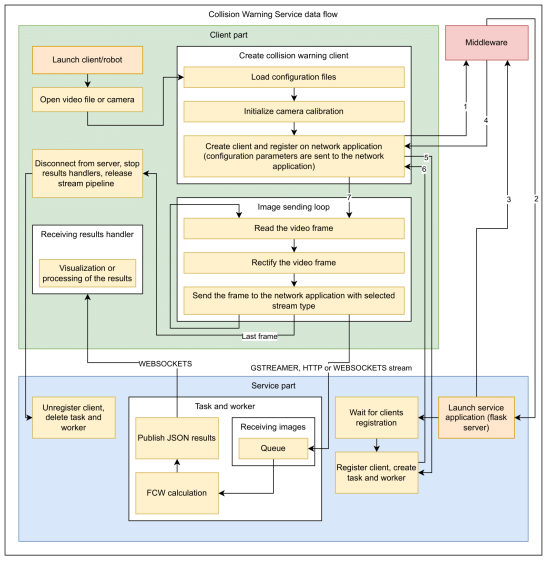

Fig. 4. Data flow of Collision warning service application.

both the `core` module for computation itself and the 5G-ERA network application framework which realizes the remote service interface. The `client` module contains two examples of clients (robots) for pure python (`client_python`) and for ROS2 (`client_ros2`). The `era-5g-client` package handles the connection to the Middleware, registering the client on the server (in the network application) and communication between the remote and client counterparts. The data flow of the Network Application is visualized in Figure 4.

IV. Evaluation of the Network Applications

We evaluated both Network applications considering the power requirements, observed latency and CPU, GPU, RAM, and video RAM usage under various conditions. The metrics were measured on three different execution platforms:

- *Robot* – Intel Core i5-6500 CPU, 4 cores @ 3.20GHz with 32 GB RAM and Nvidia GeForce GTX 1050 Ti with 4 GB RAM, running Ubuntu 22.04 operating system

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			FPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot	82	81	79	86	83	80	11	11	11	9	9	9	18	18	18	124	130	131	10.0	10.0	10.0
Eth/Edge	49	48	48	31	31	31	4	4	4	0	0	0	0	0	0	121	126	134	9.9	10.1	10.0
Wi-Fi/Edge	38	37	48	25	33	30	5	4	4	0	0	0	2	2	2	124	132	142	9.6	10.0	10.1
5G/Cloud	37	48	48	18	31	31	4	4	4	0	0	0	0	0	0	207	193	321	6.0	9.9	10.0
Eth/Cloud	37	37	37	28	23	21	4	4	4	0	0	0	0	0	0	147	168	283	10.0	10.0	10.0
Wi-Fi/Cloud																					

TABLE I. THE RESULTS FOR TRAIN DETECTOR SERVICE FOR 10 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. H, W, AND G STAND FOR HTTP, WEBSOCKETS AND GSTREAMER RESPECTIVELY.

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			FPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot	85	85	86	96	94	94	11	11	11	17	17	17	18	18	18	83	74	69	20.4	21.4	23.5
Eth/Edge	51	51	50	39	38	35	4	4	4	0	0	0	0	0	0	61	66	74	23.5	23.8	24.6
Wi-Fi/Edge	39	39	39	33	30	51	5	4	5	0	0	0	2	2	2	68	83	190	18.3	22.9	23.2
5G/Cloud	47	40	49	29	32	35	4	4	4	0	0	0	0	0	0	197	175	258	5.9	18.7	21.0
Eth/Cloud	38	40	38	32	30	54	4	4	4	0	0	0	0	0	0	128	145	195	13.7	19.0	21.4
Wi-Fi/Cloud	49	50	50	34	38	35	4	4	4	0	0	0	1	1	1	122	155	174	12.3	16.3	20.9

TABLE II. THE RESULTS FOR TRAIN DETECTOR SERVICE FOR 25 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. H, W, AND G STAND FOR HTTP, WEBSOCKETS AND GSTREAMER RESPECTIVELY.

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			FPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot	98	97	97	92	90	92	11	11	10	32	33	33	21	21	21	77	78	88	10.0	10.0	10.0
Eth/Edge	57	57	56	33	33	33	4	4	4	0	0	0	0	0	0	92	97	80	10.0	10.0	10.0
Wi-Fi/Edge	58	59	58	53	55	55	5	4	5	0	0	0	2	2	2	118	106	286	8.1	9.6	10.0
5G/Cloud	58	58	58	53	56	56	4	4	4	0	0	0	0	0	0	204	140	206	5.2	9.7	10.0
Eth/Cloud	56	55	55	34	33	34	4	4	4	0	0	0	0	0	0	127	110	88	8.0	10.0	10.0
Wi-Fi/Cloud	58	58	58	55	55	55	4	4	4	0	0	0	1	1	0	144	108	87	6.5	9.8	10.0

TABLE III. THE RESULTS FOR COLLISION WARNING APPLICATION FOR 10 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. H, W, AND G STAND FOR HTTP, WEBSOCKETS AND GSTREAMER RESPECTIVELY.

- *Edge* and *Cloud* – AMD Ryzen 7 1700X, 8 cores @ 3,4 GHz with 32 GB RAM and Nvidia GeForce GTX 1050 Ti with 4 GB RAM, running Ubuntu 20.04 operating system

The robot and the edge computers were two different computers with similar specifications. The cloud platform was a computer with exactly the same specifications as the edge platform but placed in remote network. As networking media, we consider Ethernet and WiFi providing access to local networks and the internet, and 5G connection is available through the commercial operator. For 5G access, we use a Netgear MR5200 router connected to the robotic platform. It should be noted that the only combinations enabling truly autonomous robot operation are *Robot* (local) and *5G/Cloud* variants since others enable small local area deployment only. We do not consider the *5G/Edge* variant here since we use services of the commercial 5G operator and do not have access to their infrastructure.

The results, as shown in Table I to Table IV, indicate that offloading makes sense for lowering power consumption on the robot. The power is lower in all cases when the algorithm does not run on the robot, even though there is additional overhead associated with encoding individual images or the video stream. Latency is slightly higher for Ethernet and WiFi connections, which can be considered acceptable. However, it is approximately double for a 5G connection, which we attribute to the usage of a commercial network. With a private network, it should be possible to obtain substantially better results. The results show that HTTP transport has worse performace when the network application is offloaded compared to other types of transport. With an application in the cloud, CPU usage is lower. RAM usage is lower in all cases. When running non-locally, the GPU is not used; therefore, its usage is zero.

V. CONCLUSION

We presented a use case of two computer vision applications for autonomous delivery robots – the Train Detector and Collision Warning System. Both applications are impor-

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			FPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot	134	135	134	99	99	99	11	11	10	59	62	65	21	21	21	85	91	72	21.1	22.7	24.1
Eth/Edge	57	57	57	35	39	39	4	4	4	0	0	0	0	0	0	91	83	61	14.6	23.1	23.4
Wi-Fi/Edge	58	59	59	55	59	59	5	5	5	0	0	0	2	2	2	111	96	71	8.9	19.5	22.0
5G/Cloud	57	59	59	53	59	59	4	4	4	0	0	0	0	0	0	215	146	244	5.1	20.4	24.3
Eth/Cloud	55	56	56	33	39	39	4	4	4	0	0	0	0	0	0	128	99	115	7.5	22.9	23.5
Wi-Fi/Cloud	58	59	59	54	59	59	4	4	4	0	0	0	1	1	0	146	107	122	6.6	18.9	23.0

TABLE IV. THE RESULTS FOR COLLISION WARNING APPLICATION FOR 25 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. H, W, AND G STAND FOR HTTP, WEBSOCKETS AND GSTREAMER RESPECTIVELY.

tant for the safety of the robot and its correct functioning. We experimented with the applications in different kinds of networking environments from simple local deployment to a complex automatically orchestrated deployment in the cloud using 5G technology and compared different tradeoffs between the type of execution and used networking media. Our experiment shows that the deployment of applications in the cloud environment increases latency and decreases power consumption of the robot. The difference is that in the "cloud" case, the robot does not have to contain potentially expensive hardware and serves simply as a source of sensory data with the expensive task offloaded to the cloud. The latency is not a critical limitation in the case of the Train Detector where the robot stops in front of the tracks, however, collision avoidance requires much lower latencies. This will be solved in future by exploitation of 5G network slicing features and deployment of Network Applications in *5G/Edge* mode by the 5G-ERA Middleware.

ACKNOWLEDGMENT

This paper was supported by the 5G-ERA project with funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 101016681.

REFERENCES

[1] E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi, "A tutorial on ieee 802.11ax high efficiency wlans," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 197–216, 2019.

[2] R. Qiu, D. Li, A. L. Ibáñez, Z. Xu, and R. L. Tarazón, "Intent-based deployment for robot applications in 5g-enabled non-public networks," 2023.

[3] M. Sophocleous, C. Lessi, Z. Xu, J. Špaňhel, R. Qiu, A. Lendinez, I. Chondroulis, and I. Belikaidis, "Ai-driven intent-based networking for 5g enhanced robot autonomy," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops: MHDW 2022, 5G-PINE 2022, AIBMG 2022, ML@ HC 2022, and AIBEI 2022, Hersonissos, Crete, Greece, June 17–20, 2022, Proceedings*. Springer, 2022, pp. 61–70.

[4] C. Lessi, G. Agapiou, M. Sophocleous, I. P. Chochliouros, R. Qiu, and S. Androulidakis, "The use of robotics in critical use cases: The 5g-era project solution," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops: MHDW 2022, 5G-PINE 2022, AIBMG 2022, ML@ HC 2022, and AIBEI 2022, Hersonissos, Crete, Greece, June 17–20, 2022, Proceedings*. Springer, 2022, pp. 148–155.

[5] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[6] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468.

[7] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[8] Ultralytics, "yolov5," http://github.com/ultralytics/yolov5, 2022.

[9] J. Wang, Y. Chen, M. Gao, and Z. Dong, "Improved yolov5 network for real-time multi-scale traffic sign detection," 2021.

[10] R. Labbe, "Kalman and bayesian filters in python," https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/, 2022.