IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 6 August 2023, accepted 24 August 2023, date of publication 31 August 2023, date of current version 18 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3310887

RESEARCH ARTICLE

Data-Driven MPC for a Fog-Cloud Platform With AI-Inferencing in Mobile-Robotics

DINSHA VINOD[1], DURGESH SINGH[2], AND P. S. SAIKRISHNA[1]
[1]Department of Electrical Engineering, Indian Institute of Technology Tirupati, Tirupati 517619, India
[2]Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai 600036, India

Corresponding author: Dinsha Vinod (ee18d501@iittp.ac.in)

This work was supported by the Science and Engineering Research Board (SERB) under Project SERB/F/8288/2020-21.

ABSTRACT One of the most appealing challenges, especially in mobile robotics, is the real-time inferencing of the vision data offloaded from a mobile robot for its enhanced navigation performance. Leveraging the benefits of an on-premise and an integrated fog-cloud computing platform for processing vision data is a viable solution. This paper addresses the mentioned challenge via a data-driven control approach for the development of an autonomic fog-cloud computing platform suitable for processing the offloaded vision data within a prescribed time bound called the service time. The approach comprises developing a data-driven linear parameter varying (LPV) framework for modeling and design of a model predictive controller (MPC) for the fog and the cloud platforms independently. A heuristic algorithm performs the distribution of the mobile robot vision data (MRVD) between the fog and the cloud platforms. The LPV model for the fog and the cloud platforms are developed by characterizing the MRVD as the variable frame rate and resolution of the objects (under active consideration in a given frame) acquired during navigation. We validate the developed theory for a mobile robot while navigating in the application environment such as the warehouse. The experimental results presented for object detection under service time bounds show the efficacy of the proposed approach.

INDEX TERMS Fog and cloud computing, model predictive control, robot vision, system identification.

I. INTRODUCTION

Autonomous mobile robotics is a rapidly expanding field of research owing to its prominent applications in diverse areas such as autonomous vehicles, personal assistance, space exploration, healthcare, security, and defense [1]. Autonomous mobile robots (AMR) can work in unpredictable environments with no operator assistance with human-like intelligence and ingenuity for independent decision-making. The onboard perception sensors of an AMR are used to enable its core functions such as localization and path planning by visualizing and analyzing the external environment.

The vision system of an AMR implements image processing and computer vision algorithms to emulate human vision [2]. In general, the sequence of image frames received from the vision system is subjected to image pre-processing steps like image restoration and image enhancement to remove the environmental effects such as noise, haze, and blur. The post-processing steps involve computer vision algorithms as a series of measures such as object detection, texture removal, and edge detection [3].

Traditional computer vision and image processing techniques are computationally intensive and therefore may not be feasible to execute them on distributed computing platforms [4]. Therefore, it is not possible to scale computing nodes on such systems to improve service time. Recent advances in deep learning and machine learning have shown that complex tasks can be performed efficiently and seamlessly on fog and cloud platforms. Deep learning frameworks like Tensorflow, Microsoft Cognitive Toolkit, Caffe2, Torch, Keras, Apache MXNet, and Theano work well on a distributed computing platform [5], [6].

Object detection is a technique to locate and identify an object in an image or video frame. This technique involves pre-trained or custom models with train and test data sets of images. The processing time required for object detection

The associate editor coordinating the review of this manuscript and approving it for publication was Dong Shen.

depends on image resolution and frame rate. Therefore, we characterize the mobile robot vision data (MRVD) as the variable frame rate and resolution of the objects (under active consideration in a given frame). Drawing an inference from MRVD is a computationally intensive and power-hungry task that affects the mobile robot's onboard battery life and navigation performance. For this reason, we leverage an on-premise distributed fog and cloud computing platform for offloading MRVD.

An open source DL algorithm is used to extract information from the MRVD for object detection in a warehouse. However, the service time for the DL algorithm increases significantly with a higher frame rate and high image resolution. Therefore, it becomes necessary to harness the computing power from a distributed fog and cloud platform.

We consider an on-premise fog-cloud platform for experimentation in the presented work. We restate that the average processing time required for object detection when MRVD is processed over the fog or the cloud is referred to as service time. By conducting preliminary experiments on the distributed fog computing platform, we find that the service time is well within the desirable limits until the scalable limit on fog nodes is reached [7].

However, since the on-premise cloud platform offers an alternative pool of computing resources on-demand, we develop a scalable cloud platform in addition (to the fog platform) to meet service time requirements. Such an integration is referred to as the fog-cloud platform.

Since the fog-cloud platform is on-premise, the communication overheads are minimal and do not affect the navigation and the auto-scaling control loop performance. Auto-scaling compute nodes for a desirable service time bound is achieved by adopting a control-theoretic approach. In this context, firstly, a linear parameter varying (LPV) model is developed independently for the fog and the cloud. For the LPV model, we consider the number of computing nodes as the control input, the service time as the output, and the characteristics of MRVD as the time-varying parameter. Finally, we implement decentralized LPV-MPC controllers (fog and cloud) and perform validation in the actual application environment. The decentralized framework for control gives us the flexibility to modulate the service time response under time-varying MRVD by choosing a set of configurable MPC parameters. For mobile robot navigation, we have implemented Extended Kalman Filter (EKF) based SLAM [8] and the bug-based path planning algorithm [9], which run on the on-board computer of the mobile robot. The contributions of this paper are presented in Table 1.

The rest of the paper is organized as follows: Section II describes the related work, whereas Section III outlines the problem formulation. In Section IV, we detail the mobile robot vision system. Section V describes the system identification approach for modeling of the fog and the cloud platforms, whereas in Section VI, we present control design and experimental validation for the proposed LPV-MPC controllers in a simulated and realistic environment. Finally, we conclude the paper in Section VII. Additionally, we have added more details on the modeling and navigation of mobile robots in Appendix A.

TABLE 1. Contributions of the paper.

Contribution	Technique	Platform	Remarks
Localization	EKF based SLAM [8]	onboard	Implemented
Path Planning	Bug-based [9]	onboard	Implemented
Obstacle avoidance	Collision avoidance [10]	onboard	Implemented
Object Detection	COCO-SSD mobilenet [11]	Fog-cloud	Implemented
Fog: modeling	LPV local approach	Fog	Proposed
Cloud: modeling	LPV global approach	Cloud	Proposed
Fog: Auto-scaling	LPV based MPC	Fog	Proposed
Cloud: Auto-scaling	LPV based MPC	Cloud	Proposed
Load balancing	Heuristic algorithm	Fog-cloud	Proposed

II. RELATED WORK

This section discusses the relevant research on control design in distributed fog and cloud computing platforms, object detection, and mobile robot navigation.

A. MOBILE ROBOT NAVIGATION

The mobile robot navigates in an application environment to accomplish the task of object detection. Navigation comprises various tasks such as localization, path planning, and obstacle avoidance. Numerous approaches present in the literature, such as EKF-based SLAM [8] and bug-based path planning [9], have demonstrated promising outcomes in the navigation of the mobile robot. The authors of [10] have proposed an obstacle avoidance algorithm that ensures the navigation of the mobile robot by avoiding obstacles in the path. A survey of mobile robot navigation in indoor and outdoor environments is conducted by the authors of [12]. Table 1 summarizes the navigation algorithms presented in this paper.

B. OBJECT DETECTION

The literature has provided a wide range of object detection algorithms that maintain a balance between accuracy, speed, and memory [13]. You Only Look Once (YOLO), Single Shot Multibox Detector (SSD), R-CNN (Regions with Convolutional Neural Network), Faster-RCNN, [11], [14] are a few of the significant object detection algorithms.

R-CNN and Faster RCNN are two-stage algorithms that use image identification for region selection before subjecting it to detection. In contrast, SSD and YOLO are one-stage methods. Without defining the region of objects, the entire image is submitted to detection, and the results are decoded to provide the final bounding box across the objects. For this reason, one-stage methods are faster than two-stage methods, although this may result in some accuracy loss, specifically. In this work, an SSD mobilenet model is used for object detection [11].

C. CONTROL IN FOG-CLOUD PLATFORM

Mobile robots have limited battery life and on-board processing power. Hence, a distributed computing platform is a good

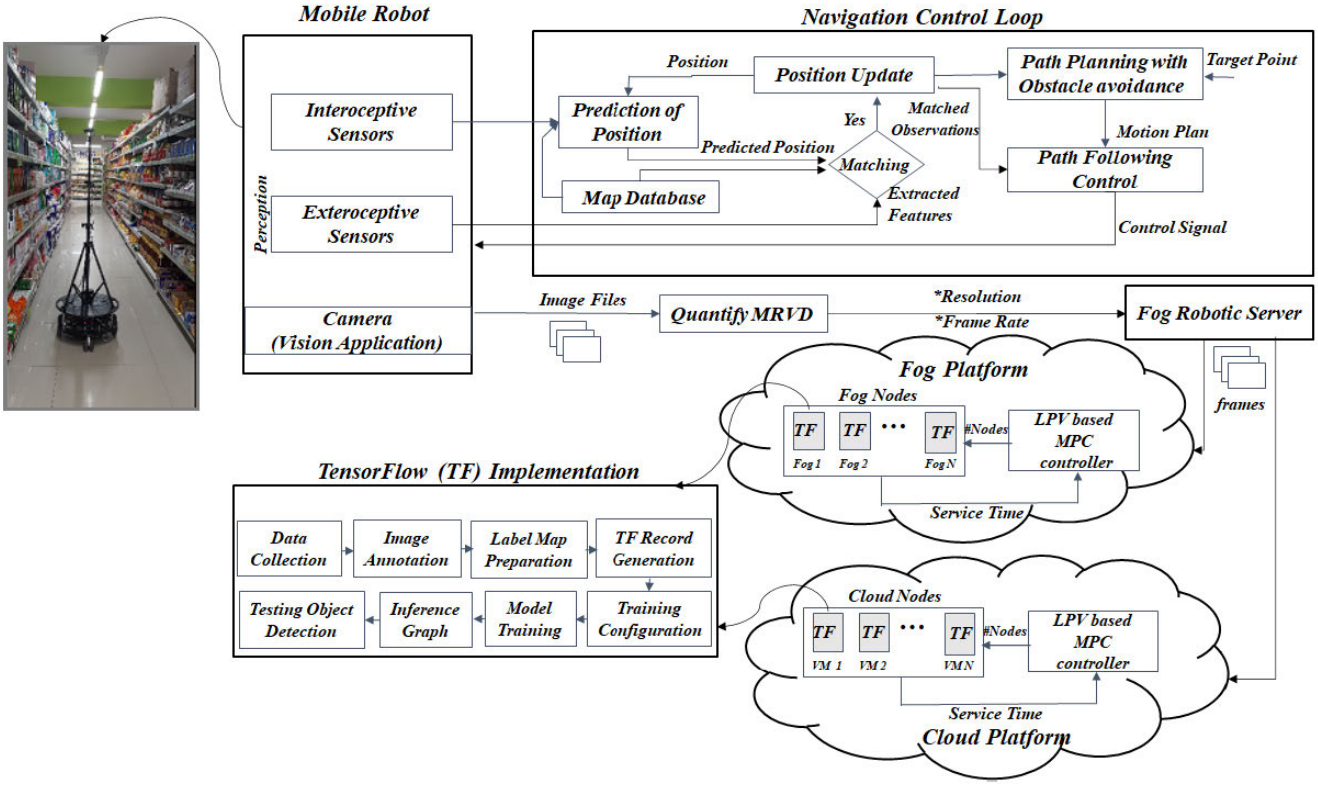


FIGURE 1. Control scheme for mobile robot navigation with object detection.

choice for offloading computationally heavy workloads. The use of fog and cloud platforms for different robotic applications to perform complicated computing tasks including vision recognition, object detection, and voice recognition has been investigated by a number of researchers. Furthermore, heuristic algorithms are used for scaling the computing nodes. Table 2 presents a few of the well-known and recent works.

We provide a control-theoretic approach for processing the MRVD offloaded to the fog and cloud platforms, in contrast to the previously described related work. According to the control-theoretic formulation, a mathematical model of the system is necessary to develop a controller. In this paper, we use system identification to develop a mathematical model in the state-space form from the input-output data [23], [24]. Later, an MPC is designed with tunable parameters for resource allocation in the fog and cloud platforms [25].

To our knowledge, no researchers have yet tried the control-theoretic approach described in this paper for developing an autonomic fog-cloud platform for vision applications using mobile robots. Hence, we believe that our contribution is novel.

III. PROBLEM FORMULATION

Fig. 1 shows the schematic diagram of a mobile robot vision system consisting of the mobile robots in the application environment with a distributed fog-cloud computing platform (additional details are available in Section IV). The mobile robot, while navigating across the application environment, captures the images of the objects (items) stacked in racks, which is referred to as MRVD.

The MRVD is characterized by the frame rate and image resolution. An open-source DL algorithm is then executed for object detection. However, the service time for the DL algorithm increases significantly with a higher image resolution and high frame rate. This demands the need for a distributed computing platform with multiple computing nodes to offload the vision data from the mobile robot. In this work, we consider the fog and the cloud as the two independently controlled distributed computing platforms on-premise. A heuristic algorithm decides the distribution of image processing tasks among the fog and the cloud layers based on MRVD characteristics.

Based on our preliminary experiments with the distributed fog computing platform, the service time overshoots the desirable value when the scalable limit on fog nodes is reached. However, this goal is accomplished by taking advantage of the available compute nodes from the on-premise or private cloud platform. The main objective is to comply with the service time requirements. We model each computing platform as an LPV state-space system with the objective of the controller as meeting the service time requirement under time-varying MRVD, with optimal use of the computing nodes. We choose MPC to achieve the mentioned objective.

TABLE 2. Robotics applications: On-premise, fog or cloud computing.

Ref.	Application	Platform	Remarks
[15]	Object recognition, speech recognition, vision recognition	Fog-Cloud	Speech recognition task performed in Google Cloud while object recognition task in the fog
[16]	3D Mapping	Cloud	Sensor data merged to a point cloud and estimate the pose of the moving camera via Software Product Lines
[17]	Object recognition and grasp planning	Cloud	An approach that distributes compute, storage, and networking resources between the cloud and the edge node
[18]	Motion planning	Cloud	Splits the motion planning computation between the robot and a high-performance cloud computing service
[19]	Human-robot interaction and environmental sensing	Cloud	Modules for smart environment, reminder, text-to-speech, internet access and speech recognition are deployed in the cloud
[20]	Video based SLAM	Cloud	Implemented a ROS-based VSLAM algorithm in a cloud
[21]	Deep learning based object detection	Fog-Cloud	Deployed deep learning-based applications in fog-cloud platform
[22]	SLAM	Fog-Cloud	A hybrid edge-fog-cloud architecture for mobile robots.

TABLE 3. Overall System Configuration.

Item	Description
• Mobile Robot (ARLO)	• Robot height: 33 cm • Robot weight: 12.5 kg • Payload capacity: 15.84 kg • Maximum speed: 0.65 m/sec • Wheel base: 39 cm • Wheel diameter: 15.4 cm • Chasis diameter: 45 cm • Battery: 12 V, 4 Ah
• Camera	• Resolution: 8 megapixels • Sensor: Sony IMX219 • Focal length: 3.04 mm
• LiDAR	• Scan Angles: $(-\pi/4, \pi/4)$ rad • Max Range : 10 m
• Encoders	• Quadrature encoder • Voltage: 5.5 VDC at 11.6 mA
• IR Sensor	• Range: 10 cm to 80 cm • Voltage: 4.5 VDC to 5.5 VDC. • Current : maximum 40 mA
• Ultrasonic distance sensor	• Range: 2 cm to 3 m. • Voltage: +5 VDC • Current: 35 mA maximum
Fog-cloud computing platform	
Fog node (Raspberry Pi 3B)	• CPU: Quadcore Broadcom BCM2837 64bit CPU @ 1.2GHz • Memory: 1GB RAM
Cloud Server	• CPU: Intel Xeon @ 2.5 GHz, 24 cores • Memory: 48 GB RAM • Disk: 2 TB

IV. MOBILE ROBOT VISION SYSTEM

This section describes the details of the mobile robot vision system. In this work, we use a differential drive mobile robot (DDMR) as depicted in Fig. 1 for navigation. Further details on the kinematic model used in navigation algorithms of the mobile robot are provided in Appendix A-A.

The mobile robot includes different sensors such as ultrasonic, LiDar, and infrared sensors to visualize the external environment. The hardware configuration and the details of the perception system of the mobile robot are shown in Table 3.

We consider the application environment to be a warehouse with numerous racks stacked with items to be detected (See Fig. 1).

TABLE 4. Performance of pre-trained models.

Model Name	Speed	COCO mAP
ssd_mobilenet_v1_coco	fast	21
ssd_inception_v2_coco	fast	24
rfcn_resnet101_coco	medium	30
faster_rcnn_resnet101_coco	medium	32
faster_rcnn_inception_resnet_v2_atrous_coco	slow	37

These items are used for training and testing in the context of object detection. A 140 cm-tall tripod stand holding three identical cameras is attached on the upper base of the mobile robot for collecting visual information from various racks.

A. FOG AND CLOUD COMPUTING PLATFORM

To process the MRVD for real-time object detection within a specific time-bound (called the service time), deep learning (DL) algorithms are hosted on distributed fog and cloud computing platforms. In Fig. 1, we present a distributed computing platform. A heuristic algorithm, primarily based on frame rate, is proposed in this work to distribute MRVD between autonomic fog-cloud computing systems. Eight Raspberry Pi modules with the specification listed in Table 3 serve as the fog nodes. Similarly, virtual machines on the cloud server serve as the cloud nodes. The cloud server allows instantiation of 10 VMs, with each VM assigned to 1 CPU core, 2 GB memory, and 50 GB disk space.

B. OBJECT DETECTION METHODOLOGY

SSD mobilenet model [11] is used in this work for real-time object detection. The objective is to detect and classify the objects by creating a bounding box across objects based on a deep neural network (DNN). It is the first method that uses a single deep neural network by eliminating the region proposal and feature sampling steps. For this reason, it is faster compared to conventional two-stage methods and more accurate compared to single-stage methods like YOLO. Mean Average

TABLE 5. Object detection accuracy of all items during training phase.

Item	Accuracy (%)	Item	Accuracy (in %)	Item	Accuracy (%)	Item	Accuracy (%)
Item-1	80-98	Item-11	70-95	Item-21	70-96	Item-31	90-95
Item-2	85-99	Item-12	84-98	Item-22	75-96	Item-32	90-98
Item-3	80-99	Item-13	80-95	Item-23	70-80	Item-33	80-95
Item-4	80-97	Item-14	96-99	Item-24	85-97	Item-34	70-99
Item-5	90-98	Item-15	84-92	Item-25	80-84	Item-35	92-98
Item-6	75-98	Item-16	70-95	Item-26	92-97	Item-36	89-98
Item-7	70-96	Item-17	80-86	Item-27	89-93	Item-37	80-90
Item-8	90-99	Item-18	90-99	Item-28	87-97	Item-38	80-91
Item-9	90-99	Item-19	80-86	Item-29	81-92	Item-39	92-99
Item-10	70-97	Item-20	70-75	Item-30	90-98	Item-40	95-99

(a) Item-1 (b) Item-2 (c) Item-3 (d) Item-4 (e) Item-5 (f) Item-6 (g) Item-7

(h) Item-8 (i) Item-9 (j) Item-10 (k) Item-11 (l) Item-12 (m) Item-13 (n) Item-14

FIGURE 2. SSD based object detection for selected items (14 out of 40) during training phase.

Precision (mAP) is a metric to measure the accuracy and speed of various pre-trained models presented in Table 4 [26].

In SSD, the output space of the bounding box is discretized to default boxes spanning various aspect ratios and scales per feature map location [11]. The confidence score indicates the match of each object in the default box with the pre-trained classes. SSD has two components, the backbone model and SSD head. The backbone model is Common Objects in Context (COCO) pre-trained model with mobilenet as the feature extractor. This helps to convert the pixels to features, and SSD head are the convolutional layers that convert the image to bounding boxes and classes.

In this work, the onboard cameras of the mobile robot are used to gather image frames of the objects in the application environment. The object detection model is pre-trained to localize and classify these objects, and a round-robin scheduling algorithm is used to distribute these frames across the fog-cloud platforms.

We have provided a comprehensive data set with multiple images of the numerous objects in the racks for efficient object detection. Images of the objects from various angles are part of the training set for the robust classifier. As a preliminary experiment, we performed object detection for 40 objects, with 70 images of each object available for training.

Bounding boxes are frequently used in annotations in the SSD mobilenet model to specify the location of the target object. In this paper, the collected images were labeled using the LabelImg image annotation tool. As illustrated in Table 5, we attained a detection accuracy 0f 70-98% during validation. Some sample items are shown in Fig. 2.

V. SYSTEM IDENTIFICATION

In this work, we propose a control-theoretic approach to develop an autonomic fog-cloud platform for robot vision applications. The objective is to meet the service time requirements for object detection with time-varying MRVD while the robot navigates across an application environment.

The time for processing the MRVD is referred to as the service time. Since object detection is computationally intensive, especially for high frame rate and high image resolution, an object detection algorithm is hosted on each of the computing nodes, which can eventually be scaled as per requirement. The variation in the number of computing nodes has an impact on the service time [27]. The increase in the number of computing nodes decreases the service time.

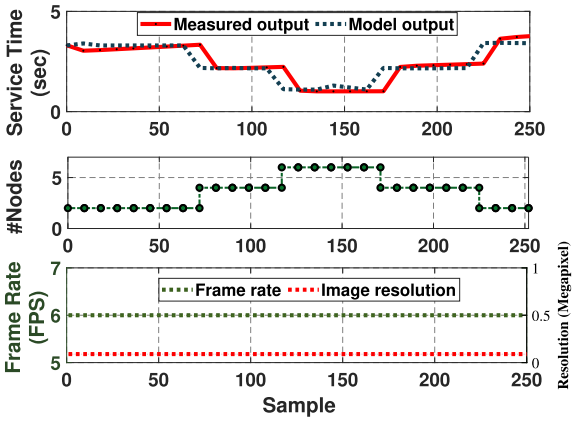


FIGURE 3. LTI validation plot: Fog computing platform.

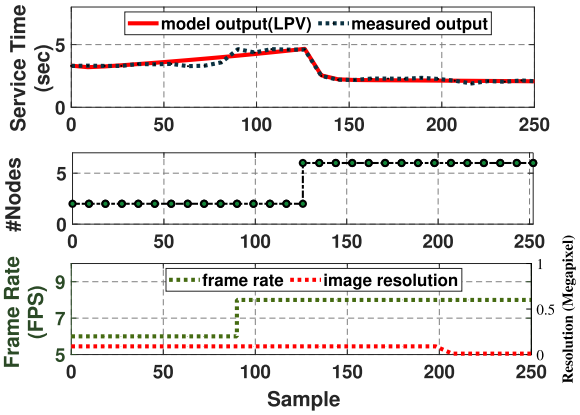


FIGURE 4. LPV validation plot: Fog computing platform.

Thus, in the mathematical model (proposed later in Section V-B and V-B2), the number of computing nodes is considered as the control input, the service time is considered as the output, and frame rate and image resolution as the parameters (refer Section V-A) for the proposed mathematical model. The modeling of each computation platform and its respective controller design is presented in the following subsections.

A. QUANTIFYING MRVD

To gather the vision data (MRVD) from the application environment, the mobile robot follows different paths such as square, elliptical, or eight shaped depending on the navigation algorithm. Appendix A-B provides further information on mobile robot navigation. Depending on the path of the mobile robot, the distance between center of gravity of the robot and the location of the image varies. This variation in distance is reflected in the resolution of the object under consideration in the image captured by the robot. Furthermore, there would be a change in the frame rate since the mobile robot may navigate at different speeds as per the path plan and unintended obstacles. Hence, the image resolution and the frame rate are considered as the parameters of the LPV model described in the next subsection.

B. LPV MODELING OF THE FOG AND THE CLOUD PLATFORM

The first principle method and the empirical approach are commonly used methods for modeling a system. The physics of the system serves as the foundation for the first-principle method. For systems whose dynamics are not entirely known, empirical modeling often referred to as "black-box modeling," is the best option. Since the application under discussion may depict system dynamics over a larger operating region with time-varying MRVD parameters, the LPV state-space model suits the requirement.

To develop an LPV model of the system, either a local or global approach can be used [23], [24]. In the local approach, we interpolate LTI models found at specified operating points to develop the LPV model, but in the global approach, the LPV model of the system is developed directly from the data. In this paper, for the fog platform, we use the n4sid technique to implement the local LPV approach. We employ the global LPV approach with the prediction error minimization approach for the cloud platform.

1) LPV MODELING OF THE FOG PLATFORM

The LPV state-space model of the distributed fog computing platform is derived in this section using system identification via the local approach [24].

To gain insight into the service time dynamics with regard to fog nodes at constant frame rate and image resolution, an identification experiment is conducted with a sampling period of 1 *sec*. This represents a single operating region for the LTI state-space system.

An LTI state-space model is identified, and the model parameters are estimated using the system identification toolbox in MATLAB. We find that a second order model has the best fit in comparison to other orders. This is given by:

$$\begin{aligned} z(k+1) &= Az(k) + Bu(k) \\ y(k) &= Cz(k) + Du(k) \end{aligned} \tag{1}$$

where $z \in \mathbb{R}^m$, $u \in \mathbb{R}^w$ and $y \in \mathbb{R}^l$ (for a second-order model with single output and single input, $m = 2$, $w = 1$, $l = 1$). Since the number of fog nodes, n affects the service time, it is considered as the input of the system, and the service time, T_s is chosen as the output of the system.

The observable form of representation is used to consistently define the local LTI models [23]. Each of the identified LTI models is valid for a particular operating region. We derive twenty such LTI models, each corresponds to an operating region of distinct frame rate and image resolution, which eventually are chosen as the scheduling parameters for the proposed LPV state-space model.

The scheduling parameters of all the operating regions are chosen in the range [1,15] FPS and [0.01, 0.25] megapixel, respectively, for frame rate and image resolution. The bounds of the parameters are chosen such that the service time constraints are satisfied.

The model validation plots for the 3^{rd} operation region (frame rate: 6 FPS, image resolution: 0.09 megapixel) are shown in Fig. 3. The LPV model is obtained by interpolating the constituent LTI models corresponding to the twenty operating regions. We present a general optimization methodology to obtain the LPV models using the local approach from the constituent LTI models in the following section. Fig. 4 shows the LPV validation plot with time-varying frame rate and image resolution.

An LPV model can be obtained by interpolating the local LTI state-space models. The parameter vector, p (parameter vector $p(k)$ is written as p in short) includes time-varying parameters, frame rate (γ) and image resolution (ρ) and is given by:

$$p = \begin{bmatrix} \gamma & \rho \end{bmatrix}' \tag{2}$$

The discrete-time LPV system is represented as:

$$\begin{aligned} z(k+1) &= A(\gamma, \rho)z(k) + B(\gamma, \rho)u(k) \\ y(k) &= C(\gamma, \rho)z(k) + D(\gamma, \rho)u(k) \end{aligned} \tag{3}$$

The structure of the state-space matrices is as below.

$$\begin{aligned} A(\gamma, \rho) &= \begin{bmatrix} 0 & a_{r,1}(\gamma, \rho) \\ 1 & a_{r,2}(\gamma, \rho) \end{bmatrix} \\ B(\gamma, \rho) &= \begin{bmatrix} 1 \\ b_{r,1}(\gamma, \rho) \end{bmatrix} \\ C(\gamma, \rho) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \\ D(\gamma, \rho) &= \begin{bmatrix} 0 \end{bmatrix} \end{aligned}$$

The system dynamics of the LTI models depend on the time-varying parameters. The least square optimization method is used in this work to estimate the LPV model by interpolating the constituent LTI models [23]. The matrix representation of the LPV system is:

$$H(p) = \left[\begin{array}{c|c} A(p) & B(p) \\ \hline C(p) & D(p) \end{array}\right] \tag{4}$$

The polynomial dependency of each block in (4) on p is given by:

$$\begin{aligned} A(p) &= \sum_{d=1}^{r} A_d(\alpha) \qquad & B(p) &= \sum_{d=1}^{r} B_d(\alpha) \\ C(p) &= \sum_{d=1}^{r} C_d(\alpha) \qquad & D(p) &= \sum_{d=1}^{r} D_d(\alpha) \end{aligned} \tag{5}$$

where α is a polynomial function of degree N, in terms of p. From the local LTI model, the state-space matrices A_d, B_d, C_d and D_d can be obtained and are given by:

$$H_d = \left[\begin{array}{c|c} A_d & B_d \\ \hline C_d & D_d \end{array}\right] \tag{6}$$

where r is the number of operating regions taken into consideration and d=1, 2,..., r. A parameter, p, is assigned to each operating region. As a result, the parameters p_1, p_2,..., p_r correspond to different operating regions. The following is the cost function for the optimization algorithm to estimate the LPV model from the constituent LTI models:

$$\begin{aligned} F(a_{1,N}, b_{1,N}, c_{1,N}) = &\sum_{j=1}^{r}\sum_{i=1}^{2} |a_i^j - \sum_{t=0}^{N} a_{1,t}^i p_j(k)^t|^2 \\ &+ \sum_{j=1}^{r} |b_1^j - \sum_{t=0}^{N} b_{1,t}^1 p_j(k)^t|^2 \end{aligned} \tag{7}$$

where r LTI models are indexed by j and i is used for indexing the coefficients, a_1 and, a_2. The objective of the optimization problem is to minimize the error between each state-space matrix entry and their polynomial estimation.

Consequently, (7) is redefined as:

$$F(s) = ||T(s)||^2 = T(s)'T(s) \tag{8}$$

where

$$T(s) = \begin{bmatrix} a_1^1 - (a_{1,0}^1 + a_{1,1}^1 p_1 + a_{1,2}^1 p_1^2 + \ldots + a_{1,N}^1 p_1^N) \\ a_1^2 - (a_{1,0}^1 + a_{1,1}^1 p_2 + a_{1,2}^1 p_2^2 + \ldots + a_{1,N}^1 p_2^N) \\ \ldots \\ a_1^r - (a_{1,0}^1 + a_{1,1}^1 p_r + a_{1,2}^1 p_r^2 + \ldots + a_{1,N}^1 p_r^N) \\ a_2^1 - (a_{2,0}^1 + a_{2,1}^1 p_1 + a_{2,2}^1 p_1^2 + \ldots + a_{2,N}^1 p_1^N) \\ a_2^2 - (a_{2,0}^1 + a_{2,1}^1 p_2 + a_{2,2}^1 p_2^2 + \ldots + a_{2,N}^1 p_2^N) \\ \ldots \\ a_2^r - (a_{2,0}^1 + a_{2,1}^1 p_r + a_{2,2}^1 p_r^2 + \ldots + a_{2,N}^1 p_r^N) \\ b_1^1 - (b_{1,0}^1 + b_{1,1}^1 p_1 + b_{1,2}^1 p_1^2 + \ldots + b_{1,N}^1 p_1^N) \\ \ldots \\ b_1^r - (b_{1,0}^1 + b_{1,1}^1 p_r + b_{1,2}^1 p_r^2 + \ldots + b_{1,N}^1 p_r^N) \end{bmatrix} \tag{9}$$

$$s = [a_{1,0}^1, a_{1,1}^1, a_{1,2}^1, \ldots, a_{1,N}^1 a_{2,0}^1, a_{2,1}^1, a_{2,2}^1, \ldots, a_{2,N}^1] \tag{10}$$

The function $T(s)$ can be represented as $T(s) = Ts - v$ where T and v can be obtained from (9). Thus the optimisation problem (8) can be rewritten as:

$$\min_s T(s) = \min_s ||Ts - v||_2^2 \tag{11}$$

and the optimized value of s is:

$$s^* = T^+ v \tag{12}$$

where T^+ is the pseudo inverse of T.

2) LPV MODELING OF THE CLOUD PLATFORM

The discrete-time LPV system dynamics for cloud is given by:

$$\begin{aligned} z(k+1) &= A(p)z(k) + B(p)u(k) \\ y(k) &= C(p)x(k) + D(p)u(k) \end{aligned} \tag{13}$$

where p is the scheduling parameter, $z(k) \in \mathbb{R}$ denotes the state, $u(k) \in \mathbb{R}$ denotes the control input, and $y(k) \in \mathbb{R}$ denotes the output and A, B, C and D are matrices of appropriate dimension. Here, the service time is considered as the state variable and also the output variable of the system. The control input is the number of virtual machines active in the cloud.

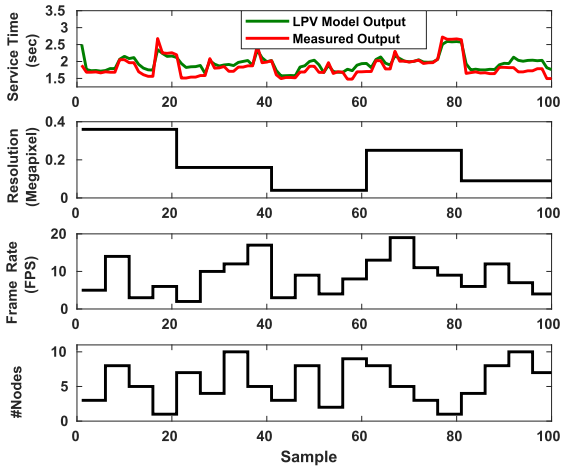


FIGURE 5. LPV validation plot: Cloud computing platform.

The LPV model of the cloud is obtained by the global LPV approach using the prediction error minimization approach.

Global LPV identification approach: Consider the discrete-time LPV system given by (13) where $z(k) \in \mathbb{R}^{n_z}$ denotes the state, $u(k) \in \mathbb{R}^{n_u}$ denotes the control input, and $y(k) \in \mathbb{R}^{n_y}$ denotes the output. Here, n_z, n_y, and n_u represent the dimensions of $z(k)$, $y(k)$, and $u(k)$ respectively and $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$. $p(k)$ is expressed as p for the sake of brevity.

Assumption: The current value of time-varying scheduling parameter $p = [p_1, p_2, \ldots, p_M] \in \mathbb{R}^M$ is known at each sample, however, the future values are uncertain. The elements in A, B, and C of (13) are given by:

$$a_{ij}(p) = \sum_{k=0}^{n_a} a_{k_{ij}} p^k; \quad b_{ij}(p) = \sum_{k=0}^{n_b} b_{k_{ij}} p^k$$
$$e_{ij}(p) = \sum_{k=0}^{n_e} e_{k_{ij}} p^k; \quad c_{ij}(p) = \sum_{k=0}^{n_c} c_{k_{ij}} p^k$$

We assume $D(p) = 0$. The parameters that are to be estimated in the LPV model are $P = [a_{k_{ij}}\ b_{k_{ij}}\ c_{k_{ij}}]^T_{L_p}$, where the dimension of P is L_p and is equal to $n_a n_x n_x + n_b n_x n_u + n_c n_x n_y$. We determine the optimal one-step ahead prediction of the output equation in (13) to estimate parameters P. The one-step ahead prediction is denoted by $\hat{y}(k|k-1, \Theta)$. Finally, the following objective function is solved for the parameter estimation.

$$J(P, \mathrm{D}) = \min_P \frac{1}{N} \sum_{k=1}^{N} \frac{1}{2} (y(k) - \hat{y}(k|k-1, P))^2 \qquad (14)$$

where $y(k)$ is measured output data and D is the available data set, $\mathrm{D} = \{u(k), y(k), \theta(k)\}$ for $k = 1, 2, \ldots, L$. The objective function can be solved numerically as the function is nonlinear.

For identifying the LPV model, a data set D with 1200 samples is collected from cloud nodes while ensuring that the entire input space is covered. The LPV model of cloud is obtained by solving (14) with different choices for the order of model structure which is based on cross-validation result obtained for various values of n_a, and n_b, where n_a and n_b denote the order of polynomial of $A(p)$ and $B(p)$. Firstly, a linear model is chosen by taking $n_a = n_b = 0$. We found that the fit is good in a few of the operating regions. Therefore, the order is increased from 1 to 5 with a step size of 1. The maximum accuracy is obtained for $n_a = n_b = 1$.

The model validation plot is shown in Fig. 5. We see that the output, i.e., the service time obtained from the model closely follows actual service time with time-varying frame rate and image resolution.

VI. CONTROL DESIGN AND EXPERIMENTAL VALIDATION

Subsequent to modeling each of the fog and the cloud computing platforms independently as LPV state-space systems, we design an MPC for control (also called LPV-MPC controller). Also, we propose a decentralized MPC for the fog and cloud platform. The objective of these controllers is to meet the service time requirement under time-varying MRVD, with optimal use of the respective computing nodes. We chose a set value of service time to be 2 seconds for the mobile robot application under consideration; doing so enables the mobile robot vision system to capture image frames with minimal aberrations.

A. MPC FOR LPV SYSTEM

An explicit MPC can be designed for the discrete-time LPV systems given in (13). In this approach, the inputs to the system are precomputed as piece-wise affine functions of the state and stored in a look-up table where at each sampling instance the look-up table needs to be evaluated. In MPC, a cost function based on finite-horizon predictions is minimized and the resulting control $u(k)$ is applied. Let the control sequence over the prediction horizon (N_p) be given by $\mathrm{U} := \{u(k), u(k+1), \ldots, u(k+N_p-1)\}$ and the unknown sequence of future scheduling parameters be given as $\mathrm{P} := \{p(k+1), \ldots, p(k+N_p-1)\}$. The standard cost function for the MPC [28] is given as:

$$\begin{aligned} &J(x(k), p(k), \mathrm{U}, \mathrm{P}) \\ &= \sum_{i=0}^{N_p-1} \Big[x(k+i)^T Q x(k+i) \\ &\quad + u(k+i)^T R u(k+i) \Big] + x(k+N_p)^T Q_t x(k+N_p) \end{aligned} \qquad (15)$$

where the positive definite weighting matrices for the states, control inputs, and terminal state, are denoted by Q, R, and Q_t respectively.

Also, in many cases, the states and the control input of the system are constrained and let this be given by $x(k) \in \mathbb{X}$ and $u(k) \in \mathbb{U}$. The set $\mathbb{X}$ covers the entire state space while the set $\mathbb{U}$ includes all the possible values of control input. Also, the terminal state is constrained over a set $\mathbb{X}_t$, i.e. $x(k+N_p) \in \mathbb{X}_t$. Since the future values of the scheduling parameters are unknown, the worst-case cost is considered. The optimization

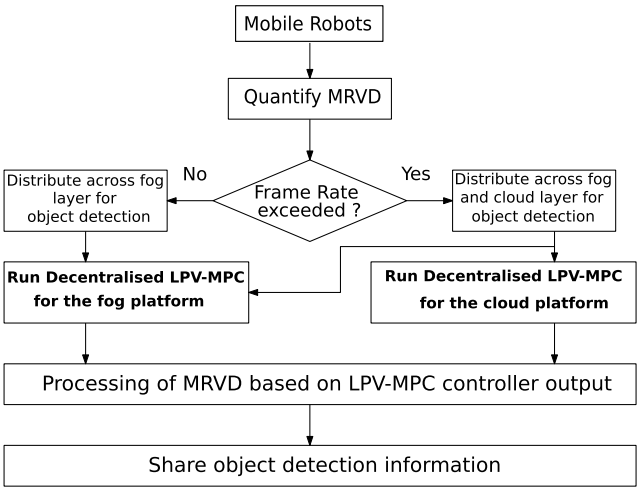


FIGURE 6. Summary of the proposed approach.

problem solved over the finite horizon is given by:

$$u^*(k) = \arg \min_{u(k)} \max_{p(k+1)} \cdots \min_{u(k+N_p-1)} J(x(k), p(k), \mathrm{U}, \mathrm{P}(k)) \tag{16}$$

$$\begin{aligned} \text{s.t. } \forall\, i \in\ & \{0, 1, \ldots, N_p - 1\} \\ & x(k+i+1) = A(\theta(k+i))x(k+i) \\ & + B(\theta(k+i))u(k+i) \\ & y(k+i) = Cx(k+i) \leq y_{max} \\ & x(k+i) \in \mathbb{X}; \quad u(k+i) \in \mathbb{U}; \quad x(k+N_p) \in \mathbb{X}_t \end{aligned} \tag{17}$$

where C is of appropriate dimension.

The optimization problem (16) can be solved by a dynamic programming approach iterating backward in time [25]. Only the first control input in the sequence of control inputs is applied to the system. This sub-section presents the experimental results of the LPV-MPC controllers in a simulated and realistic environment. For validation, we consider a single cloud machine with a single fog layer and multiple cloud machines with multiple fog layers, where a layer refers to a physical machine having multiple compute nodes. The flowchart shown in Fig. 6 summarizes the proposed approach.

The MPC parameters for the experiments performed in this section are shown in Table 6. In Table 7, we have summarized the experimental results and provided reference to corresponding figures.

To perform the controller validation experiments, we have chosen one cloud layer (a cloud server with 10 VMs) and one fog layer (with 8 fog computing nodes), refer Fig. 1 for the fog-cloud architecture for MRVD processing. The distribution of frames for processing is done in the following way: if the frame rate is less than or equal to a given threshold, then the image frames are sent to the fog, else the remaining frames are sent to the cloud.

For the fog platform, Fig. 7 shows the MPC control action with frame rates below the threshold, and the control action

TABLE 6. Model accuracy and MPC parameters.

Parameters	Cloud	Fog
System identification (LPV model)		
Fit to estimation data	70.94%	87.57%
MPC controller design		
Prediction horizon (N_p)	20	20
$[Q, Q_t, R]$	$[1, 0.2, 0.5]$	$[I_2, 0.5, 1]$
Reference service time	2 sec	2 sec
Input constraints	$[0, 10]$	$[0, 8]$

TABLE 7. Overview of experimental results.

Description	Platform	Layer	Reference
Single fog layer and single cloud layer			
Frame rate $\leq$ threshold	Fog	Layer 1	Fig. 7
Frame rate $>$ threshold	Fog	Layer 1	Fig. 8
Frame rate overhead from Fog	Cloud	Layer 1	Fig. 9
Load Sharing : Validation	Fog-Cloud	Layer 1	Fig. 10
Load Sharing : Actual	Fog-Cloud	Layer 1	Fig. 26
Simulation with multiple fog layers and multiple cloud layers			
Load Sharing : Algorithm 1	Fog	Layer 1	Fig. 11
	Fog	Layer 2	Fig. 12
	Fog	Layer 3	Fig. 13
	Cloud	Layer 1	Fig. 14
	Cloud	Layer 2	Fig. 15
	Cloud	Layer 3	Fig. 16
Load Sharing: Algorithm 2	Fog	Layer 1	Fig. 19
	Fog	Layer 2	Fig. 20
	Fog	Layer 3	Fig. 21
	Cloud	Layer 1	Fig. 22
	Cloud	Layer 2	Fig. 23
	Cloud	Layer 3	Fig. 24

leads to a deviation of service time well below the threshold of 2 *sec*. Whereas in Fig. 8, we present the case with the fog as the only computing resource (without the cloud being used), the control action obviously leads to deviation of service time, much above the threshold of 2 *sec*. This necessitates the use of the cloud nodes for higher frame rates. The MPC validation for the cloud layer serving frame overheads from the fog layer is shown in Fig. 9.

B. SINGLE FOG LAYER AND SINGLE CLOUD LAYER
C. LOAD BALANCING BETWEEN SINGLE FOG AND SINGLE CLOUD LAYER

Before we present the working of the integrated system consisting of the mobile robot and the fog-cloud computing platform, we detail the working of a load balancing mechanism. Distributing frame overheads from the fog to the cloud is executed by a heuristic algorithm primarily based on the frame rate. As seen in Fig. 10, the fog platform maintains the service time mostly below the reference value 2 *sec* while the frame rate is below 15 FPS. However, when the frame rate is above 15 FPS, the cloud platform takes over the processing of the additional frames with the aim to keep service time below the threshold value of 2 *sec*.

D. MULTIPLE FOG LAYERS AND MULTIPLE CLOUD LAYERS

To show the applicability of the proposed approach in the context of multiple fog and cloud layers, we choose three cloud layers (each cloud layer with ten virtual machines as nodes) and three fog layers (each layer consisting of eight fog nodes). Such a situation can arise when there are multiple

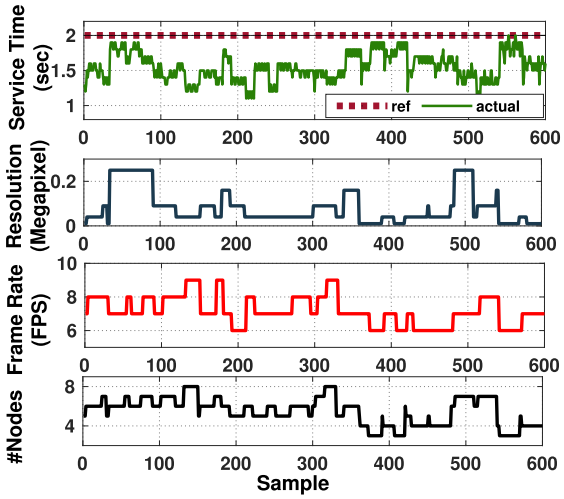


FIGURE 7. MPC validation: Fog layer 1 with FPS ≤ 15.

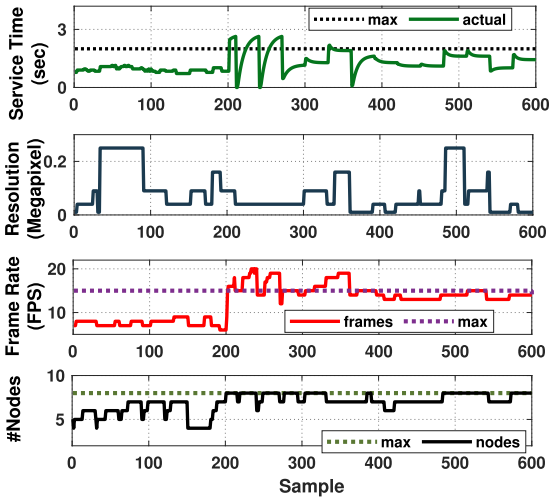


FIGURE 8. MPC validation: Fog layer 1 with FPS > 15 intermittently.

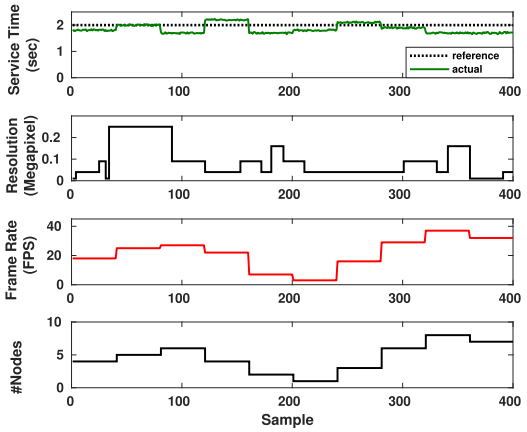


FIGURE 9. MPC validation: Cloud platform (serving frame overheads from the fog layer).

mobile robots performing a coordinated vision task to acquire image frames.

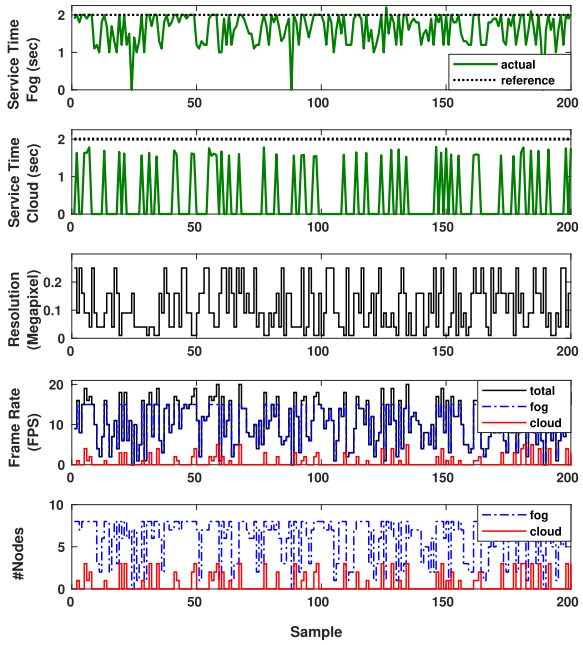


FIGURE 10. Validation of load sharing in fog-cloud platform.

Algorithm 1 Regular Scheme for Load Sharing

Input: Total Frame at instant k
Output: (Fog Layer, #Frame), (Cloud Layer, #Frame)
// n_f is no. of available fog layers and m_f is the limit of each fog layer
// n_c is no. of available cloud layers and m_c is the limit of each cloud layer
for each $Fr_i \in$ Frame **do**
  if $Fr_i \leq n_f m_f$ **then**
    for $n = 1$ *to* $ceil(Fr_i/m_f)$ **do**
      for each $fog_n \in Fog$ **do**
        $fog_n \leftarrow [(n-1)m_f + 1]$ *to* $[min(Fr_i, nm_f)]$ ▷ // send these frames to fog_n
      end for
    end for
  else
    for $n = 1$ *to* n_f **do**
      for each $fog_n \in$ Fog **do**
        $fog_n \leftarrow [(n-1)m_f + 1]$ *to* $[nm_f]$ ▷ // send these frames to fog_n
      end for
    end for
    for $p = 1$ *to* $ceil((Fr_i - n_f m_f)/m_c)$ **do**
      for each $cloud_p \in$ Cloud **do**
        $cloud_p \leftarrow [n_f m_f + (p-1)m_c + 1]$ *to* $[min(Fr_i, n_f m_f + pm_c)]$ ▷ // send these frames to $cloud_p$
      end for
    end for
  end if
end for

Refer Fig. 1 for the fog-cloud architecture for MRVD processing. A heuristic algorithm, based on frame rate, is used to distribute the load either to fog or to a fog-cloud platform as shown in Algorithm 1 and Algorithm 2. The flowcharts corresponding to these Algorithms are shown respectively in Fig. 17 and Fig. 18.

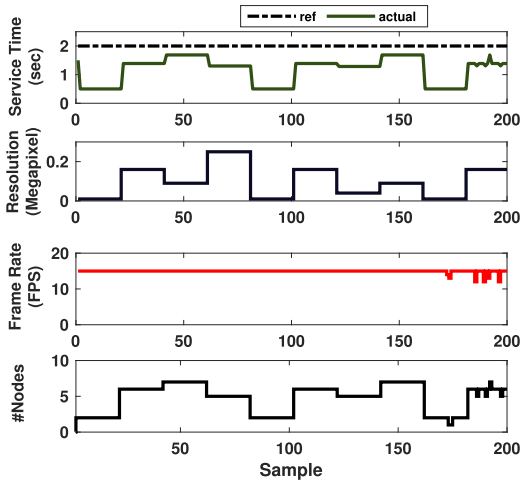


FIGURE 11. Alg. 1 for load sharing: Fog server 1.

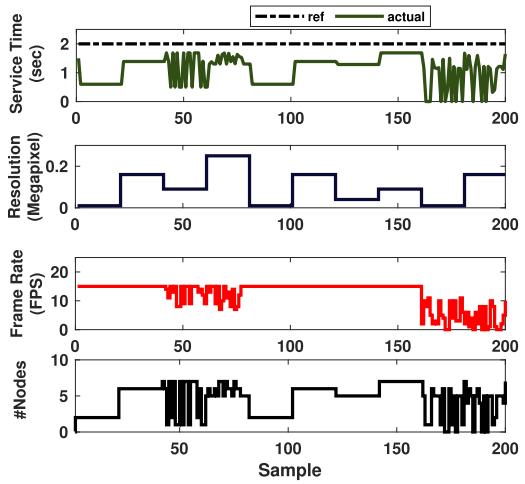


FIGURE 12. Alg. 1 for load sharing: Fog server 2.

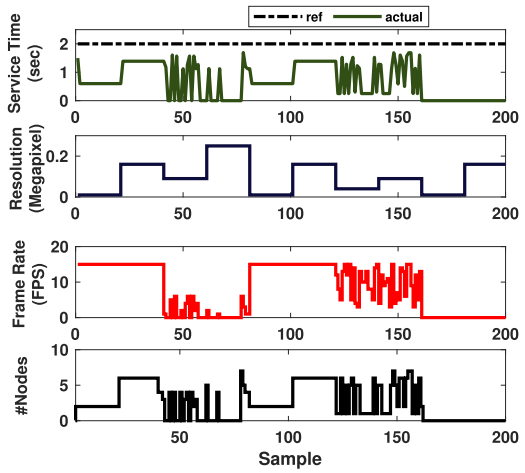


FIGURE 13. Alg. 1 for load sharing: Fog server 3.

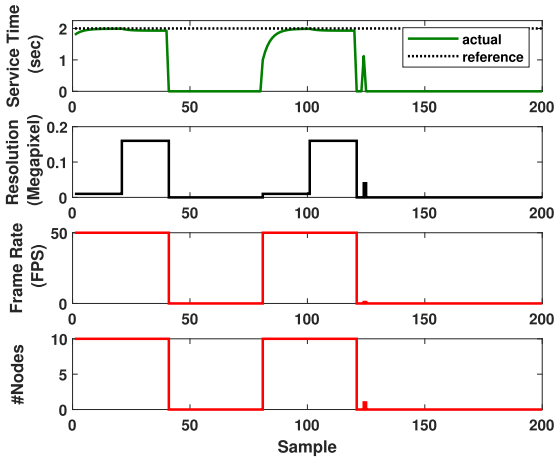


FIGURE 14. Alg. 1 for load sharing: Cloud server 1.

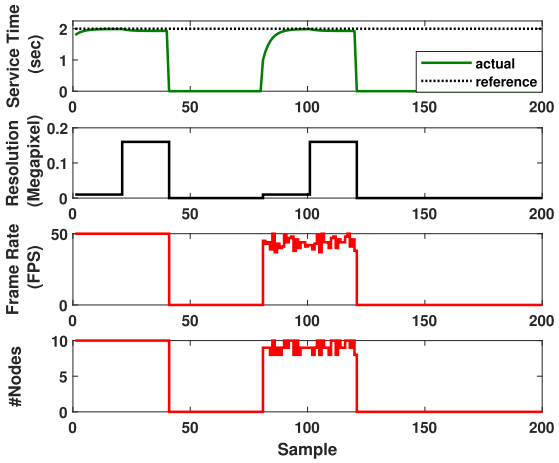


FIGURE 15. Alg. 1 for load sharing: Cloud server 2.

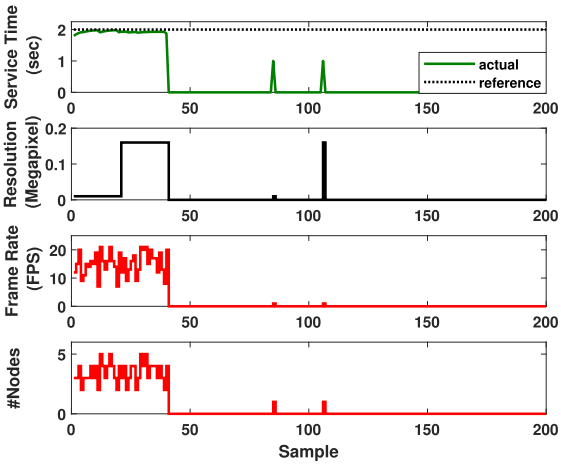


FIGURE 16. Alg. 1 for load sharing: Cloud server 3.

Algorithm 1 presents a scheme for the distribution of image frames among the fog and the cloud layers. If the total frames received by the load balancer at any instant (Fr_i) is less than or equal to the frame limit of the total fog layers, $n_f m_f$, where n_f is the number of available fog layers and m_f is the frame limit of each fog layer, the MRVD is sent to the fog platform, or else

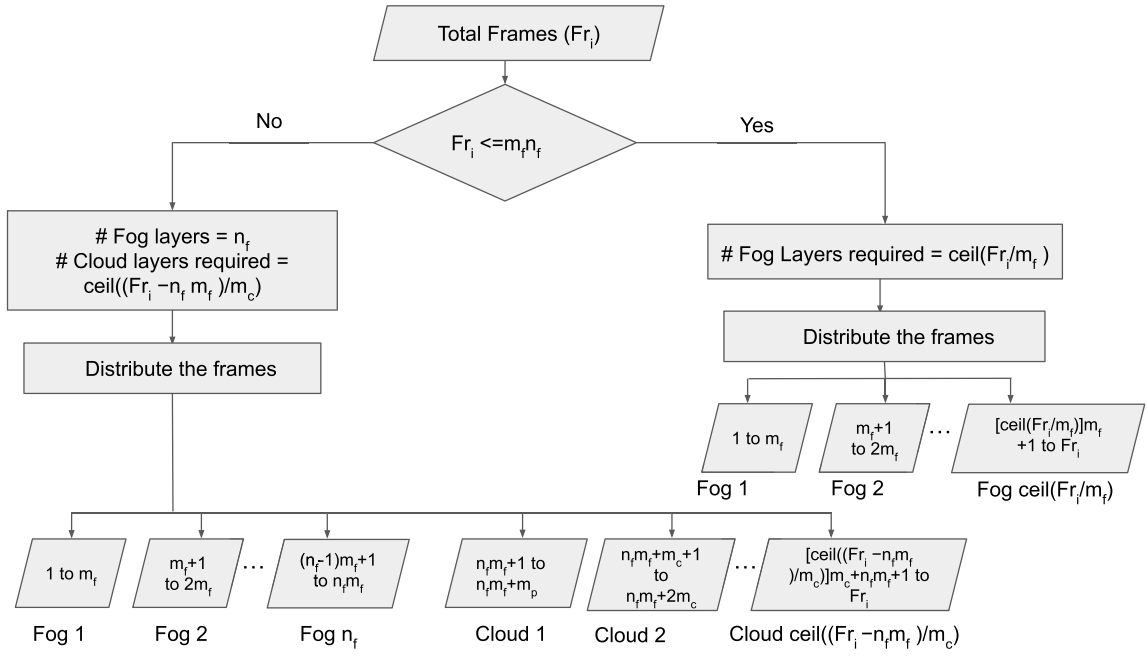


FIGURE 17. Flowchart corresponding to Algorithm 1.

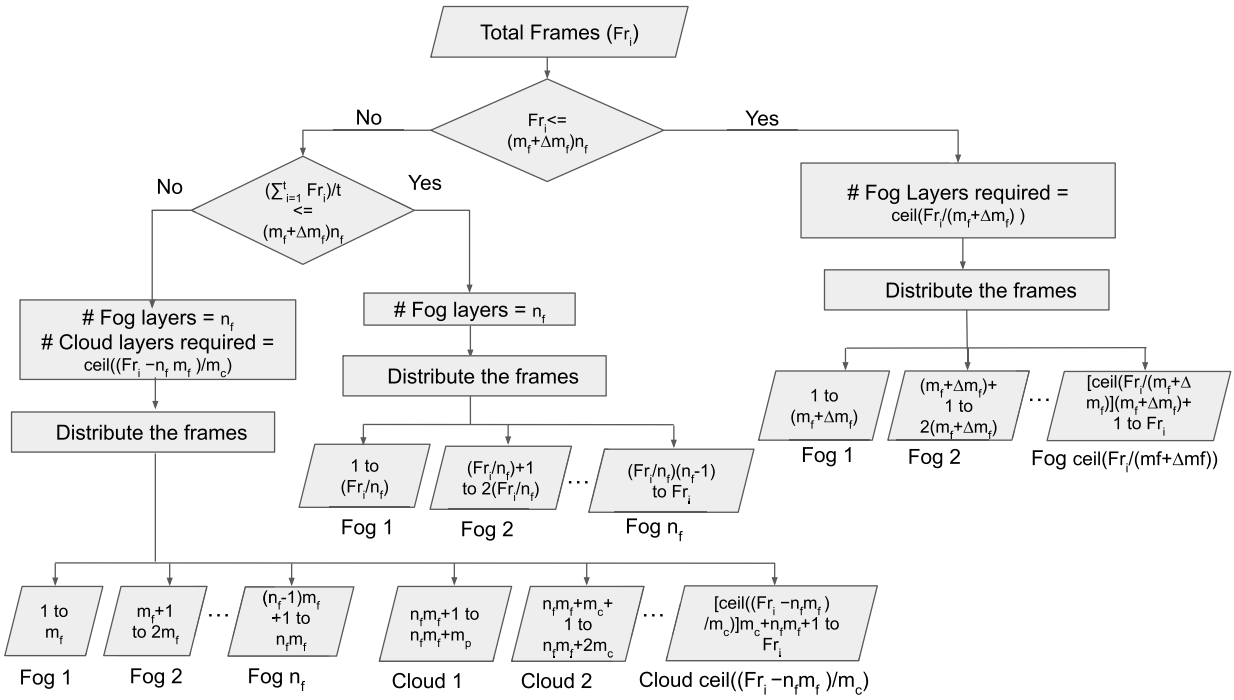


FIGURE 18. Flowchart corresponding to Algorithm 2.

it is sent to cloud platform in addition. The MRVD is sent to one computing layer (fog/cloud) only once the previous layer reaches the maximum limit (m_f for fog and m_c for cloud). In this experiment, n_f=3, m_f=15 FPS, and m_c=50 FPS. The fog platform maintains the service time below 2 *sec* for the processing of MRVD when Fr_i is less than $n_f m_f$ as shown in Fig. 11, Fig. 12, and Fig. 13. But with Fr_i above $n_f m_f$ cloud platform takes over the processing of MRVD (serving frame overhead from the fog layer) with service time less than 2 *sec* as shown in Fig. 14, Fig. 15, and Fig. 16.

Algorithm 2 details the modified scheme for the distribution of frames. The objective of this algorithm is to minimize

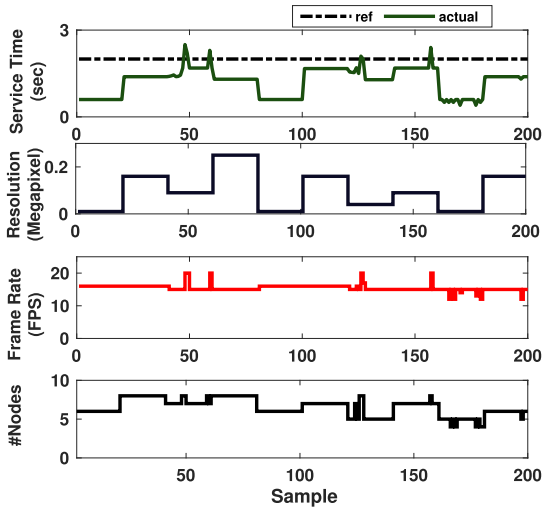


FIGURE 19. Alg. 2 for load sharing: Fog server 1.

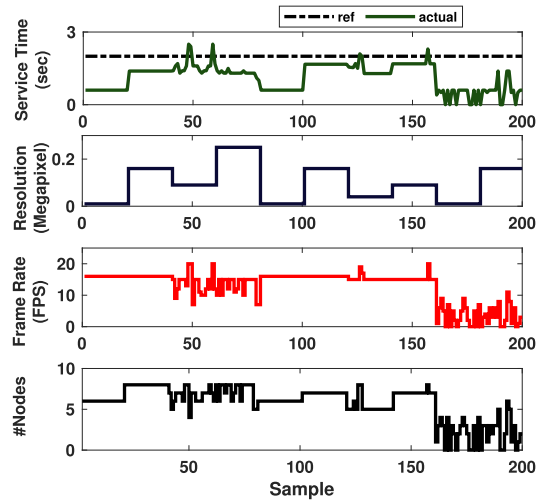


FIGURE 20. Alg. 2 for load sharing: Fog server 2.

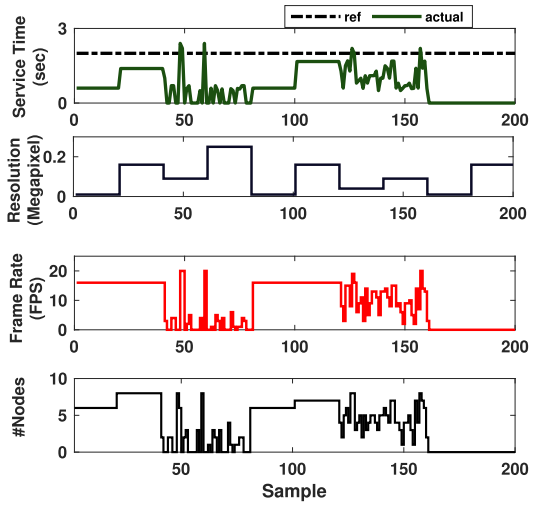


FIGURE 21. Alg. 2 for load sharing: Fog server 3.

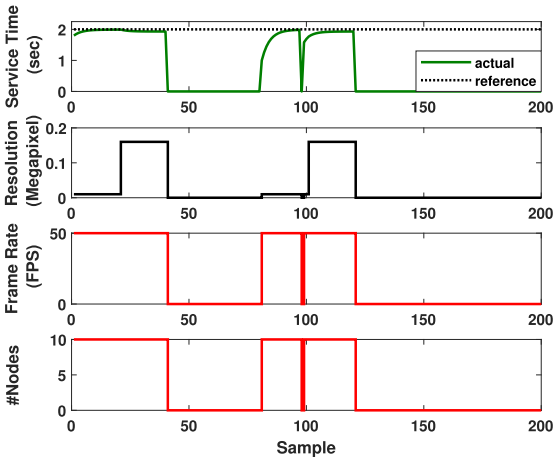


FIGURE 22. Alg. 2 for load sharing: Cloud server 1.

the usage of the cloud layers at the cost of increased service time. This happens when only the fog nodes are used with the number of frames exceeding the capacity of the fog layer by a certain threshold. The modification has been presented in Algorithm 2, in which if Fr_i is less than or equal to $(m_f+\Delta_m)n_f$, where Δ_m is the extra frames that a fog layer can handle with a small increase in the service time, the MRVD is sent to the fog platform. Or else, the average of Fr_i for the last five samples is calculated, and if it is less than or equal to $(m_f+\Delta_m)n_f$, the MRVD is again sent to the fog platform to save the cloud resources. The MRVD is sent to the cloud in addition to fog only if the average of Fr_i for the last five samples is greater than or equal to $(m_f+\Delta_m)n_f$. For the MRVD distributed to the fog nodes, the fog platform maintains the service time for the processing of MRVD below 2s as shown in Fig. 19, Fig. 20, and Fig. 21. Otherwise, the cloud platform takes over the processing of MRVD (over and above fog platform) with service time less than 2 *sec* as shown in Fig. 22, Fig. 23, and Fig. 24.

E. CONTROLLER VALIDATION IN THE APPLICATION ENVIRONMENT

The controller validation results presented so far were considered individually for the fog and the cloud layer with varying MRVD. We now present the controller validation results in the context of a realistic application environment such as a warehouse.

Fig. 25 corresponds to the environment map of the warehouse considered. The black rectangular boxes in Fig. 25 resemble the warehouse racks, where items are stacked together. The red cross marks show the way points that determine the reference path, and the black circles represent obstacles in the path of the mobile robot. The aim is to navigate the mobile robot from the start point to the destination (the green line shows the actual path of the robot). To improve the readability of the validations, we have provided additional

Algorithm 2 Relaxed Scheme for Load Sharing

Input: Total Frame at instant k
Output: (Fog Layer, #Frame), (Cloud Layer, #Frame)
// n_f is no: of available fog layers and m_f is the limit of each fog layer
// n_c is no: of available cloud layers and m_c is the limit of each cloud layer
for each $Fr_i \in Frame$ **do**
  if $Fr_i \leq (m_f + \Delta_m)n_f$ **then**
    //Δ_m is the extra frames that can be handled by fog nodes by compromising service time
    for $n = 1$ *to* $ceil(Fr_i/(m_f + \Delta_m))$ **do**
      for each $fog_n \in Fog$ **do**
        $fog_n \leftarrow [(n-1)(m_f+\Delta_m)+1]$ *to* $[min(Fr_i, n(m_f + \Delta_m))]$ ▷ // send these frames to fog_n
      end for
    end for
  else
    Calculate average of Fr_i for last t samples ▷ // t is threshold samples chosen
    if average of Fr_i for last t samples $\leq (m_f + \Delta_m)n_f$ **then**
      for $n = 1$ *to* n_f **do**
        for each $fog_n \in Fog$ **do**
          $fog_n \leftarrow [ceil((Fr_i - (n-1))/n_f)]$ ▷ // send these frames to fog_n
        end for
      end for
    else
      for $n = 1$ *to* n_f **do**
        for each $fog_n \in Fog$ **do**
          $fog_n \leftarrow [(n-1)m_f + 1]$ *to* $[nm_f]$ ▷ // send these frames to fog_n
        end for
      end for
      for $p = 1$ *to* $ceil((Fr_i - n_f m_f)/m_c)$ **do**
        for each $cloud_p \in Cloud$ **do**
          $cloud_p \leftarrow [n_f m_f + (p-1)m_c + 1]$ *to* $[min(Fr_i, n_f m_f + pm_c)]$ ▷ // send these frames to $cloud_p$
        end for
      end for
    end if
  end if
end for

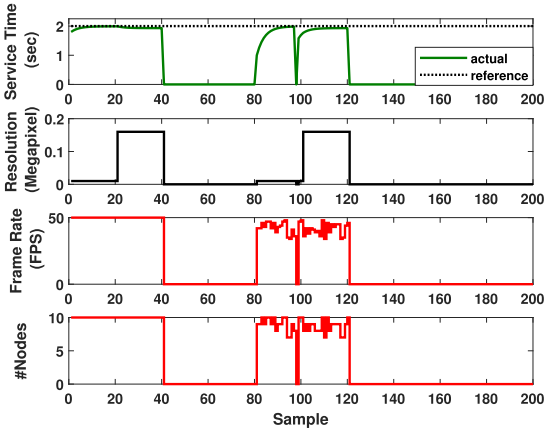


FIGURE 23. Alg. 2 for load sharing: Cloud server 2.

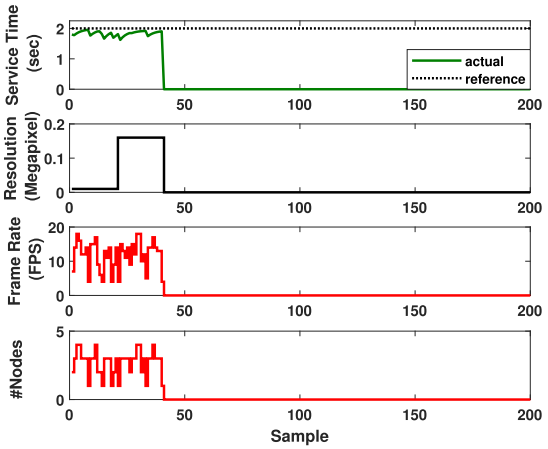


FIGURE 24. Alg. 2 for load sharing: Cloud server 3.

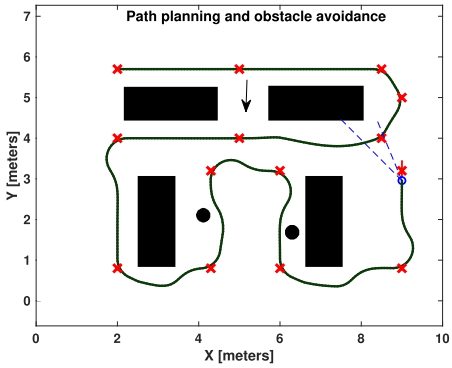


FIGURE 25. DDMR in application environment.

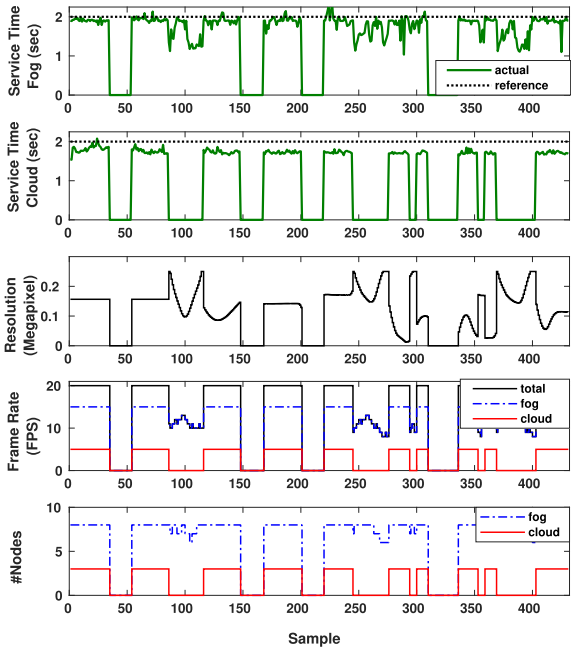


FIGURE 26. Load sharing: fog-cloud platform.

information related to the kinematic modeling of the mobile robot and navigation algorithms in Appendix A-A. The navigation algorithm comprises localization and path planning with obstacle avoidance (summarized in Table 1).

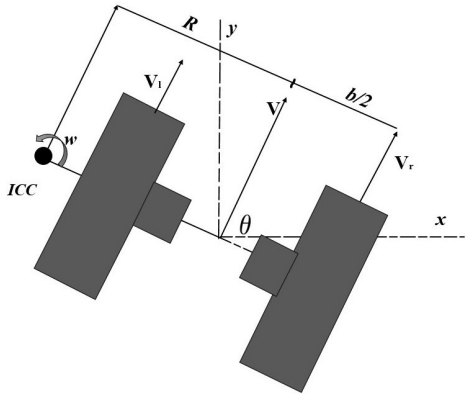


FIGURE 27. DDMR kinematics.

The objective of this work is to meet the service time requirement for object detection with time-varying MRVD while the robot navigates through the application environment. Hence, it has two independent algorithms, the auto-scaling algorithm and the navigation algorithm.

MRVD is generated while the robot navigates and captures the images of the items stacked in the racks.

Later, the image frames are sent to a load balancer to be distributed across the fog or an integrated fog-cloud platform for object detection. The distribution among fog and cloud is based on a heuristic algorithm, which is primarily based on frame rate as mentioned in Table 7. The auto-scaling controller decides the number of computing nodes needed to satisfy the service time requirements considering the MRVD characteristics.

The MPC validation plots are shown in Fig. 26. Image resolution is calculated based on the distance between the robot's center of gravity and the boundary of the application environment. (refer V-A) as in Fig. 26 c. Frame rate is chosen as in Fig. 26 d. Fig. 26 e shows the auto-scaling of the computing nodes to satisfy the service time requirements. The fog platform maintains the service time almost below the reference value of 2 *sec* (Fig. 26 a) when the frame rate is below 15 FPS. However, above the frame rate of 15 FPS, the cloud platform (serving frame overhead from the fog layer) takes over the processing of the MRVD with service time below the reference value of 2 *sec* (Fig. 26 b).

Note: Frame rate and image resolution are zero at certain time instants, which shows no image to detect at these instants (one such area is indicated by an arrow in Fig. 25), and consequently, the corresponding number of computing nodes and service time are zero.

VII. CONCLUSION AND FUTURE WORK

This paper presents a control-theoretic approach for developing an autonomic fog-cloud integrated computing platform for vision-based mobile robot applications. To accelerate the process of object detection in the application environment, the time-varying MRVD acquired during navigation is offloaded to an on-premise fog-cloud platform for hosting DL algorithms. The scalability feature of fog and cloud platforms is leveraged in this work.

The objective of the proposed research is to process the MRVD within the service time deadline by auto-scaling initially the fog computing nodes and later the cloud nodes on demand. To enforce this requirement, decentralized controllers using the LPV-MPC framework have been developed for the fog and the cloud platforms. Furthermore, a heuristic algorithm performs the distribution of MRVD between the fog and the cloud platforms. We conclude the paper by providing validation results for object detection during mobile robot navigation in the actual environment. Also, simulation results for scenarios assuming multiple fog and cloud layers have been provided.

For future work, we aim to enhance the current research further by incorporating the proposed approach in a multi-robot scenario for coordination control tasks. Multi-robot systems find essential applications in surveillance, underwater robotics, and space exploration. In this setting, the robot's sensing bandwidth might be constrained by the network bandwidth limiting the design of coordination strategies. From our perspective, an event-triggered control [29] can offer an acceptable solution and is a good starting point for future work.

APPENDIX A
DDMR NAVIGATION

A. MODELING A DDMR

A differential drive mobile robot, commonly known as a DDMR, features two independently driven wheels mounted on a common axis. By varying the relative velocity of the wheels, DDMR can navigate back and forth and also it can rotate about an axis. Different trajectories can be generated by altering the relative speed of the wheels, thus shifting the point of rotation, as demonstrated in [30]. The point about which the DDMR rotates is referred to as the ICC (Instantaneous Center of Curvature) (see Fig. 27). It is a point on the common axis of both wheels.

Both the wheels should have same angular velocity, ω about the ICC, at every instant of time. The resulting linear velocities are as follows:

$$\omega(R + \frac{b}{2}) = v_r; \quad \omega(R - \frac{b}{2}) = v_l \tag{18}$$

where v_r is the velocity of the right wheel, v_l is the velocity of the left wheel along the ground, b is the distance between the centers of the two wheels along the axle, and R is the distance between ICC and the midpoint of two wheels.

Solving for R and ω using (18) gives the following equation.

$$R = \frac{b}{2}\frac{(v_r + v_l)}{(v_r - v_l)}; \quad \omega = \frac{(v_r - v_l)}{b}. \tag{19}$$

1) FORWARD KINEMATIC MODEL

Forward kinematics is the study of the robot's location (x,y,θ) in reference to its control inputs (v_l, v_r). Consider the mobile robot positioned at (x,y) and at an angle θ to the x axis. The velocities v_l and v_r, as well as R and ω, are functions of time [30]. ICC can be obtained from (19) given control parameters as below:

$$ICC = [x - Rsin(\theta), y + Rcos(\theta)] \tag{20}$$

Let s_r and s_l be the distance traveled and $\delta\theta$ be the change in orientation angle over a small period of time, δt. $\delta\theta = \omega\delta t$; $s_l = v_l\delta t$; $s_r = v_r\delta t$ As a result, R can be expressed in terms of s_r and s_l as shown below:

$$R = \frac{b}{2}\frac{(s_r + s_l)}{(s_r - s_l)} \tag{21}$$

The robot should be rotated around ICC by $\omega\delta t$ to estimate its pose at $t + \delta t$. To do this, translate the ICC to the origin, rotate the robot around the origin by $\omega\delta t$, and then translate back to the ICC [9]. The robot's pose at time $t + \delta t$ is given by:

$$\begin{bmatrix} x_{t+\delta t} \\ y_{t+\delta t} \\ \theta_{t+\delta t} \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{22}$$

The state-space model of the DDMR as it rotates at a distance R from its ICC with an angular velocity of ω is shown in (22).

The system vector can be found from (20)-(22) given by:

$$\begin{aligned} \begin{bmatrix} x_{t+\delta t} \\ y_{t+\delta t} \\ \theta_{t+\delta t} \end{bmatrix} &= \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \frac{b}{2}\frac{s_r + s_l}{s_r - s_l}(sin(\theta + \delta\theta) - sin(\theta)) \\ \frac{b}{2}\frac{s_r + s_l}{s_r - s_l}(-cos(\theta + \delta\theta) + cos(\theta)) \\ \delta\theta \end{bmatrix} \\ &= \mathbf{f}(x_t, y_t, \theta_t, s_r, s_l) \end{aligned} \tag{23}$$

The non-linear model given by (23) can be linearised by the Jacobian linearization. By using the Jacobian of system vector, $\mathbf{f}$, the state transition matrix, $\mathbf{A_l}$ and input matrix $\mathbf{B_l}$ can be obtained. $\mathbf{C_l}$ is an identity matrix since all states are measurable [31].

B. LOCALIZATION AND PATH PLANNING

For navigation of the mobile robot towards the goal point, a pure pursuit controller is used in this paper. It can regulate the linear velocity of the wheels (left and right) and helps to estimate the angular velocity at which the mobile robot should move from the current pose to the final position through look-ahead points.

Simultaneous localization and mapping (SLAM) is the problem of updating the environmental map using sensors and concurrently getting localized in the updated map. In this work, this problem is addressed using an EKF based localization, where an extended Kalman filter is used as the estimator. The EKF algorithm [31] is executed based on the forward kinematic model of the robot to the estimate position of the mobile robot. For obstacle avoidance, the algorithm mentioned in [10] has been implemented.

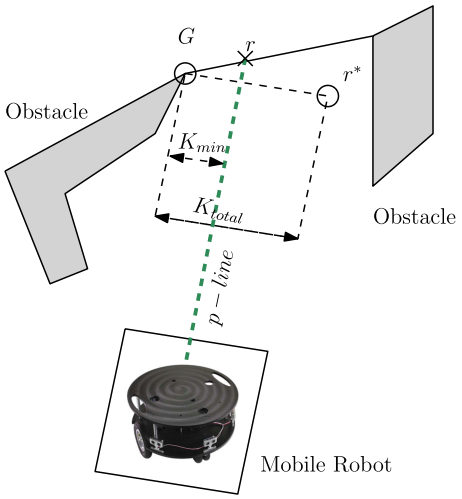


FIGURE 28. A situation for obstacle avoidance.

The mobile robot moves from the start point to the final point based on a bug-based path planning algorithm [9], a local path planning approach. The local path planning methods have limited information about the environment, which necessitates the robot to update its environment with the latest information provided by the sensors before planning each move toward the final position.

1) EKF BASED LOCALIZATION

In this paper, the position of the mobile robot is estimated using an EKF based localization. The forward kinematic model of the robot can be used with the prediction and correction equations of the EKF to precisely locate the robot [31] and is given by (24)-(25).

Prediction Steps:

$$\begin{aligned} \mathbf{x}_k^- &= f(\mathbf{x}_{k-1}, s_r, s_l) \\ \mathbf{P}_k^- &= \mathbf{A_l}\mathbf{P}_{k-1}\mathbf{A_l}^T + \mathbf{Q} \end{aligned} \tag{24}$$

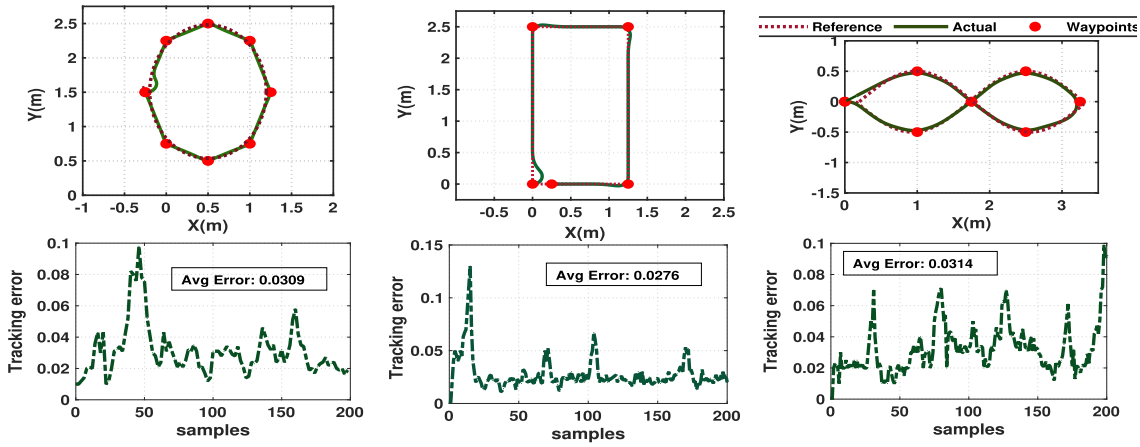
Correction Steps:

$$\begin{aligned} K_k &= \mathbf{P}_k^-\mathbf{C_l}^T(\mathbf{C_l}\mathbf{P}_k^-\mathbf{C_l}^T + \mathbf{R})^{-1} \\ \hat{\mathbf{x}}_k &= \mathbf{x}_k^- + K_k(\mathbf{y}_k - \mathbf{C_l}\mathbf{x}_k^-) \\ \mathbf{P}_k &= (1 - K_k\mathbf{C_l})\mathbf{P}_k^- \end{aligned} \tag{25}$$

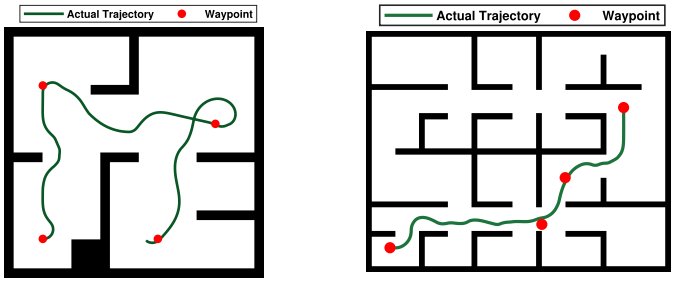
The measurement error covariance matrix, $\mathbf{R}$ and the process error covariance matrix, $\mathbf{Q}$ can be considered as diagonal matrices since the errors for each states are uncorrelated.

$$Q = \begin{bmatrix} w_x^2 & 0 & 0 \\ 0 & w_y^2 & 0 \\ 0 & 0 & w_\theta^2 \end{bmatrix}; \quad R = \begin{bmatrix} v_x^2 & 0 & 0 \\ 0 & v_y^2 & 0 \\ 0 & 0 & v_\theta^2 \end{bmatrix}$$

where w_x, w_y, w_θ are process error covariance matrices and v_x, v_y, v_θ are measurement error covariance matrices for x, y and θ respectively [10].



(a) Path following: Elliptical Path (b) Path following: Square Path (c) Path following: Eight shaped path



(d) Obstacle avoidance: case 1 (e) Obstacle avoidance: case 2

FIGURE 29. **Path planning with obstacle avoidance of mobile robot.**

2) PATH PLANNING WITH OBSTACLE AVOIDANCE

In this work, for collision-free navigation of the mobile robot, the collision avoidance algorithm in [10] is used. The grey blocks in Fig. 28 represent the obstacles in the path of the mobile robot. and the black cross mark shows the local target, r, measured by minimising the cost function [9]:

$$C(P_i) = d(P_i - P_r) + D(P_i) \tag{26}$$

where P_r is the position vector of the robot, and P_i is the i_{th} grid point on the line that passes across the gaps that were identified (determined by using the gradient of distance vector received from LIDAR and obstacles are indicated by discontinuous points). $D(P_i)$ is the euclidean distance transform (EDT) [32] for P_i, and $d(P_i)$ is the euclidean distance between the robot and P_i. The objective of the work is to locate P_i such that the cost function, $C(P_i)$ is minimized. This grid point is chosen as the local target. The process of creating a continuous path from start to destination using these local targets is known as local path planning.

The p-line in Fig. 28 is the line that connects the mobile robot's center of gravity to the local target, r. The obstacle avoidance algorithm considers obstacle points received from LIDAR data that are closer to the local target. The critical collision point, G, is the point that is closest to the p-line. The algorithm updates the goal, r as r^*, if the distance between G and the p-line, K_{min}, is smaller than the threshold value, K_{total} (depends on the kinematic model and size of the robot). The selection of the new target point, r^*, ensures that it is K_{total} from G and that the line connecting G and r^* is perpendicular to the p-line [10].

TABLE 8. **Navigation algorithm parameters.**

Pure Pursuit controller	**Parameters**
Max angular velocity (rad/s)	1.3
Desired linear velocity (m/s)	0.74
Look ahead distance (m)	0.36
Obstacle avoidance	**Parameters**
Minimum turning radius (m)	0.048
Distance limits (m)	[0.048 3]
Thresholds (m)	[5 10]
Number of angular sectors	35
Safety distance (m)	0.37

The parameters related to mobile robot navigation are summarized in Table 8. The results for path following and obstacle avoidance for different cases are shown in Fig. 29.

REFERENCES

[1] M. B. Alatise and G. P. Hancke, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39830–39846, 2020, doi: 10.1109/ACCESS.2020.2975643.

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. London, U.K.: Pearson, 2002.

[3] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.
[4] A. Asaduzzaman, A. Martinez, and A. Sepehri, "A time-efficient image processing algorithm for multicore/manycore parallel computing," in *Proc. SoutheastCon*, 2015, pp. 1–5, doi: 10.1109/SECON.2015.7132924.
[5] H. M. B. Priyabhashana and K. P. N. Jayasena, "Data analytics with deep neural networks in fog computing using TensorFlow and Google cloud platform," in *Proc. 14th Conf. Ind. Inf. Syst. (ICIIS)*, Dec. 2019, pp. 34–39, doi: 10.1109/ICIIS47346.2019.9063284.
[6] Amazon. *Deep Learning on AWS*. Accessed: Aug. 5, 2023. [Online]. Available: https://aws.amazon.com/deep-learning
[7] D. Vinod and P. S. SaiKrishna, "Development of an autonomous fog computing platform using control-theoretic approach for robot-vision applications," *Robot. Auto. Syst.*, vol. 155, Sep. 2022, Art. no. 104158, doi: 10.1016/J.ROBOT.2022.104158.
[8] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM algorithm," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 3562–3568.
[9] S. Karakaya, H. Ocak, and G. Kucukyildiz, "A bug-based local path planning method for static and dynamic environments," in *Proc. Int. Symp. Innov. Technol. Eng. Sci.*, 2015, pp. 846–855.
[10] S. Karakaya, G. Kucukyildiz, and H. Ocak, "A new mobile robot toolbox for MATLAB," *J. Intell. Robot. Syst.*, vol. 87, no. 1, pp. 125–140, Jul. 2017.
[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. ECCV*, vol. 9905. Cham, Switzerland: Springer, Dec. 2016, pp. 21–37.
[12] G. N. Desouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 2, pp. 237–267, Feb. 2002, doi: 10.1109/34.982903.
[13] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3296–3297, doi: 10.1109/CVPR.2017.351.
[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
[15] H. Ahn, "A function as a service based fog robotic system for cognitive robots," *Appl. Sci.*, vol. 9, no. 21, p. 4555, Oct. 2019, doi: 10.3390/APP9214555.
[16] L. Gherardi, D. Hunziker, and G. Mohanarajah, "A software product line approach for configuring cloud robotics applications," in *Proc. IEEE 7th Int. Conf. Cloud Comput.*, Jun. 2014, pp. 745–752.
[17] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 4559–4566.
[18] J. Ichnowski, J. Prins, and R. Alterovitz, "Cloud-based motion plan computation for power-constrained robots," in *Algorithmic Foundations of Robotics XII*, vol. 13. Cham, Switzerland: Springer, 2000, pp. 96–111.
[19] K. Obayashi and S. Masuyama, "Pilot and feasibility study on elderly support services using communicative robots and monitoring sensors integrated with cloud robotics," *Clin. Therapeutics*, vol. 42, no. 2, pp. 364–371, Feb. 2020.
[20] B. A. Erol, S. Vaishnav, J. D. Labrado, P. Benavidez, and M. Jamshidi, "Cloud-based control and vSLAM through cooperative mapping and localization," in *Proc. World Automat. Congr. (WAC)*, 2016, pp. 1–6, doi: 10.1109/WAC.2016.7582999.
[21] S. Tuli, N. Basumatary, and R. Buyya, "EdgeLens: Deep learning based object detection in integrated IoT, fog and cloud computing environments," in *Proc. 4th Int. Conf. Inf. Syst. Comput. Netw. (ISCON)*, Nov. 2019, pp. 496–502.
[22] V. K. Sarker, J. Peña Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Offloading SLAM for indoor mobile robots with edge-fog-cloud computing," in *Proc. 1st Int. Conf. Adv. Sci., Eng. Robot. Technol. (ICASERT)*, May 2019, pp. 1–6.
[23] P. S. Saikrishna, R. Pasumarthy, and N. P. Bhatt, "Identification and multivariable gain-scheduling control for cloud computing systems," *IEEE Trans. Control Syst. Technol.*, vol. 25, no. 3, pp. 792–807, May 2017, doi: 10.1109/TCST.2016.2580659.
[24] A. K. Tangirala, *Principles of System Identification: Theory and Practice*, 1st ed. Boca Raton, FL, USA: CRC Press, 2014.
[25] T. Besselmann, J. Lofberg, and M. Morari, "Explicit MPC for LPV systems: Stability and optimality," *IEEE Trans. Autom. Control*, vol. 57, no. 9, pp. 2322–2332, Sep. 2012, doi: 10.1109/TAC.2012.2187400.
[26] R. Shanmugamani, *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras*. Birmingham, U.K.: Packt Publishing, 2018.
[27] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "A control approach for performance of big data systems," in *Proc. 19th IFAC World Congr.*, Le Cap, South Africa, 2014, pp. 1–12.
[28] E. Carlos Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989.
[29] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Trans. Autom. Control*, vol. 52, no. 9, pp. 1680–1685, Sep. 2007.
[30] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, doi: 10.1017/CBO9780511780929.
[31] S. Karakaya, H. Ocak, G. Kucukyildiz, and O. Kilinc, "A hybrid indoor localization system based on infra-red imaging and odometry," in *Proc. Int. Conf. Image Process., Comput. Vis., Pattern Recognit. (IPCV)*, 2015, p. 224.
[32] J. C. Elizondo-Leal, E. F. Parra-González, and J. G. Ramírez-Torres, "The exact Euclidean distance transform: A new algorithm for universal path planning," *Int. J. Adv. Robot. Syst.*, vol. 10, no. 6, p. 266, Jun. 2013, doi: 10.5772/56581.

DINSHA VINOD received the master's degree in control systems from the College of Engineering Trivandrum, in 2017. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Indian Institute of Technology Tirupati, Tirupati, India. Her research interests include systems and control, robotics, computer vision, and distributed computing systems.

DURGESH SINGH received the bachelor's degree in electrical engineering from the Birsa Institute of Technology Sindri, India, in 2015, and the Ph.D. degree in systems and control from the Indian Institute of Technology Madras, Chennai, India. His current research interests include system identification and performance control of computing systems, design of predictive control, and stability analysis of cyber-physical systems.

P. S. SAIKRISHNA received the Ph.D. degree in systems and control from the Indian Institute of Technology Madras, Chennai, India. He is currently an Assistant Professor with the Department of Electrical Engineering, Indian Institute of Technology Tirupati, Tirupati, India. His current research interests include autonomic cloud computing, distributed control of large-scale systems, and the development of low-cost training simulators for process plants.