Received 13 October 2022, accepted 11 November 2022, date of publication 18 November 2022, date of current version 23 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3223446

APPLIED RESEARCH

E-CNMPC: Edge-Based Centralized Nonlinear Model Predictive Control for Multiagent Robotic Systems

ACHILLEAS SANTI SEISA, (Student Member, IEEE), BJÖRN LINDQVIST, SUMEET GAJANAN SATPUTE, (Member, IEEE), AND GEORGE NIKOLAKOPOULOS, (Member, IEEE)
Robotics and AI Team, Department of Computer, Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden

Corresponding author: Achilleas Santi Seisa (achsei@ltu.se)

This work was supported by the European Union's Horizon 2020 Research and Innovation Program through the Marie Skłodowska-Curie under Agreement 953454.

ABSTRACT With the wide deployment of autonomous multi-agent robotic systems, control solutions based on centralized algorithms have been developed. Even though these centralized algorithms can optimize the performance of the multi-agent robotic systems, they require a lot of computational effort, and a centralized unit to undertake the entire process. Yet, many robotic platforms like some ground robots and even more, aerial robots, do not have the computing capacity to execute this kind of frameworks on their onboard computers. While cloud computing has been used as a solution for offloading computationally demanding robotic applications, from the robots to the cloud servers, the latency they introduce to the system has made them unsuitable for time sensitive applications. To overcome these challenges, this article promotes an Edge computing-based Centralized Nonlinear Model Predictive Control (E-CNMPC) framework to control, and optimize, in swarm formation, the trajectory of multiple ground robotic agents, while taking under consideration potential collisions. The data processing procedure for the time critical application of controlling the robots in a centralized manner, is offloaded to the edge machine, thus the framework benefits from the provided edge resources, features, and centralized optimal performance, while the latency remains bounded in desired values. Besides, real experiments were conducted as a proof-of-concept of the proposed framework to evaluate the system's performance and effectiveness.

INDEX TERMS Edge-based centralized nonlinear model predictive control (E-CNMPC), edge computing, Kubernetes, robotics.

I. INTRODUCTION

Many key technologies have emerged over the recent years to encounter the challenges of massive data production and processing. From cloud and edge computing for data storage and processing to 4G/LTE and 5G networks for low latency, high bandwidth, data transferring [1], [2], [3]. Researchers seek to take advantage of these technologies and integrate them to their robotic applications [4], [5], [6], [7], [8], [9], [10], [11]. Edge computing - 5G enabled robotic applications will provide the advantages of computational resources, features for scalable, automated and self-healing applications, and low latency. Through edge-based architectures [12], robots will be able to communicate, exchange information and cooperate with each other to execute complex tasks in optimal ways. In that context, centralized schemes will play an important role to ensure the optimization of these multi-agent systems, while edge computing technologies will manage the computational workload over powerful clusters.

Cloud and edge computing have been used for numerous different robotic applications, based on Robotic Operating System (ROS), like in [13], [14], and [15] or in [16], where researchers developed a system for a ground robot to autonomously navigate inside data center rooms and collect useful measurements. The different processes and the Graphical User Interface (GUI) for the measurement visualization

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

are handled by the cloud. Mainly, cloud and edge have been studied for offloading computationally intensive tasks, such as simultaneous localization and mapping (SLAM). That is the subject in [17], [18], [19], and [20], where researchers created a framework to expand the ROS environment so it would be easy for users to offload the SLAM algorithms to the edge servers [17], and architectures for a multi-ground-robot [18], [19] or single-robot [20] edge-based SLAM mechanism. While the previous works focused on the SLAM problem, [21] and [22] asses the problem of deep learning based visual odometry and the problem of path planning and localization, respectively, for mobile robots through the edge. In [23], a cloud-based framework is presented for Unmanned Aerial Vehicles (UAVs), where the resourced-constrained UAVs operates as a client and connects to the cloud servers to access information about their mission, thus it can overcome the limited computational processing capabilities. In [24], a swarm of UAVs has access to a mobile edge server to offload the computation tasks based on a deep reinforcement learning model, while respecting latency requirements. On the other hand, a powerful UAV has been deployed, as an assisted edge computing node, in [25] and [26]. In these cases, computation procedures can be offloaded from ground users to the UAV edge node or, if the UAV cannot handle the process, it operates as a relay to offload the task to the ground base station. The researchers also took under consideration a communication strategy to minimize the total energy consumption.

Even though some studies like [27], where a cloud-based formation control was applied for multiple robots, none of the above works are focused on the centralized or distributed control properties that cloud and edge can offer. Model Predictive controller (MPC) is a complex and advanced method to control a system while satisfying a set of constraints. It uses optimization methods, in fixed and bounded time instances, which can have high complexity. MPC schemes, which are the base of the proposed framework, have been assisted by edge or cloud in several cases like [28], [29], [30], [31], [32], [33], [34], and [35]. All these works though, investigate the control system for only one agent as in [30] and [32], where we introduced architectures to control an UAV in a simulation and real-world environment, respectively. For multi-agent systems, distributed or centralized frameworks should be implemented. A detailed analysis for centralized and distributed control schemes for UAV swarms based on cloud and edge computing is presented in [36]. In this work, only separately systems are studied for UAV swarms task allocation missions. Either a cloud system is utilized for the centralized control system or edge servers are used. The centralized schemes clearly outperform the distributed one when the number of agents that form the swarm is small. However, when the number of agents is large enough, the centralized framework introduces major scheduling latencies, while for the distributed framework, the scheduling latency maintains the same. A hybrid framework was not investigated but the authors suggest that it could probably exceed the performance of the two studied schemes. Distributed approaches were introduced in [37] and [38] to achieve optimized behavior but at the same time reduced computational complexity for the cooperation of multiple UAVs. In [39], a decentralized control scheme based on leader-follower formation for an UAV swarm system was presented. In [40] and [41], distributed Nonlinear Model Predictive Control (NMPC) for collision avoidance between multiple aerial agents was presented, while in [42] distributed and centralized MPC frameworks were evaluated for cooperative motion of multiple UAVs. In [43], authors proposed and compared centralized and decentralized MPC formulations for controlling the UAVs' maneuvers when carrying a payload. In [44], a strategy with centralized and distributed MPC algorithms was addressing the problem of controlling platoons with both autonomous and human-driven vehicles, and in [45] distributed and centralized MPC formulations are also studied for tracking multiple targets using a swarm of UAVs. Centralized schemes were proposed in [46], [47], and [48], for controlling platoons based on leader's behavior, for placing UAVs to form a mesh network, and for preventing potential collisions between multiple UAVs navigating in a narrow area, respectively.

Another important part of this work, is the integration of cloud services and technologies to our framework, to make our application scalable and robust. Containerized applications should be implemented in a way to minimize edge latency and enable resource provision as in [49]. Afterwards, a proper orchestrator should be used. Thus, it has been proposed the utilization of microk8s which is a lightweight container platform comparable to kubernetes [50]. In [51], authors suggested an infrastructure to connect the cloud to the edge, and expand the cloud services and kubernetes management to the edge through a proposed network protocol, while in [52], a remote controller was implemented in a form of a docker containerized application and was running through a mobile edge server. In [53], an automated process based on stochastic processes and implemented through kubernetes for optimizing the distribution of containers to the cloud, edge and fog, was introduced. A virtual robotics lab, based on a kubernetes cluster implementation, was presented in [54], for students to access it, get familiar with it and develop their robotic applications. Finally, a framework based on docker, kubernetes and ROS for monitoring containerized robot tasks, was proposed in [55] and a mechanism for deploying robotic containerized application to the edge and the cloud was presented in [56].

The motivation behind this work was to control multiple agents in an optimized manner, while keeping their swarm behavior. To the best of our knowledge, the existing frameworks rely on distributed methods because the needed computational effort for centralized methods is drastically increasing with respect to the increase in the number of agents. Thus, robots' onboard computers cannot execute these centralized controllers since the controller fails under resource constraints, like the limited Central Processing Unit (CPU) and the Random-Access Memory (RAM) capacity. On the other hand, centralized control schemes seems to

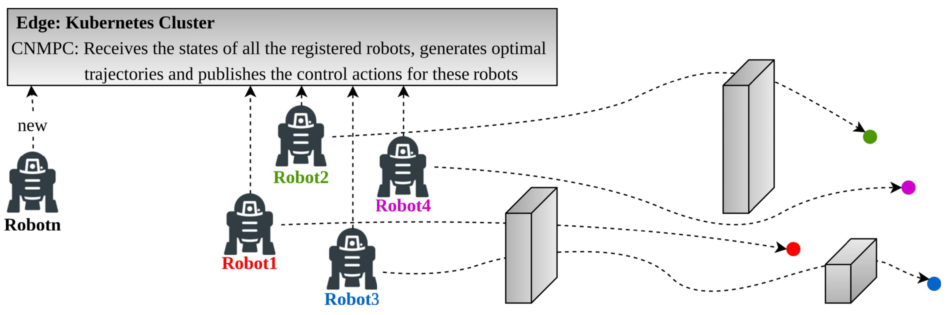
Edge: Kubernetes Cluster
CNMPC: Receives the states of all the registered robots, generates optimal trajectories and publishes the control actions for these robots
new
Robotn
Robot1
Robot2
Robot3
Robot4

FIGURE 1. Concept of edge computing-based centralized control scheme for multiple ground robots. The robots send their states, generated by their sensors, to the edge where the E-CNMPC is executed, and they are receiving commands from the edge to navigate into the environment, avoid collisions, and reach desired points (colour dots) and/or behaviors (swarm).

have enhanced performance as quoted in [36], but need to be executed in powerful external centralized units. The improved centralized performance can be in fact, of great importance for our application because controlling multiple agents is a time critical task and we must ensure that potential collisions will be prevented. In the proposed work, the parameters of the optimization problem, which significantly increase the complexity, are depended on the number of agents, the prediction horizon, the initial tolerance, the hard constrains for collision avoidance and swarm behavior. To overcome the drawback of computational limitations, our proposed framework is based on edge computing solutions by utilizing an edge machine. Suggested solutions based on cloud computing cannot be applied for controlling robots in real-time, since the system will fail due to the introduced latency. The novel proposed E-CNMPC framework is offloaded to an edge machine that hosts a kubernetes cluster. The kubernetes cluster and the ground robots exchange messages for the operation of navigating the robots in the environment in a swarm behavior and at the same time prevent any potential collisions, while this concept is depicted in Fig. 1. The edge machine is located in physical proximity to ground robots and therefore, remarkably shortens data communication distance, lowers offloading transmission delay, and allows the advanced quality of kubernetes services.

The proof-of-concept experimental analysis corroborates the feasibility and efficiency of the proposed framework, while demonstrating the advantages of the edge-based system. Thus, several key contributions can be highlighted.

The initial contribution stems from the fact that the proposed E-CNMPC framework is based on a novel robot-edge architecture to enable relatively real-time trajectory control for a multi-agent system. In the proposed approach, each agent can offload their states to the edge so they do not need to run the computationally demanding MPC on their onboard computer. Thus, we can secure the availability of sufficient resources for the execution of the controller, even when we want to change the values of some MPC parameters, such as the MPC prediction horizon, that improve the behavior of the system but on the other hand increase its complexity. Moreover, edge kubernetes cluster can give the capability of assigning resources for the application, thus we can request more resources on demand, which is of great importance for a centralized scheme, where the number of agents and several control parameters, might vary, thus the required resources will vary as well.

The second contribution concerns the novel architecture of the proposed framework that it is utilizing technologies such as containers and orchestrators that enable novel capabilities from a control systems approach, as scalability, robustness, management and overall resiliency. The proposed controller and the related architecture can be fast and easily deployed and redeployed, in case of failure, in any external edge machine. In addition, the proposed control architecture is introducing a novel concept in edge based closed loop systems as it is running in a form of containerized application, thus it does not have software dependencies from the host machine for instance the robot's onboard computer or the users operating system. Users will just have to connect to the edge machine to control the robotic fleet, from any device, without having to carry computational heavy devices, and will not have to worry whether their computer or device can handle the execution of the application.

By investigating the current state-of-art, we realized that the existing multi-agent systems are based on distributed control schemes that fall behind in terms of performance. Our centralized optimization framework considers all the ground agents and solves the optimization problem for the entire system online and in an edge architecture orientation that forms the third contribution. As such the E-CNMPC framework is able to generate control actions for the trajectories of every agent, in an optimal manner and to optimizes the behavior of the entire system as in [48]. The designed framework differs from [48], in terms of architecture since our framework is based on a total novel robot-edge architecture and real-life experiments.

Finally, the last contribution is related to the comparison of the E-CNMPC framework to [48]. In this case we introduce a system that provides a swarm-kind behavior for the cooperation of the agents. Even though most swarm systems that are based on classical approaches, such as the leader-follower, the proposed E-CNMPC guarantees the swarm behavior through hard constrains. Minimum and maximum distances have been considered, thus the agents would always try to keep the swarm behavior while navigating in an area and avoiding collision between each other and the surrounding environment. In Fig. 2 this concept is demonstrated, where r_{swarm} represents the radius for maximum allowed distance between agents, and r_{safe} the minimum allowed distance between agents. The E-CNMPC takes care that these constrains would not be violated, or if they do violated, the optimizer will generate trajectories for all the ground robots, in order to move accordingly and respect again the constrains.

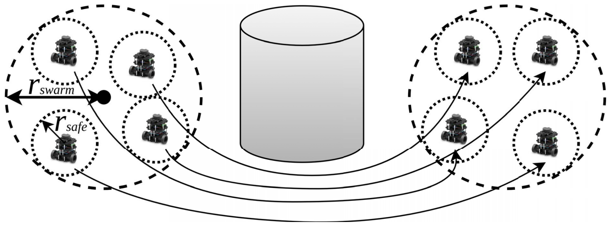

FIGURE 2. The E-CNMPC generates control actions for the ground robots in an effort to respect the constrains. The constrains are depending on the maximum distance between the agents in order to keep the swarm behavior and the minimum distance between the agents with each other and the surrounding environment in order to avoid any type of collision.

The rest of the article is divided into discrete sections and subsections to present and describe the different components of the proposed framework, as well the results of the experimental setup. The developed framework is presented in Section II. This Section is divided into several parts in an effort to address in detail the main implementations, the developments and the multiple proposed components of this work. In Section III, we demonstrate the experimental setup for the multi-agent system and we quote representative results for each study case. Lastly, we end this work with Section IV, where we state our conclusions and discuss future directions.

II. CENTRALIZED CONTROL FRAMEWORK

A. MULTIPLE AGENTS

The concept of this work is to control multiple agents through the proposed edge-based centralized framework as depicted in Fig. 3. In an effort to achieve this task, all the agents have to send their states to the edge machine, where the centralized controller is offloaded and running. Despite the spatially distributed nature of the multi-agent system, the states are sent to the edge as a unified flow of data collection to satisfy the input requirements of the system. Then, the edge-based centralized control framework calculates and generates control action for each agent. In that way, the control framework can optimize the performance of the entire system by generating trajectories for each agent, while taking into account the trajectories of the other agents and potential collisions. In addition, by setting an increased prediction horizon (60 steps or greater) for the E-CNMPC, the framework can make more accurate predictions for the future trajectories of the agents, thus, it can produce smoother and safer trajectories for every agent. This optimal behavior though, comes with computational cost, which our framework was able to overcome since it integrates edge computing resources and services.

The right part of Fig.3 depicts the closed looped system. The states that need to be sent to the edge for the execution of the controller are the position and orientation of each robot, $x_1(k), x_2(k), \ldots, x_n(k)$. Since these states do not arrive at the edge at the exact time that they are generated, the states that arrive as the inputs to the controller are delayed, hence are represented as $x_1(k-d_1), x_2(k-d_1), \ldots, x_n(k-d_1)$. The parameter d_1 expresses the robot to edge travel time delay and depends on the network characteristics, as well to the ROS publishing/subscribing delays and kubernetes message forwarding delays. Once the controller receives the states of the agents, it generates control action denoted as $u_1(k-d_2), u_2(k-d_2), \ldots, u_n(k-d_2)$. These control actions are the control commands forward/backward velocities and angular rates for each robot. Due to the execution time of the controller, which can be comparable to the travel time delays, we introduced another parameter specified as d_2. This parameter is the sum of the d_1 plus the execution time delay of the controller. Yet again, the control commands need to be sent through the network from the edge cluster to the robots. To express this delay, the input commands arriving to the robots are denoted as $u_1(k-d_3), u_2(k-d_3), \ldots, u_n(k-d_3)$, where d_3 describes the travel time delay from the edge to the robot plus the delays d_2. The last parameters of the closed loop system are the outputs $y_1(k), y_2(k), \ldots y_n(k)$.

B. ROBOTIC OPERATING SYSTEM

ROS is the fundamental software that researchers use to develop their robotic applications. Due to its friendly interface and the convenience it provides when it comes to coding experience, it is universally accepted. On the other contrary, on the communication level, it introduces some challenges when integrated with technologies, such as containers and kubernetes. ROS handles the source code and manages the communication into topics through publishers and subscribers. All the robotic agents are running ROS nodes that need to publish their states to their own odometry topic and subscribe to the command topic to receive their control actions. On the edge side, a CNMPC ROS node is running to execute the controller and it needs to subscribe to every odometry topic to receive the states of every robot and publish the control action for each agent to the corresponding command topic. When different ROS nodes need to communicate to exchange data through a topic, these nodes first need to register to the same ROS master. Then, the master opens a random socket and assigns the nodes to communicate through that socket by publishing and subscribing to the topic. The way ROS master opens ports for communication is random and the ports are different each time the nodes need to communicate. However, kubernetes by default, only allows some ports to be exposed for communication. In this work, because some ROS nodes are running on the edge side of the architecture, where they are deployed inside the kubernetes cluster, and

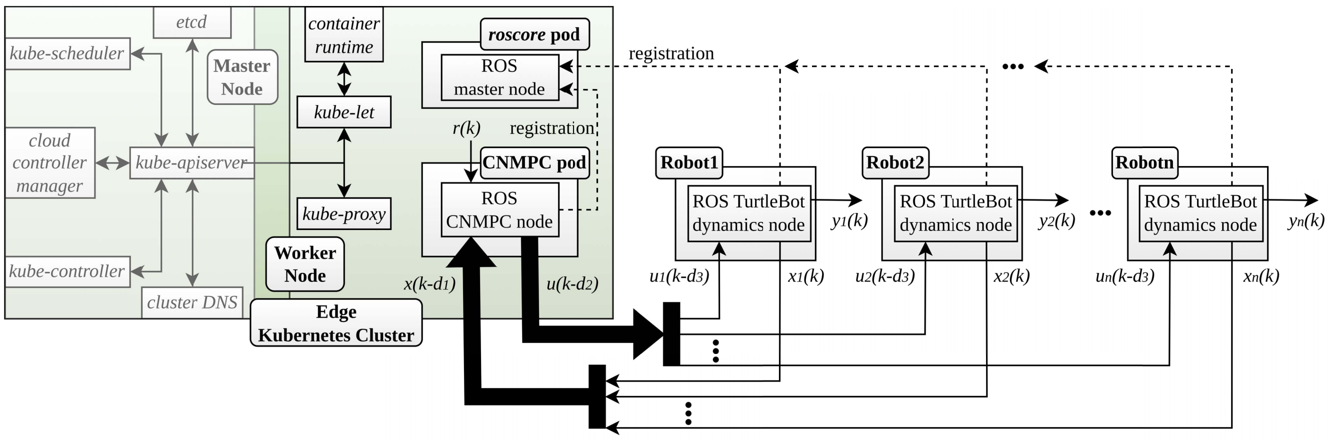

FIGURE 3. Block diagram of CNMPC framework including the kubernetes architecture for controlling multiple ground robots.

others are running on the robots, the kubernetes cluster should allow all the ports to be exposed. To overcome this challenge, when deploying the kubernetes pods, we choose the host Network option to expose all the ports of the edge machine to all the pods and vice versa. By doing that, we allow all the data that arrive to the edge machine to be forward inside the pods.

C. DOCKER IMAGES

For the CNMPC framework, we had to create images for each different task. We used docker to develop our customized images, which were two in total. For both images, we used ROS entrypoint and ROS noetic environment, running on Ubuntu 20.04. The main image is the one running the CNMPC execution. We deployed the image with all the necessary libraries and dependencies as well as the needed packages for the execution of the CNMPC. When we run the kubernetes pod created by that docker image, the UAVs register to the CNMPC, and the operator get the options discussed in Section II-A. The other image hosts the ROS master and is responsible for executing the `roscore`. By developing and using docker containers, our application does not depend on software and operating system dependencies of the host unit, thus the application can run in any environment.

D. KUBERNETES

Kubernetes is responsible for managing the procedure regarding the containerized applications. To achieve that, kubernetes consists of many different components, each one of them handling a different task. These components are running either on the master node, which is the control panel or "brain" of kubernetes, or the worker nodes which host the applications as depicted in Fig. 3. On the master node the main components are the controller manager (`kube-controller`), which runs all build-in controllers, like node or replication controller, the scheduler (`kube-scheduler`) that distributes unscheduled workloads across the available worker nodes, the API server (`kube-apiserver`), which is tracking the state of all cluster components and it is managing interactions between them and finally, the `etcd` which is the key value store for all cluster configuration data. Optional components on the master node, that can be useful for the edge cluster operation are: the cluster DNS, which provides in-`cluster DNS` for pods and services and the `cloud controller manager`, which runs cloud controller processes that take care of tasks, such as node auto scaling, creating DNS entries, etc. The `kube-controller`, `kube-scheduler`, `cluster DNS`, and `cloud controller manager` watch for changes related to their tasks and register them to the `kube-apiserver`, which reads and writes data from/to the `etcd`. The worker node on the other hand consists mainly of the application's components. These are the `kube-proxy`, which accepts and controls network connections to the node's pods, the `kube-let`, which manages containers based on incoming pod specifications and uses `container runtime` that implements the `CRI`, and finally, the pods which host the application's several tasks.

E. CENTRALIZED NONLINEAR MODEL PREDICTIVE CONTROL SCHEME

The base CNMPC module that is combined with the edge framework is based on the preliminary work in [48]. In this work, we apply the framework on a multi-agent set-up of ground robots as opposed to aerial vehicles and we transform it into a novel edge based architecture. The ground robots are non-holonomic and as such pose an interesting receding-horizon problem where the predictive nature of the CNMPC scheme can shine in orchestrating trajectories for all agents in the system while considering the kinematic constraints of their movements. The ground robots are described by a simple nonlinear kinematic model as:

$$\begin{aligned} \dot{p}_x(t) &= \cos\psi(t)u_v(t) \\ \dot{p}_y(t) &= \sin\psi(t)u_v(t) \\ \dot{\psi}(t) &= u_\omega(t) \end{aligned} \tag{1a}$$

where $[p_x, p_y]$ describes the position states and ψ is the heading state. It is assumed that the ground robot takes input commands in the form of a forward/backward velocity u_v and an angular rate command as u_w. We describe each agents' states as $x^{(i)} = [p_x^{(i)}, p_y^{(i)}]$, $\psi^{(i)}$ and inputs $u^{(i)} = [u_v^{(i)}, u_w^{(i)}]$, where $i = 1 \dots N_a$, and N_a denotes the total number of agents in the system. In the centralized scheme, the states are collected into an augmented model as $x = [x^1, x^2 \dots x^{N_a}]$ and similarly we define the total control actions as $u = [u^1, u^2 \dots u^{N_a}]$. The entire system dynamics can then be discretized with a sampling time T_s using the forward Euler to obtain the predictive form $x_{k+1} = \zeta(x_k, u_k)$. The CNMPC problem is solved with a receding horizon, where we denote the prediction horizon as N and denote predicted time steps with $k + j|k$ denoting a discrete prediction j steps into the future produced at time k. From this, we can describe the full vectors of predicted states and control inputs along the horizon, and for all agents, as $\boldsymbol{x}_k = (x^{(i)}_{k+j|k})_{j,i}$ and $\boldsymbol{u}_k = (u^{(i)}_{k+j|k})_{j,i}$ respectively.

We form a cost function for the complete centralized system that penalized deviations from a state reference, while minimizing the inputs' actuation and change in actuation from one time step to the next as:

$$\begin{aligned}
&J(\boldsymbol{x}_k, \boldsymbol{u}_k; u_{k-1|k}) \\
&= \sum_{j=0}^{N} \sum_{i=1}^{N_a} \|p^{(i)}_{\text{ref}} - p^{(i)}_{k+j|k}\|^2_{Q_p} \\
&\quad + Q_\psi(-\cos(\psi^{(i)}_{\text{ref}} - \psi^{(i)}_{\text{ref}}) + 1) + \|u_{\text{ref}} - u^{(i)}_{k+j|k}\|^2_{Q_u} \\
&\quad + \|u^{(i)}_{k+j|k} - u^{(i)}_{k+j-1|k}\|^2_{Q_{\Delta u}},
\end{aligned} \tag{2}$$

where the cost matrices for various states, inputs and input rates respectively are denoted as $Q_p \in \mathbb{R}^{2\times 2}$, $Q_\psi \in \mathbb{R}$, $Q_u, Q_{\Delta u} \in \mathbb{R}^{2\times 2}$. While the other terms follow the classical quadratic penalties, the penalty on the heading state stands out as strange. The motivation is quite simple: the centralized scheme requires that all agents share the same coordinate frame for their states, and as such we need to properly capture the 2π-modularity of the heading state ψ as to avoid the discontinuity at $\psi \sim \pm\pi$. Although more computationally complex, the utilized solver [57] in combination with the assistance from the Edge offloading, has no problems with it.

To enforce collision avoidance and swarm behavior, we impose set-exclusion constraints [58] on the available position space of each agent in the system. First, the most critical component in any multi-agent scheme is to avoid agent-agent collisions. As such, we can form a constraint for each pair of agents l, i as:

$$\begin{aligned}
C^{l,i}_{\text{safe}}(\boldsymbol{x}_k) := [r^2_{\text{safe}} &- (p^{(i)}_{x,k+j|k} - p^{(l)}_{x,k+j|k})^2 \\
&- (p^{(i)}_{y,k+j|k} - p^{(l)}_{y,k+j|k})^2]_+ = 0,
\end{aligned} \tag{3}$$

where we use the $max(a, b) = h[a, b]_+$ operator to form an expression such that $C^{l,i} = 0$ implies that the constraint is satisfied, e.g., we can write it as an equality constraint. The result is that each agent is commanded to be at least a distance of r_{safe} from each other. Let us also form an additional $C_{\text{circle}}(\boldsymbol{x}_k, p_{\text{obs}}, r_{\text{obs}})$ as a general static circle-type obstacle that all agents should avoid using the same kind of "circle"-expression.

We also pose a similar constraint to set a maximum distance among agents in the system as:

$$\begin{aligned}
C^{l,i}_{\text{swarm}}(\boldsymbol{x}_k) := [-r^2_{\text{swarm}} &+ (p^{(i)}_{x,k+j|k} - p^{(l)}_{x,k+j|k})^2 \\
&+ (p^{(i)}_{y,k+j|k} - p^{(l)}_{y,k+j|k})^2]_+ = 0,
\end{aligned} \tag{4}$$

that sets a maximum allowed distance between agents in the system defined by r_{swarm}. We should note that these constraints are also imposed along the prediction horizon at all predicted time steps for all agents. Classically, in swarm robotics these terms are handled as competing potentials or costs but, in this research, we will impose these "attractive" and "repulsive" terms as hard bounds, while letting the agents move freely as long as those two conditions are met (we will for example, in Section III, set $Q_p, Q_\psi = 0$ for all agents except one forming a leader-follower set-up). In this way, the grouping and agent-agent safety terms of the swarm dynamics are implicitly formed. This results in a very high number of constraints, and for a large horizon it becomes an incredibly complex CNMPC problem to be solved. We fully utilize the edge computation offloading in order to be able to solve an optimization problem with such a high number of constraints. The resulting CNMPC problem is (let us for the sake of notation combine all constraints into the simple $C^{l,i}(\boldsymbol{x}_k)$):

$$\begin{aligned}
&\underset{\boldsymbol{u}_k, \boldsymbol{x}_k}{\text{Minimize}}\ J(\boldsymbol{x}_k, \boldsymbol{u}_k; u_{k-1|k}) && \text{(5a)}\\
&\text{s.t.: } x_{k+j+1|k} = \zeta(x_{k+j|k}, u_{k+j|k}), \\
&\qquad j = 0, \dots, N-1, && \text{(5b)}\\
&\qquad u_{\min} \le u^{(i)}_{k+j|k} \le u_{\max}, \quad j = 0, \dots, N, && \text{(5c)}\\
&\qquad C^{l,i}(\boldsymbol{x}_k) = 0, \quad j = 0, \dots, N, \\
&\qquad i, l = 1, \dots, N_a, && \text{(5d)}\\
&\qquad x^{(i)}_{k|k} = x^{(i)}_k, \quad i = 1, \dots, N_a, && \text{(5e)}
\end{aligned}$$

This problem fits into the framework of the open-source solver `OpEn` [57], that solves parametric optimization problems of the general form (see the preliminary works for details [48], [57]:

$$\begin{aligned}
&\text{Minimize}_{z\in Z}\ f(z) && \text{(6a)}\\
&\text{subject to: } F(z) = 0, && \text{(6b)}
\end{aligned}$$

For the consideration of equality constraints, a quadratic Penalty Method [58] is applied. The penalty method is based on solving gauge problems (referred to as *inner* problems), which have the form: $\text{Minimize}_{z\in Z} f(z) + c\|F(z)\|^2$, where c is a positive penalty parameter. The inner problems are solved using PANOC and the penalty parameter is increased in an *outer* iteration loop until $\|F(z)\|_\infty$ drops below a specified *infeasibility tolerance*. This provides the perfect use-case for the edge offloading system as this method of solving the CNMPC problem in many ways benefits greatly from increased computational effort. Optimally, we want to impose a higher number of penalty method iterations and a lower (more strict) tolerance to solve for collision-free trajectories more perfectly. Additionally, increasing the prediction horizon N, especially in the case of the swarm concept, will result in more optimal orchestration of trajectories at the trade-off of computational effort. Also, as we increase the number of agents, the constraints in (3) and (4) become considerably more complex. The result is an optimization problem whose performance scales with the available computational resources.

III. EXPERIMENTAL EVALUATION

In this Section we present the experimental setup, we describe the different study cases, and we evaluate the results. The proposed framework consists of many different components; thus, the experimental setup has many different components as well. The controllable ground robots for the following experiments are the TurtleBots (TBs) [59], and the odometry of these robots is generated by Vicon Motion Capture (MoCap) System [60]. This MoCap system provides high accuracy for the position, velocity, and orientation of each robot, and is considered as the ground truth of the system. Moreover, the edge kubernetes cluster is hosted in a local Linux-based unit. The specifications of the unit are shown in Table 1. For the kubernetes cluster, microk8s [50], which is a lightweight kubernetes environment with one master node and one worker node, was utilized.

The accessibility of the system is justified by accessing the edge kubernetes cluster through a Secure Socket Shell (SSH). The user can use any device as operating working station to access the edge and the TBs when connected to the same network. In our case, we used a laptop connected to the edge. For the transmission of the data from the MoCap system to the edge, two different connection topologies were carried out as depicted in Fig. 4. In the first case, both the laptop and the edge unit are connected to the same network through Router 1 via Wi-Fi. This is the TBs' network to which the TBs are connected and through which they are receiving the control commands. Additionally, a second network is accessed through Router 2 for the MoCap system. The working station receives the odometry of the robots from the MoCap system through Router 2 and sends it to the edge unit through Router 1. Finally, edge can transmit the control

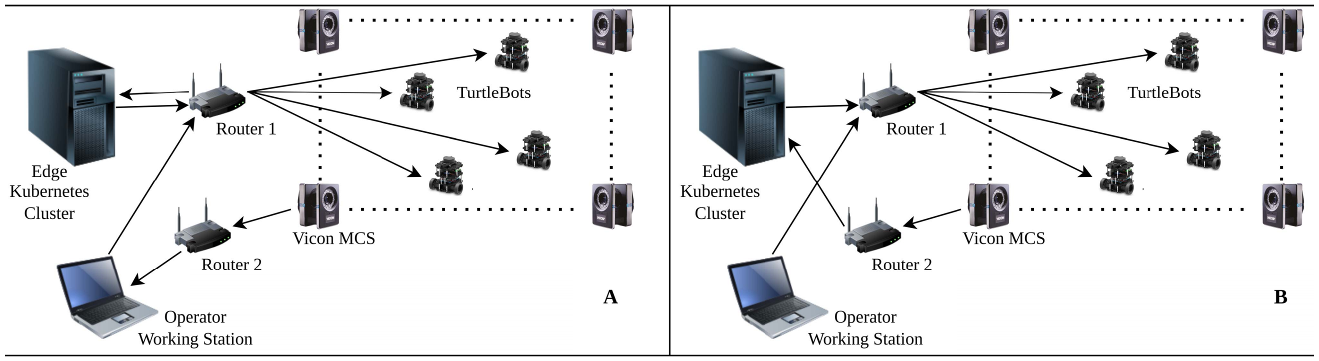

FIGURE 4. Experimental setup of the proposed framework. In setup A, the edge unit is connected in the TurtleBots' network only while in setup B, the edge unit is connected in both the MoCap network and the TurtleBots' network.

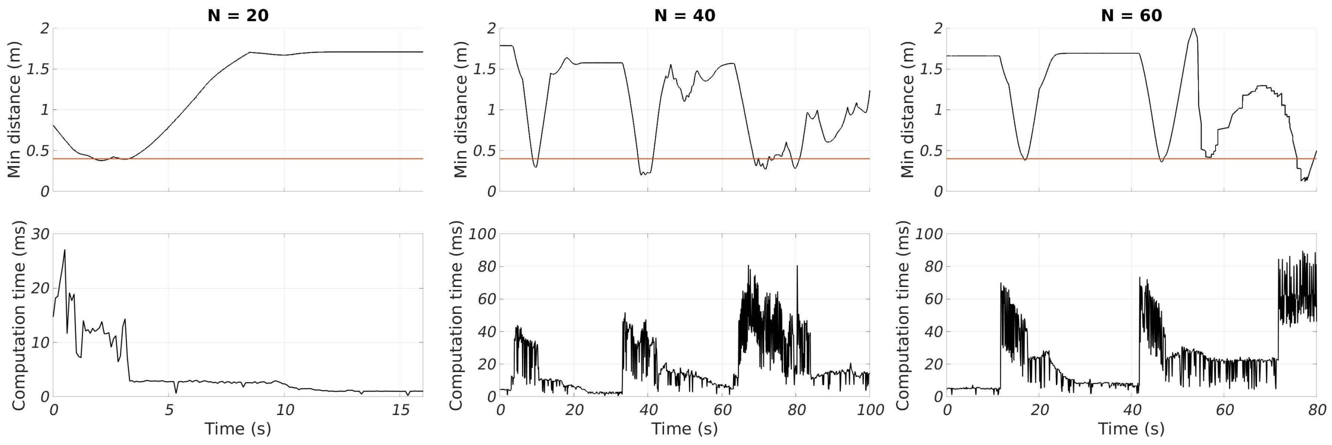

FIGURE 5. Minimum distance between the TBs (top figures) and computation time (bottom figures) of the solver for prediction horizon of 20 steps (left figures), 40 steps (middle figures) and 60 steps (right figures).

commands to the TBs through Router 1. In the second case, the edge unit is connected to both the TBs' network via Wi-Fi and the MoCap's network via ethernet cable. Since the edge unit and the MoCap system are communicating directly with each other via ethernet cable, the travel time from the agents to the edge in significantly smaller, in comparison to the first case where the data had to be forwarded from the MoCap to the operating working station, to the edge, wireless. In both cases, TBs do not communicate directly with each other but only through the edge machine via Wi-Fi.

TABLE 1. Edge machine specifications.

Processor	**Intel Core i5-8400 CPU@2.80GHz×6**
CPU Cores	6
Memory (RAM)	64GB
Operating System	Ubuntu 20.04 LTS
Disk Capacity	2.5TB

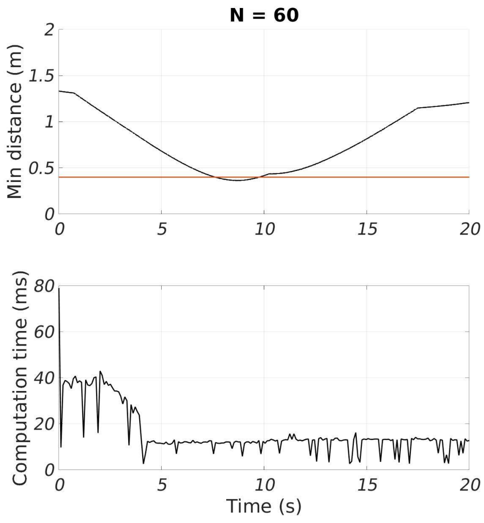

FIGURE 6. Minimum distance between the TBs (top figures), and computation time (bottom figures) of the solver when the initial tolerance is set at 0.001, and the prediction horizon is set at 60 steps.

For the first set of experiments, we utilized four TBs that should avoid potential collision while navigating into space.

In Fig. 5 the minimum distance between the TBs (top figures) and the computation time (bottom figures), during the duration of the mission, for three different prediction horizons is depicted. When the prediction horizon was set to 20 steps, the TBs could not follow the desired trajectory and they would even crash with each other. With 40 steps the behavior of the system was better, but the best performance was when the horizon was set at 60 steps. The utilization of the edge was beneficial because it gave us the chance to assign high values for the horizon, without having computational problems. The black line shows the minimum distance between the agents, and the red line shows the allowed minimum distance between the agents (constrain for collision avoidance).

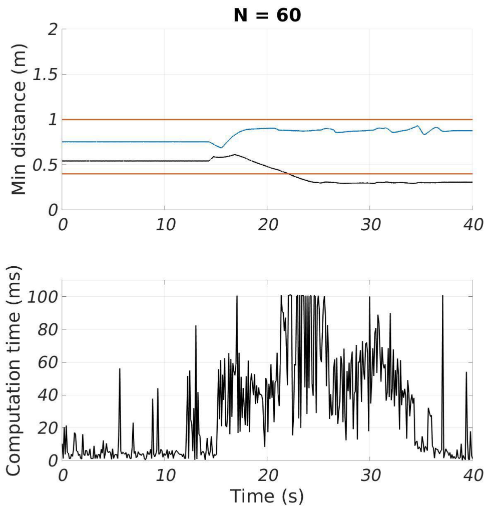

FIGURE 7. Distances between the TBs (top figure) and computation time (bottom figure) of the solver for prediction horizon of 60 while utilizing swarm behavior.

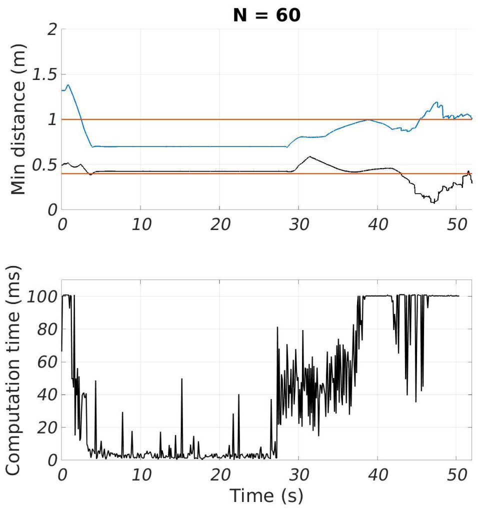

FIGURE 8. Distances between the TBs (top figure) and computation time (bottom figure) of the solver for prediction horizon of 60 while utilizing swarm behavior and avoiding obstacle.

Besides experimenting with different horizons, we also conducted experiments with different initial tolerance. The initial tolerance increases the complexity of the optimization problem, when reduced, since the solution of the optimization problem should be accurate enough to be within the certain initial tolerance value. Even though in the previous runs the initial tolerance was set at 0.00001, which is a relatively small value, the solver could find solutions in bounded time and do not break the execution of the application, as depicted in the previous graphs, thanks to the edge resources. Because in some light onboard computers, we cannot solve the problem when we use this small value, we increased the value of the initial tolerance up to 0.001. The results are depicted in Fig. 6.

With reduced initial tolerance, the solver can find solutions faster as depicted in Fig. 6 but it can fail to local minima. The

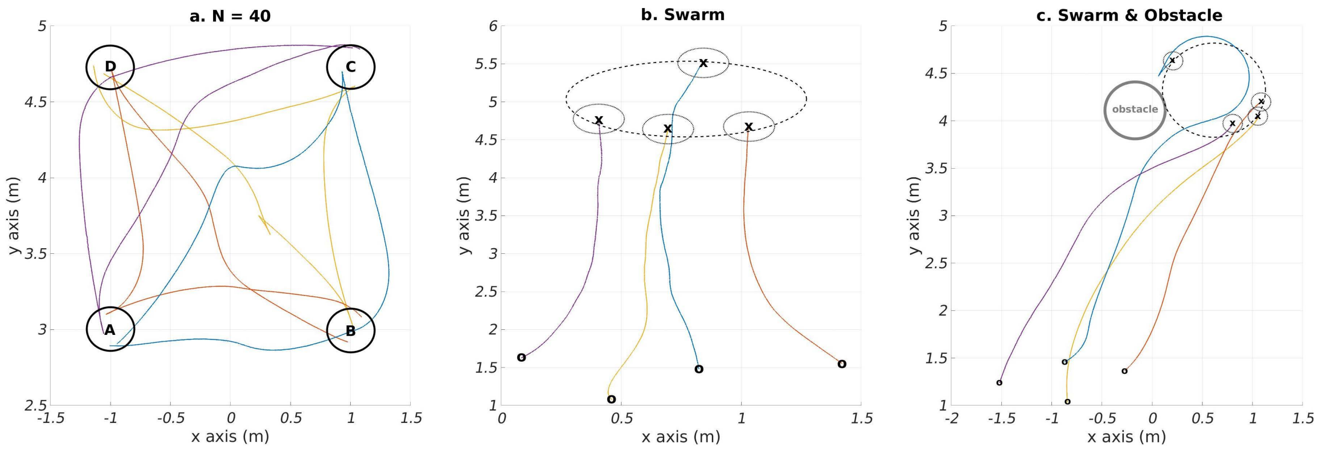

FIGURE 9. Trajectories of the TBs for a. exchanging positions and ending up in their initial position (TB1 (blue trajectory) goes from A to B then C and back to A, TB2 (red trajectory) goes from B to A then D and back to B, TB3 (yellow trajectory) goes from D to C then B and back to D, TB4 (purple trajectory) goes from C to D then A and back to C) while the prediction horizon is set at 40 steps, b. following the swarm behavior, c. following the swarm behavior while trying to avoid a static obstacle. In figures b. and c. the *o* symbol represent the starting position of the TBs and the *x* symbol their ending position. The big circle dashed line represents the maximum allowed distance between the TBs in order to keep the swarm behavior while the small circle dashed lines represent the minimum distance between two TBs in order to avoid collisions.

behavior of the TBs is not the desired one, since they do not follow the reference points but instead, they might avoid the collision but then they go to random positions.

Up next, we performed the same experiments with 4 TBs for the swarm behavior. The results with and without obstacles are depicted in Fig 7 and Fig 8, respectively. The blue line shows the maximum distance between the leader and any other agent, and the red lines show the allowed maximum distance between the leader and any other agent and the allowed minimum distance between the agents, respectively. The prediction horizon for these experiments was set at 60 steps and the initial tolerance 0.00001.

In Fig. 9, the trajectories of the TBs are depicted for the previous experiments. The figure a. shows the exchanging positions and ending up in their initial position while the prediction horizon is set at 40 steps. The TB1 (blue trajectory) starts from point A goes to point B then point C and then back to point A, TB2 (red trajectory) goes from B to A then D and back to B, TB3 (yellow trajectory) goes from D to C then B and back to D, and finally TB4 (purple trajectory) goes from C to D then A and back to C. The figure b. demonstrates the swarm behavior when we indicate the end point for the TB following the blue trajectory. The blue TB have to reach the end point while the E-CNMPC has to figure out the trajectories of all the TBs in order to respect the constrains regarding the maximum distance between the TBs (the big circle dashed line to keep the swarm behavior), and the minimum distance between the TBs (the small circle dashed lines to avoid collisions). The maximum distance was set at 1 m and the minimum 0.4 m. Finally, in figure c. the TBs should again keep the swarm behavior, but at the same time they should be trying to avoid a static obstacle and respect the previous constrains.

In addition, the Round-Trip Time (RTT) was calculated for the presented system, and every time delay that the RTT

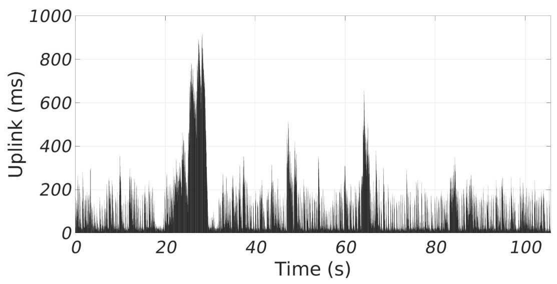

FIGURE 10. Uplink: the travel time for the data to travel from the ground robots to the edge kubernetes cluster. The A setup is used for these measurements.

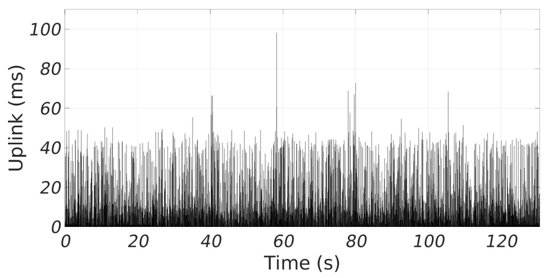

FIGURE 11. Uplink: the travel time for the data to travel from the ground robots to the edge kubernetes cluster. The B setup is used for these measurements.

is depended on is depicted in Fig. 10, 11, 12 and 13. The uplink time (t_{up}) and downlink time (t_{down}) represent the travel time for the data to travel from the agents to the edge, and vice versa respectively. These time delays were described in Section II-A through the parameters d_1 and d_3, are shown in Fig. 10, 11 and in Fig. 13. The mean uplink and downlink were measured 124.7 ms and 214.3 ms while the maximum measurements were 917.9 ms and 793 ms respectively when

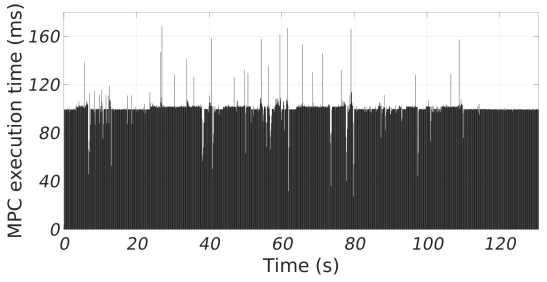

FIGURE 12. The computational time for the execution of the E-CNMPC.

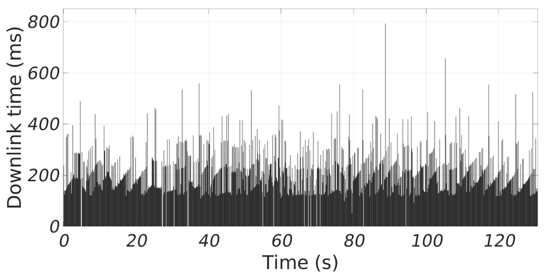

FIGURE 13. Downlink: the travel time for the data to travel from the edge kubernetes cluster to the ground robots.

setup A was used. For the uplink, when setup B was used, the mean and maximum measurements were 10 ms and 98.2 ms, while the downlink time was the same since the transmission of the data for the downlink trip was the same in both setups. Moreover, the execution time (t_{exec}) of the controller which was described through d_2 in Section II-A as well, is depicted in Fig. 12. The measured mean execution time was 100.6 ms and the maximum 168.2 ms. Since we measured the time dependencies of RTT, we can calculate the RTT from the following expression 7. The deviation of RTT is depicted in Fig. 14.

$$RTT = t_{up} + t_{exec} + t_{down} \tag{7}$$

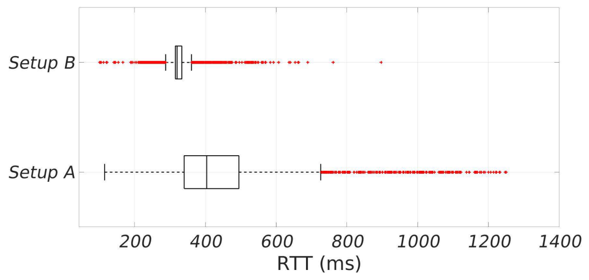

FIGURE 14. The deviation of the round trip time for both setup A and setup B.

In Fig. 14 the RTT is calculated for both setups. For setup A, the mean RTT is 448.6 ms and the maximum is 1250.2 ms while for setup B the mean and maximum measurements were 325.3 ms and 897.4 ms respectively. The difference between the two RTTs is expected since for the first setup the routing for uplink the data is longer in comparison to the second setup.

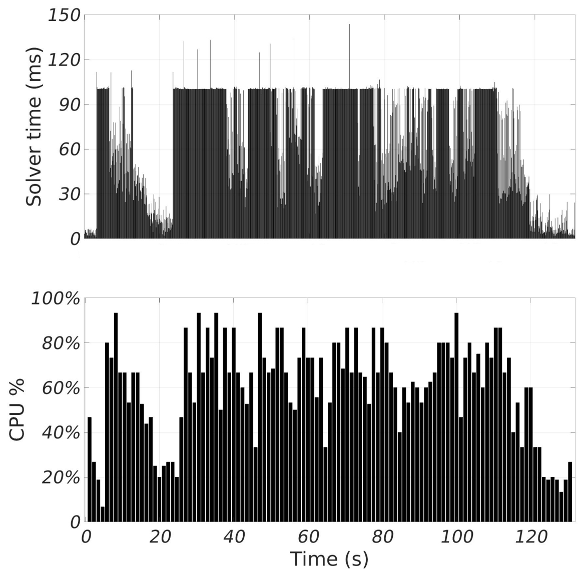

FIGURE 15. Computation time (top figure) of the solver and CPU usage for the execution of the E-CNMPC (bottom figure).

In Fig. 15, we evaluate the usage of CPU for the execution of the E-CNMPC. From the figure, it is obvious that the usage of the CPU depends on the computation time of the solver. The solver requires more time when we increase the complexity of the optimization problem (more agents, higher prediction horizon, reduced initial tolerance) and when the constrains are about to or are even violated, and they are trying to pull the system to the desired behavior. In these cases, the kubernetes pod is using near maximum resources and the CPU is even over 90%.

IV. CONCLUSION AND FUTURE WORK

In this work we presented an edge-based centralized nonlinear model predictive control framework to control multiple agents through an edge unit. The utilized technologies provide several advantages, and the centralized mechanism enables the swarm behavior of the system, while optimizing the performance of the entire system. To evaluate the proposed framework, a sequence of experiments was conducted and tested. Even though the system suffered from latency, when we selected high values for the prediction horizon and the initial tolerance, the ground robots followed the desired behavior. We were able to choose these values and control 4 ground robots simultaneously thanks to the edge resources.

Even though we were able to control multiple agents in a centralized manner thanks to the edge resources, there are still some limitations that should be taken into consideration. Our framework is characterized by latency. In some cases, the introduced latency might be higher and not bounded in the desired values, thus, the E-CNMPC will not be the optimal solution for controlling the multi-agent system. The maximum latency the robots can tolerate to guarantee safety is depended on several parameters such as the speed and the mission of the robots. In addition, the CNMPC has some computing limitation in terms of controllable agents and prediction horizon, regardless of the available resources.

An interesting future direction would be to make the application even more scalable and control more agents. This could be by utilizing bigger edge units or by dividing the E-CNMPC into more sub-controllers that would be responsible for controlling up to a specific number of agents. Additionally, safety actions can be implemented to secure the smooth transition between different edge controllers or local ones in case of network failure, high latency or when a controller or application is crashing. Thanks to the kubernetes managing properties and features, these additions can be implemented and can provide a secure environment for offloading time sensitive robotic applications for scalable systems.

REFERENCES

[1] N. A. Sulieman, L. Ricciardi Celsi, W. Li, A. Zomaya, and M. Villari, ''Edge-oriented computing: A survey on research and use cases,'' *Energies*, vol. 15, no. 2, p. 452, Jan. 2022.

[2] G. Barb and M. Otesteanu, ''4G/5G: A comparative study and overview on what to expect from 5G,'' in *Proc. 43rd Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2020, pp. 37–40.

[3] H. Zhu, M. Sharma, K. Pfeiffer, M. Mezzavilla, J. Shen, S. Rangan, and L. Righetti, ''Enabling remote whole-body control with 5G edge computing,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Jan. 2021, pp. 3553–3560.

[4] B. Dhiyanesh, ''Dynamic resource allocation for machine to cloud communications robotics cloud,'' in *Proc. Int. Conf. Emerg. Trends Electr. Eng. Energy Manage. (ICETEEEM)*, Dec. 2012, pp. 451–454.

[5] M. Groshev, G. Baldoni, L. Cominardi, A. De La Oliva, and R. Gazda, ''Edge robotics: Are we ready? An experimental evaluation of current vision and future directions,'' *Digit. Commun. Netw.*, May 2022.

[6] T. Haidegger, P. Galambos, and I. J. Rudas, ''Robotics 4.0—Are we there yet?'' in *Proc. IEEE 23rd Int. Conf. Intell. Eng. Syst. (INES)*, Apr. 2019, pp. 117–124.

[7] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. Vasilakos, ''Cloud robotics: Current status and open issues,'' *IEEE Access*, vol. 4, pp. 2797–2807, 2016.

[8] G. Hu, W. P. Tay, and Y. Wen, ''Cloud robotics: Architecture, challenges and applications,'' *IEEE Netw.*, vol. 26, no. 3, pp. 21–28, May 2012.

[9] O. Saha and P. Dasgupta, ''A comprehensive survey of recent trends in cloud robotics architectures and applications,'' *Robotics*, vol. 7, no. 3, p. 47, 2018.

[10] J. P. Queralta, L. Qingqing, Z. Zou, and T. Westerlund, ''Enhancing autonomy with blockchain and multi-access edge computing in distributed robotic systems,'' in *Proc. 5th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Apr. 2020, pp. 180–187.

[11] G. Toffetti and T. M. Bohnert, ''Cloud robotics with ROS,'' in *Robot Operating System (ROS)* (Studies in Computational Intelligence), vol. 831, A. Koubaa, Ed. Cham, Switzerland: Springer, 2020. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-20190-6_5, doi: 10.1007/978-3-030-20190-6_5.

[12] A. S. Seisa, G. Damigos, S. G. Satpute, A. Koval, and G. Nikolakopoulos, ''Edge computing architectures for enabling the realisation of the next generation robotic systems,'' in *Proc. 30th Medit. Conf. Control Autom. (MED)*, Jun. 2022, pp. 487–493.

[13] J. Luo, L. Zhang, and H. Zhang, ''Design of a cloud robotics middleware based on web service technology,'' in *Proc. 18th Int. Conf. Adv. Robot. (ICAR)*, Jul. 2017, pp. 487–492.

[14] S. A. Miratabzadeh, N. Gallardo, N. Gamez, K. Haradi, A. R. Puthussery, P. Rad, and M. Jamshidi, ''Cloud robotics: A software architecture: For heterogeneous large-scale autonomous robots,'' in *Proc. World Autom. Congr. (WAC)*, Jul. 2016, pp. 1–6.

[15] R. Rahimi, C. Shao, M. Veeraraghavan, A. Fumagalli, J. Nicho, J. Meyer, S. Edwards, C. Flannigan, and P. Evans, ''An industrial robotics application with cloud computing and high-speed networking,'' in *Proc. 1st IEEE Int. Conf. Robotic Comput. (IRC)*, Apr. 2017, pp. 44–51.

[16] S. Rosa, L. O. Russo, and B. Bona, ''Towards a ROS-based autonomous cloud robotics platform for data center monitoring,'' in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–8.

[17] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, ''FogROS: An adaptive framework for automating fog robotics deployment,'' in *Proc. IEEE 17th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2021, pp. 2035–2042.

[18] P. Huang, L. Zeng, X. Chen, L. Huang, Z. Zhou, and S. Yu, ''Edge robotics: Edge-computing-accelerated multirobot simultaneous localization and mapping,'' *IEEE Internet Things J.*, vol. 9, no. 15, pp. 14087–14102, Aug. 2022.

[19] F. Okumuş and A. Fatih, ''Exploring the feasibility of a multifunctional software platform for cloud robotics,'' in *Proc. Int. Conf. Artif. Intell. Data Process. (IDAP)*, Sep. 2018, pp. 1–4.

[20] D. Dechouniotis, D. Spatharakis, and S. Papavassiliou, ''Edge robotics experimentation over next generation IIoT testbeds,'' in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2022, pp. 1–3.

[21] F. M. de Sousa, M. Silva, and R. Oliveira, ''Applying edge AI towards deep-learning-based monocular visual odometry model for mobile robotics,'' in *Proc. 24th Int. Conf. Enterprise Inf. Syst.*, 2022, pp. 561–568.

[22] D. Spatharakis, M. Avgeris, N. Athanasopoulos, D. Dechouniotis, and S. Papavassiliou, ''A switching offloading mechanism for path planning and localization in robotic applications,'' in *Proc. Int. Conferences Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData) IEEE Congr. Cybermatics (Cybermatics)*, Nov. 2020, pp. 77–84.

[23] G. Mehrooz, E. Ebeid, and P. Schneider-Kamp, ''System design of an open-source cloud-based framework for Internet of Drones application,'' in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2019, pp. 572–579.

[24] X. Wang and H. Guo, ''Mobility-aware computation offloading for swarm robotics using deep reinforcement learning,'' in *Proc. IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2021, pp. 1–4.

[25] H. Sun and H. Xi, ''Resource optimization technology using genetic algorithm in UAV-assisted edge computing environment,'' *J. Robot.*, vol. 2022, pp. 1–8, Apr. 2022.

[26] S. Wang and N. Kong, ''Network resource allocation strategy based on UAV cooperative edge computing,'' *J. Robot.*, vol. 2022, pp. 1–9, Mar. 2022.

[27] L. Turnbull and B. Samanta, ''Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure,'' in *Proc. IEEE Southeastcon*, Apr. 2013, pp. 1–4.

[28] K.-E. Årzén, P. Skarin, W. Tärneberg, and M. Kihl, ''Control over the edge cloud-an mpc example,'' in *Proc. 1st Int. Workshop Trustworthy Real-Time Edge Comput. Cyber-Phys. Syst.*, Nashville, TN, USA, 2018. [Online]. Available: https://portal.research.lu.se/en/publications/control-over-the-edge-cloud-an-mpc-example and https://cps-vo.org/group/TREC4CPS_conference and http://2018.rtss.org/program-for-the-1st-international-workshop-on-trustworthy-and-real-time-edge-computing-for-cyber-physical-systems-trec4cps/

[29] P. Skarin, J. Eker, M. Kihl, and K.-E. Arzen, ''Cloud-assisted model predictive control,'' in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2019, pp. 110–112.

[30] A. S. Seisa, S. G. Satpute, B. Lindqvist, and G. Nikolakopoulos, ''An edge architecture oriented model predictive control scheme for an autonomous UAV mission,'' in *Proc. IEEE 31st Int. Symp. Ind. Electron. (ISIE)*, Jun. 2022, pp. 1195–1201.

[31] A. Papadimitriou, H. Jafari, S. S. Mansouri, and G. Nikolakopoulos, ''Multi-stage NMPC for a MAV based collision free navigation under varying communication delays,'' 2022, *arXiv:2208.03692*.

[32] A. S. Seisa, S. G. Satpute, B. Lindqvist, and G. Nikolakopoulos, ''An edge-based architecture for offloading model predictive control for UAVs,'' *Robotics*, vol. 11, no. 4, p. 80, Aug. 2022.

[33] P. Skarin, J. Eker, and K.-E. Arzen, ''A cloud-enabled rate-switching MPC architecture,'' in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Dec. 2020, pp. 3151–3158.

[34] P. Skarin, J. Eker, and K.-E. Årzén, ''Cloud-based model predictive control with variable horizon,'' *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6993–7000, 2020.

[35] X. Yang and J. Ni, ''A cloud-edge combined control system with MPC parameter optimization for path tracking of unmanned ground vehicle,'' *Proc. Inst. Mech. Eng., D, J. Automobile Eng.*, 2022, doi: 10.1177/09544070221080312.

[36] J. Hu, A. Bruno, D. Zagieboylo, M. Zhao, B. Ritchken, B. Jackson, J. Y. Chae, F. Mertil, M. Espinosa, and C. Delimitrou, ''To centralize or not to centralize: A tale of swarm coordination,'' 2018, *arXiv:1805.01786*.

[37] Y. Kuwata and J. P. How, ''Cooperative distributed robust trajectory optimization using receding horizon milp,'' *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 2, pp. 423–431, Mar. 2011.

[38] J. Huang, Z. Ji, S. Xiao, C. Jia, Y. Jia, and X. Wang, ''Multi-agent vehicle formation control based on MPC and particle swarm optimization algorithm,'' in *Proc. IEEE 6th Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, Mar. 2022, pp. 288–292.

[39] E. Nejabat and A. Nikoofard, ''Switched robust model predictive based controller for UAV swarm system,'' in *Proc. 29th Iranian Conf. Electr. Eng. (ICEE)*, May 2021, pp. 721–725.

[40] B. Lindqvist, P. Sopasakis, and G. Nikolakopoulos, ''A scalable distributed collision avoidance scheme for multi-agent UAV systems,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2021, pp. 9212–9218.

[41] H. Zhou and C. Liu, ''Distributed motion coordination using convex feasible set based model predictive control,'' in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 8330–8336.

[42] S. S. Mansouri, G. Nikolakopoulos, and T. Gustafsson, ''Distributed model predictive control for unmanned aerial vehicles,'' in *Proc. Workshop Res., Educ. Develop. Unmanned Aerial Syst. (RED-UAS)*, Nov. 2015, pp. 152–161.

[43] J. Wehbeh, S. Rahman, and I. Sharf, ''Distributed model predictive control for UAVs collaborative payload transport,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 11666–11672.

[44] J. Zhan, Z. Ma, and L. Zhang, ''Data-driven modeling and distributed predictive control of mixed vehicle platoons,'' *IEEE Trans. Intell. Vehicles*, early access, Apr. 19, 2022, doi: 10.1109/TIV.2022.3168591.

[45] A. Sahu, H. Kandath, and K. M. Krishna, ''Model predictive control based algorithm for multi-target tracking using a swarm of fixed wing UAVs,'' in *Proc. IEEE 17th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2021, pp. 1255–1260.

[46] S. Graffione, C. Bersani, R. Sacile, and E. Zero, ''Model predictive control of a vehicle platoon,'' in *Proc. IEEE 15th Int. Conf. Syst. Syst. Eng. (SoSE)*, Jun. 2020, pp. 513–518.

[47] S. Sabino, N. Horta, and A. Grilo, ''Centralized unmanned aerial vehicle mesh network placement scheme: A multi-objective evolutionary algorithm approach,'' *Sensors*, vol. 18, no. 12, p. 4387, Dec. 2018.

[48] B. Lindqvist, S. S. Mansouri, P. Sopasakis, and G. Nikolakopoulos, ''Collision avoidance for multiple micro aerial vehicles using fast centralized nonlinear model predictive control,'' *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9303–9309, 2020.

[49] S. Hu, W. Shi, and G. Li, ''CEC: A containerized edge computing framework for dynamic resource provisioning,'' *IEEE Trans. Mobile Comput.*, early access, Feb. 7, 2022, doi: 10.1109/TMC.2022.3147800.

[50] S. Böhm and G. Wirtz, ''Profiling lightweight container platforms: MicroK8s and K3s in comparison to Kubernetes,'' in *Proc. ZEUS*, 2021, pp. 65–73.

[51] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, ''Extend cloud to edge with KubeEdge,'' in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 373–377.

[52] I. A. Tsokalo, H. Wu, G. T. Nguyen, H. Salah, and F. H. P. Fitzek, ''Mobile edge cloud for robot control services in industry automation,'' in *Proc. 16th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2019, pp. 1–2.

[53] P. Kochovski, R. Sakellariou, M. Bajec, P. Drobintsev, and V. Stankovski, ''An architecture and stochastic method for database container placement in the edge-fog-cloud continuum,'' in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 396–405.

[54] R. P. Salas and J. Ho, ''A remote/virtual robotics lab,'' in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2021, pp. 1–4.

[55] S. Aldegheri, N. Bombieri, S. Germiniani, F. Moschin, and G. Pravadelli, ''A containerized ROS-compliant verification environment for robotic systems,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 222–225.

[56] F. Lumpp, M. Panato, F. Fummi, and N. Bombieri, ''A container-based design methodology for robotic applications on Kubernetes edge-cloud architectures,'' in *Proc. Forum Specification Design Lang. (FDL)*, 2021, pp. 1–8.

[57] P. Sopasakis, E. Fresk, and P. Patrinos, ''Open: Code generation for embedded nonconvex optimization,'' *Int. Fed. Autom. Control*, vol. 53, no. 2, pp. 6548–6554, 2020.

[58] B. Hermans, G. Pipeleers, and P. P. Patrinos, ''A penalty method for nonlinear programs with set exclusion constraints,'' *Automatica*, vol. 127, May 2021, Art. no. 109500.

[59] K. Li and H. Tu, ''Design and implementation of autonomous mobility algorithm for home service robot based on Turtlebot,'' in *Proc. IEEE 5th Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Oct. 2021, pp. 1095–1099.

[60] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, ''A study of Vicon system positioning performance,'' *Sensors*, vol. 17, no. 7, p. 1591, Jul. 2017.

ACHILLEAS SANTI SEISA (Student Member, IEEE) received the integrated master's degree in electrical engineering and computer science from the University of Patras, Greece, with a focus on automatic control and robotic systems. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden. He is a part of the Robotics and Artificial Intelligence Research Group, Signal and System Division, under the guidance of Prof. George Nikolakopoulos. His research interests include edge computing architectures for robotic applications, aerial manipulation autonomy components, and augmented reality for remote aerial operation.

BJÖRN LINDQVIST received the master's degree in space engineering with specialization aerospace engineering from the Luleå University of Technology, Sweden, in 2019, where he is currently pursuing the Ph.D. degree in aerial robotics with the Robotics and AI Team, Department of Computer Science, Electrical and Space Engineering. He worked as a part of JPL-NASA led Team CoSTAR in DARPA Sub-T Challenge on subterranean UAV exploration applications and specifically in the search-and-rescue context. His research interests include collision avoidance and path planning for single and multiagent unmanned aerial vehicle systems, and field applications of such technologies.

SUMEET GAJANAN SATPUTE (Member, IEEE) received the Ph.D. degree from the Onboard Space Systems Group, Kiruna, and the master's degree in electrical engineering with specialization in control systems from the Veermata Jijabai Technological Institute (VJTI), India. He is currently a Postdoctoral Researcher with the Robotics and Artificial Intelligence Group, Luleå University of Technology, Luleå, Sweden. His current research interests include multiple spacecraft formation and autonomous planetary explorations with multiple agents.

GEORGE NIKOLAKOPOULOS (Member, IEEE) is currently the Chair of robotics and artificial intelligence and a Professor of robotics and automation with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden. His main research interests include robotics and artificial intelligence, field robotics, UAVs, automatic control applications, learning, reasoning, networked embedded controlled systems, wireless sensor and actuator networks, mechatronics, adaptive control, system identification and multimedia wireless sensor networks, and cyber-physical systems.