Edge Computing in 5G for Drone Navigation: What to Offload?

Samira Hayat, Roland Jung, Hermann Hellwagner, Christian Bettstetter, Driton Emini, and Dominik Schnieders

***Abstract*—Small drones that navigate using cameras may be limited in their speed and agility by low onboard computing power. We evaluate the role of edge computing in 5G for such autonomous navigation. The offloading of image processing tasks to an edge server is studied with a vision-based navigation algorithm. Three computation modes are compared: onboard, fully offloaded to the edge, and partially offloaded. Partial offloading is expected to pose lower demands on the communication network in terms of transfer rate than full offloading but requires some onboard processing. Our results on the computation time help select the most suitable mode for image processing, i.e., whether and what to offload, based on the network conditions.**

***Index Terms*—Aerial systems, autonomous vehicle navigation, perception and autonomy, vision-based navigation.**

I. Introduction

AUTONOMY is desirable in many robot systems. In terms of navigation, it dictates onboard perception and computations. The concept of self-driving cars has spurred research in autonomous navigation. Improvements in sensor accuracy, perception algorithms, high onboard processing power, and low-latency communication have been topics of interest. The existing solutions remain infeasible for unmanned aerial vehicles (UAVs), or drones, as explained in the following.

Navigation autonomy in drone systems is essential in applications employing multiple drones such as disaster response [1]. Small multicopter drones weighing less than a kilogram are desirable candidate platforms due to their high maneuverability; both moments of inertia and angular acceleration scale with the characteristic dimension [2]. The maneuverability introduces opportunities in exploration and mapping of obstacle-ridden environments (e.g., flight through narrow vertical gaps [3]); the small size and payload capacity limits the onboard compute power. High-speed vision-based navigation techniques targeting autonomous driving fail to meet the challenges posed by small drone systems: high acceleration and 3D mobility coupled with low capabilities in terms of payload and onboard computation [4]. The six-degree-of-freedom trajectories comprised of high linear and angular velocities for high-speed navigation, such as in drone racing, are not yet solved by the existing state estimation solutions. The state-of-the-art methods deployed in practice use simulations to learn control and perception policies for navigation [5], [6]. Such learning-based solutions require extensive simulations. In most cases, they can only tackle static scenarios. For accurate trajectory generation in a specific scene with moving obstacles, training on multiple static scenes (with different obstacle placements) is needed to tackle the dynamics. Higher speed compromises the navigation accuracy due to the limited onboard computation capabilities [6].

This letter explores the role of edge computing to facilitate high-speed vision-based autonomous navigation. We study the offloading of the computationally intensive estimation of poses based on sensor measurements to the edge server. Such offloading requires a communication link with low latency and high throughput to achieve real-time operation. New possibilities arise with standardization activities in the Third Generation Partnership Project (3GPP), tackling the integration of drones into cellular networks [7]. The promised latency and throughput figures are sufficient to counter the mentioned challenges. A drone may respond to its environment in near real-time by sending the onboard generated sensor information to an edge server and receiving corresponding state estimation and control commands.

We evaluate the benefits of using edge servers in enabling real-time vision-based autonomous navigation of drones. To this end, we use experimental results on 5G drone connectivity [8] and profiling results of a standard monocular *Visual-Inertial Odometry* (VIO) algorithm. Three modes are compared: no offloading, partial offloading, and full offloading of the image processing tasks to the edge server. With full offloading, the edge server executes the entire image processing pipeline, which

Manuscript received October 15, 2020; accepted February 8, 2021. Date of publication February 25, 2021; date of current version March 16, 2021. This letter was recommended for publication by Associate Editor J. Cacace and Editor P. Pounds upon evaluation of the reviewers' comments. This work was supported in part by Magenta Telekom (T-Mobile Austria GmbH), in part by Deutsche Telekom AG, Germany, and in part by the University of Klagenfurt (Karl Popper Kolleg NAV), Austria. *(Corresponding author: Samira Hayat.)*

Samira Hayat is with the Institute of Information Technology (ITEC), University of Klagenfurt, Klagenfurt 9020, Austria (e-mail: samira.hayat@aau.at).

Roland Jung is with the Karl-Popper-Kolleg on Networked and Aerial Vehicles (NAV), University of Klagenfurt, Klagenfurt 9020, Austria (e-mail: roland.jung@aau.at).

Hermann Hellwagner is with the Institute of Information Technology (ITEC), University of Klagenfurt, Klagenfurt 9020, Austria and also with the Karl-Popper-Kolleg on Networked and Aerial Vehicles (NAV), University of Klagenfurt, Klagenfurt 9020, Austria (e-mail: hellwagn@itec.uni-klu.ac.at).

Christian Bettstetter is with the Karl-Popper-Kolleg on Networked and Aerial Vehicles (NAV), University of Klagenfurt, Klagenfurt 9020, Austria and also with the Institute of Networked and Embedded Systems (NES), University of Klagenfurt, Klagenfurt 9020, Austria (e-mail: Christian.Bettstetter@uni-klu.ac.at).

Driton Emini and Dominik Schnieders are with the Deutsche Telekom AG, Bonn, Germany (e-mail: Driton.Emini@telekom.de; Dominik.Schnieders @telekom.de).

Digital Object Identifier 10.1109/LRA.2021.3062319

requires the transfer of the full images to the server, resulting in low computation burden onboard and high communication demands. With partial offloading, image features are detected and tracked onboard; the drone then transfers the features to the edge server, thus offloading the remaining image processing. This yields more onboard computations than full offloading and lower communication demands. All computations are performed onboard the drone in the no offloading mode. The three modes are compared with respect to their image processing times (including computation and communication time). Considering an edge server that is computationally more powerful than the drones, the results indicate the following: If uplink rates are low (up to 100 Mb/s in our setup), partial offloading offers the best performance, whereas full offloading is beneficial for better uplink rates.

This work offers three contributions to autonomous drone navigation:

- Description of use cases motivating the use of edge computing to support drones, emphasizing the real challenges in autonomous navigation of small drones
- Extension of an open-source VIO algorithm to extract computation times on the server as compared to onboard processing
- Description of a logical partitioning of image processing into tasks for partial offloading to the edge without compromising the navigation decisions' integrity

We focus on the impact of 5G data rates on autonomous drone navigation but do not tackle the challenges of connecting drones to the network, such as handovers and antenna tilt.

In the following, Section II discusses the related work. Section III highlights the role of edge computing from the perspectives of drone applications and network providers. Section IV describes the system model. Section V presents and discusses the results. Section VI concludes.

II. Related Work

Computation offloading from drones to an edge server is gaining increasing popularity [9]–[15]. Many articles focus on using drones to enable offloading data from mobile ground users to an edge server (refer to [16] for a detailed survey on drone-enabled/drone-assisted offloading). In the following, we review articles that study the advantages and challenges of offloading tasks to an edge server.

In [9], the authors seek to minimize the computation burden by offloading tasks to the edge server while considering the delay requirements. The delay comes from task queuing at the server in order to use the server cores. Communication delays are not considered. The authors in [10] provide a survey regarding challenges in task partitioning, allocation and execution when offloading to an edge. The role of communication to enable edge computing is not explored. In [11], the focus is on collaboration among edge computing entities for data caching and executing computational tasks. A measurement study of Wi-Fi and LTE in terms of offloading is reported in [12], focusing on network interference. Image processing is offloaded to a ground station to generate a flight trajectory for a drone. LTE links are reported to be more stable than Wi-Fi for mobile drones. The authors in [13] focus on the security of the offloading process to counter eavesdropping. Perhaps the works most closely related to this letter are [14] and [15]. The authors provide a comprehensive comparison of offloaded (over cellular and Wi-Fi networks) and onboard computations in [14]. The overhead (delay, energy and communication costs) is studied. Drones decide, using a game theory approach, whether to offload or execute the tasks fully onboard. In [15], the concept of partial offloading to entities such as a base station or an edge server is introduced, and synthetic values are used to compare partial and full offloading to these entities versus onboard processing. Using synthetic values is an oversimplified assumption, as application partitioning is one of the biggest challenges identified by the research community [10]. Such generalized values do not work to model real-world applications; even if an application may be partitioned, the compute complexity of the partitioned tasks may differ. A general solution to the problem of task offloading in a network is impossible because: (i) not all applications may be partitioned, (ii) the interdependence of the tasks makes offloading a challenge, and (iii) the compute complexity of the tasks may vary widely.

One of the novelties of this letter is that we use realistic values in terms of both image processing (simulations) and communication (real-world experiments). The simulations use a standard algorithm to illustrate how image processing for visual-inertial navigation may be split into realistic tasks to facilitate partial offloading.

III. Motivating Examples

We consider drones maneuvering in a GPS-denied environment using visual-inertial navigation. One use case is forest inventorying [17]: Tree parameters, such as diameter at a certain height and position, are collected to extract the trees' volume and distribution in a forest. A drone must stay at constant height above ground to extract these values. The forest offers an obstacle-rich terrain, which necessitates high maneuverability ensured by small multicopters. They are limited in their payload but carry the appropriate visual sensors and computation boards. They must accurately and autonomously navigate through the forest. The drone speed and coverage area must be enough to justify the use of drones instead of manual methods. High flight accuracy may require processing of images with high resolution. Real-time performance requires heavier and more power-hungry processing boards for higher position estimation accuracy. High payload translates into less agility, shorter flights, and smaller coverage. The payload limitations, along with speed and coverage requirements, motivate the need for offloading.

A second use case to motivate edge computing for autonomous drones is 3D mapping. In the Karl Popper Kolleg (doctoral school) on *Networked Autonomous Aerial Vehicles*, we consider a multi-drone system that navigates through an unknown dynamic 3D environment (e.g., an area after an earthquake) and collaboratively maps it. Drones navigate using visual

and inertial information. An edge server may support the drones beyond vision-based navigation tasks in this case. It may serve as a central entity to ensure the collision-free operation and collect and stitch map fragments together to create a consistent overall map. Significant computational power of the edge server is required to justify offloading the computations; low-latency and high-rate network performance are necessary to transmit the high-volume map data to the edge.

Finally, from a network operator's perspective, relevant questions arise motivating this work. One of them is: In which applications and to what degree is edge computing beneficial or indispensable for low-latency operations and services? Investigating (partial) offloading of vision-based navigation tasks from drones to the edge will give insights into this context.

IV. System Model and Problem Formulation

We study the time to process images captured by an onboard visual sensor to estimate the drone's pose relative to a local navigation reference frame using a VIO-based approach. This estimated pose needs periodic updates for accurate navigation and trajectory planning. To obtain realistic results, we perform timing evaluation (profiling) using our publicly available *AAU synthetic ROS dataset for VIO* [18] and a modified publicly available implementation[1] of the multi-state constraint Kalman filter for visual-inertial navigation (MSCKF-VIO) [19]. The VIO dataset can be used to evaluate different camera resolutions and sensor sample rates for the same trajectory in the same synthetic environment. The dataset provides three camera resolutions, $r_1 = 320 \times 240$, $r_2 = 640 \times 480$, and $r_3 = 1280 \times 960$ pixels, at a rate of 100 Hz, and noisy and unbiased Inertial Measurement Unit (IMU) measurements at a rate of 500 Hz.

The MSCKF-VIO algorithm uses a FAST corner detector [20] and tracks features using readings from the IMU and the optical flow between consecutive images. The algorithm allows relative vehicle trajectory estimation using only IMU measurements and static features tracked across images. The MSCKF-VIO approach is similar to an extended Kalman filter for simultaneous localization and mapping (EKF-SLAM) algorithm with the difference that the state vector does not include the visual features; by nature of the imposed constraints, MSCKF-VIO is computationally efficient, and more accurate [19]. It scales linearly with the number of observed visual features and cubically with the number of states in the filter's state vector, which can be parameterized [19]. The number of states is kept constant while assuming that a higher resolution image in a texture-rich environment will provide more details and thus more reliable visual features. This assumption is strongly dependent on the observed scene though.

To ensure deterministic processing of measurements sequentially and at a configurable rate, our modified VIO algorithm processes the synthetic dataset (see Fig. 1) in a synchronous and single-threaded manner. Thus, the profiling evaluation is slower than real-time and avoids message drops due to the introduced profiling overhead.

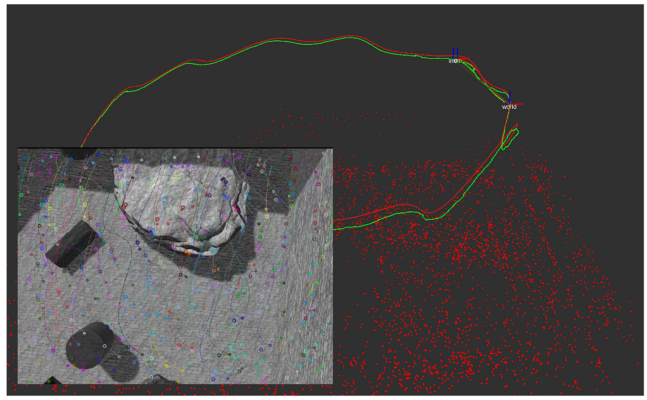


Fig. 1. Estimated (green) and true trajectory (red) of the evaluated algorithm using a synthetic dataset [18]. The red dots are estimated positions of the tracked features indicated by circles in the lower left camera image.

The code is compiled as release build with debug information, a flag to support vectorization, and executed and profiled single-threaded on an Intel i7-7820HQ CPU. The profiling results for the tested resolutions r_1, r_2, and r_3 are obtained by analyzing the *callgrind* reports in *kcachegrind*[2] that help determine the number of executed instructions and function calls for the image processing tasks described in Fig. IV-A. During each profiling run, we processed $10{,}000$ IMU samples at a rate of 100 Hz and 1000 camera images at a rate of 10 Hz (the ratio between IMU and camera rate is $i = 10$). Furthermore, we limited the number of camera poses in the filter's state vector to 20; a larger number adversely impacts the filter performance and execution times.

It is not possible to directly relate the evaluated timing results to other processing platforms with a different CPU architecture, e.g., ARM with different accelerators. This motivates the use of the clock cycle count instead of the execution time. The cycle count per function call (*cycles per call*, CPC) can then be transformed into execution time using the average CPU frequency and assuming the same processing platform: $t_{\text{exec}} = \frac{\text{CPC}}{f_{\text{CPU}}}$. For simplicity, we assume that the CPU instructions take the same number of cycles on both drone and edge server.

Our considered setup consists of a single small drone connected to a 5G base station. We consider a network configuration where the drone stays in coverage of the base station, i.e., we do not consider handovers. An edge server is located at the base station with compute power equal or greater than that onboard the drone. The infrastructure (base station and edge server) relies on the power grid, receiving regular power supply. With this setup, we study the impact of network parameters while offloading image processing tasks for autonomous drone navigation to the edge server and ignoring varying network conditions and the resulting communication challenges. Studies of network dynamics, multi-server environment and multi-drone systems will be part of future work.

In our considered use case where large-sized image data is transferred over the network using the User Datagram Protocol (UDP), the network delay responsible for initial transfer latency

[1]https://github.com/jungr-ait/msckf_mono/tree/profiling

[2]https://valgrind.org/

TABLE I
MSCKF-VIO PROFILING RESULTS FOR DIFFERENT RESOLUTIONS AND NUMBERS OF FEATURES FOR THE IMAGE PROCESSING FRONT-END AND BACK-END IN MILLION CYCLES PER CALL (CPC)

			[10^6 CPC]		
Type	Resolution	# Features	Front-end	Back-end	Total
r_1	320×240	8×6	5.64	12.98	18.63
r_2	640×480	12×9	26.09	22.67	48.76
r_3	1280×960	16×12	112.51	36.69	149.21
r_4	2560×1920	20×16	532.64	59.10	591.74

may be ignored. Such latency is of significance for small-sized packet transfers or interactive applications. We may also safely ignore the effects of packet loss in our system, as the filter in the MSCKF-VIO algorithm is robust against IMU and image packet losses to a certain degree. In our setup, the network parameter of significance is average throughput, which is extracted using real-world experiments [8]. These throughput values include any retransmission that may be necessary for error correction. The experimental campaigns report stable average network throughput in the uplink for the duration of the drone flight in the coverage of the 5G base station.

In the following, we detail the structure of the evaluated algorithm in terms of its processing tasks and average CPC count, followed by a description of the three computation modes and processing time calculation for each mode. Finally, the system parameters used in the following performance evaluation are summarized.

A. Structure of the Algorithm

The VIO algorithm is partitioned into three tasks: (i) The state propagation providing a predicted pose $\text{Pose}^{(-)}$ at the rate of the IMU, allowing aggressive maneuvers. (ii) The image processing front-end performs the visual feature extraction, estimation, and tracking. It takes the full image and the buffered IMU measurements as input and outputs the extracted features. (iii) The image processing back-end contains the feature trail and state maintenance. It takes visual features as input and results in a state and pose correction $\text{Pose}^{(+)}$.

The pose prediction task is computationally inexpensive and refers to the last corrected/updated state estimate and is neglected in the profiling. Table I lists the profiling results for the resolutions r_1, r_2, r_3 and extrapolated values for a still higher resolution r_4. The extrapolation for r_4 is justified using the compute complexity scaling of the front-end and back-end tasks. The table shows that the image front-end scales almost linearly with the pixel count. The image back-end depends on the number of tracked and newly detected visual features and the number of camera poses maintained within the filter's state vector. In general, the processing times of the front-end and back-end tasks strongly depend on the observed environment, motion, and the quality of the observed visual features. In this letter, we compute the task processing times for one specific scene. For different scenes, we expect different processing times.

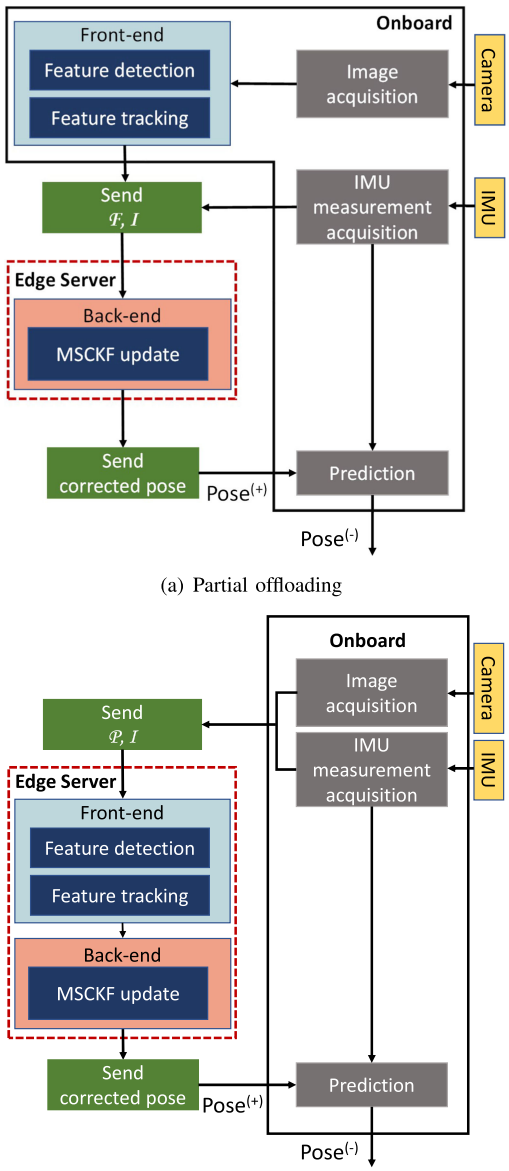


Fig. 2. Difference between partial and full offloading in terms of image processing tasks.

Fig. 2 shows the three VIO algorithm tasks and how they are shared among the drone and the edge server in the computation (offloading) modes described in the following.

B. Computation Modes

To evaluate the benefit of using edge computing, we compare three computation modes: (i) onboard computation, (ii) full offloading, and (iii) partial offloading. As the name suggests, in the first mode, all tasks including image processing are performed onboard. The difference between the second and the third mode is the set of tasks being offloaded to the edge server. Fig. 2 illustrates the differences. In the partial offloading mode, only the back-end task is offloaded, while in the full offloading mode,

both front-end and back-end tasks are offloaded to the edge server.

To extract the processing times for the different modes, we first assume that the compute power of the edge server is N times the compute power of the drone, with $N \geq 1$ (N depends on f_{CPU} and the edge server hardware architecture for optimized code execution). We use $\mathcal{P}$ ($|\mathcal{P}|$) to represent the set (number) of pixels of the acquired uncompressed image (input to the front-end) and $\mathcal{F}$ ($|\mathcal{F}|$) as the set (number) of extracted features provided as output by the front-end. We use uncompressed images to preserve the high-frequency image components (e.g., edges, corners), which constitute the most relevant information for feature extraction. The number of buffered IMU measurements depends on the capture rates of the IMU and the camera, and results in $|\mathcal{I}| = i$ IMU measurements captured per camera image.

For the onboard mode, the image processing time t_o is:

$$t_o = t_A(\mathcal{P}) + t_F(\mathcal{P}) + t_B(\mathcal{F}), \tag{1}$$

where t_A is the image acquisition time to copy the captured camera image into the local memory, t_F is the resolution dependent image front-end time, and t_B is the feature dependent image back-end processing time.

For the partial offloading mode, processing the image back-end on the edge server requires the extracted features and the buffered IMU measurements to be transmitted at the uplink rate $R_\uparrow$ to the edge server first. As the edge server has N times higher compute power, the image back-end t_B reduces proportionally. The corrected pose Pose$^{(+)}$ is then sent back to the drone at the downlink rate $R_\downarrow$. The processing time t_p for the partial offloading mode is:

$$t_p = t_A(\mathcal{P}) + t_F(\mathcal{P}) + t_T(\mathcal{F}, \mathcal{I}, R_\uparrow) + \frac{t_B(\mathcal{F})}{N} + t_R(R_\downarrow), \tag{2}$$

where t_T and t_R are the transmission and reception times to and from the edge server, respectively.

In the full offloading mode, both the image front-end and back-end are processed on the edge server, reducing the computation time of these components by N. This mode requires sending the full uncompressed image and the buffered IMU measurements to the edge server at the uplink rate $R_\uparrow$; Pose$^{(+)}$ is sent downlink to the drone similar to the partial offloading mode. The processing time for the full offloading mode t_f is:

$$t_f = t_A(\mathcal{P}) + t_T(\mathcal{P}, \mathcal{I}, R_\uparrow) + \frac{t_F(\mathcal{P})}{N} + \frac{t_B(\mathcal{F})}{N} + t_R(R_\downarrow). \tag{3}$$

Here, t_T depends on $\mathcal{P}$, $\mathcal{I}$, and $R_\uparrow$.

A single feature has 12 bytes (two single-precision floats for the subpixel position and an integer value for the identifier). An IMU measurement contains the current linear acceleration in three dimensions and the angular velocity about three axis (in total six single-precision floating point values requiring 24 bytes). Each image, IMU measurement, corrected pose, and set of features additionally includes a timestamp of size 's' of 4 bytes. We use UDP for data transmission over the Internet Protocol (IP). This results in an overhead 'o' of 28 bytes per transferred packet (taking into account both UDP and IP headers). We define n_f and n_p as the number of UDP packets needed to transfer $\mathcal{F}$ and $\mathcal{P}$ at $R_\uparrow$, respectively. Using the above information, the data transmission time in the partial offloading mode is:

$$t_T(\mathcal{F}, \mathcal{I}, R_\uparrow) = t_{\text{lat}} + \frac{(|\mathcal{F}| \times 12 + s + |\mathcal{I}| \times (24+s) + o \times n_f) \times 8}{R_\uparrow}, \tag{4}$$

where t_{lat} is the time taken by the packets to traverse the protocol stack layers plus propagation time.

For transmitting the uncompressed 8-bit grey scale images in the full offloading mode, the data transmission time is:

$$t_T(\mathcal{P}, \mathcal{I}, R_\uparrow) = t_{\text{lat}} + \frac{(|\mathcal{P}| + |\mathcal{I}| \times (24+s) + o \times n_p) \times 8}{R_\uparrow}. \tag{5}$$

The time to receive the corrected pose Pose$^{(+)}$ is assumed to be constant and very small. A pose (a unit quaternion for the orientation and a position vector) requires seven single-precision floating point values (28 bytes):

$$t_R(R_\downarrow) = t_{\text{lat}} + \frac{(28 + s + o) \times 8}{R_\downarrow}. \tag{6}$$

These equations highlight the following: In the full offloading case, transferring the uncompressed image imposes specific demands on the communication link, which increase with the image resolution. In partial offloading, where the features are transferred, a much lower data rate is required. As an example, the size of an uncompressed 8-bit grey scale r_3 image is 9.8 Mb. Using the simulated scene, the maximum number of extracted features for r_3 images is 192. The size of the corresponding transferred data reduces to 18.5 kb. In short, partial offloading poses lower communication demand than full offloading by performing part of the image processing onboard the drone.

C. System Parameters

We use results from a recently conducted 5G experimental campaign with drones [8] to extract the average uplink (drone to edge server) and downlink (edge server to drone) data rates. These values are reported to be 40 Mb/s and 320 Mb/s, respectively. We further extend our investigation using the target 5G data rates in both uplink and downlink. We use images of different resolutions to study the impact of offloading (full and partial) on image processing times over varying network rates. Table II lists the parameters and their chosen values. We ignore the latency component t_{lat} for the following results. As image acquisition is performed on the drone for all computation modes, we do not consider $t_A(\mathcal{P})$ in the following performance evaluation.

V. Performance Evaluation

We present results based on the system model and the chosen parameter values. Figures 3 and 4 show the image processing

TABLE II
SYSTEM PARAMETERS (ALL SIZES ARE IN BITS)

Parameter	Value(s)
Drone processor frequency f_{CPU}	1 GHz
Edge/drone compute power N	1, 2, 5, 10, 20
Image resolutions	$r_1, \ldots, r_4$ (see Table I)
Uncompressed image size $\lvert\mathcal{P}\rvert \times 8$	$r_1, \ldots, r_4$: 0.6, 2.5, 9.8, 39.3 Mb
Number of features $\lvert\mathcal{F}\rvert$	$r_1, \ldots, r_4$: 48, 108, 192, 320
Feature size	96 b
Corrected pose Pose$^{(+)}$ size	224 b
Timestamp size s	32 b
UDP overhead o	224 b
Uplink rate $R_\uparrow$	40, 100, 200, 400, 800 Mb/s
Downlink rate $R_\downarrow$	320, 800, 1600, 3200, 6400 Mb/s

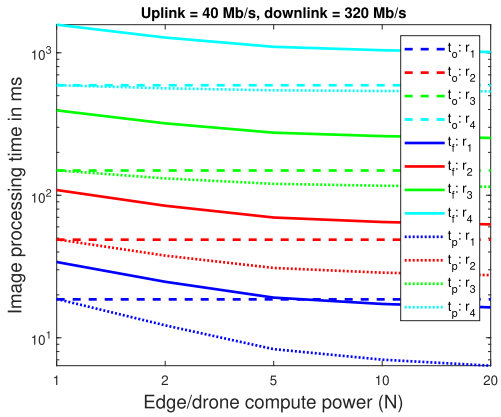


Fig. 3. Image processing time at 5G uplink rate of 40 Mb/s. The dashed lines show the image processing time in the onboard computation mode, the solid lines depict the full offloading mode while the dotted lines represent the partial offloading mode. The plots in blue, red, green and cyan represent results for r_1, r_2, r_3 and r_4, respectively.

times for different uplink and downlink rates for different resolution images with varying N. In the following, we focus on uplink rates, as t_R (corrected pose communication time from the edge to the drone) is very small.

1) Impact of edge/drone Compute Power N: We first study the impact of N (the edge's compute power as a multiple of the drone's compute power) on the image processing time needed to extract the current drone position. For a 40 Mb/s uplink, Fig. 3 shows that if drone and edge server have the same compute power ($N=1$), onboard computation outperforms the other two modes, especially for high resolutions; transferring the images/features from the drone to the edge requires transmission time t_T. Note that the difference between the image processing times of the full/partial offloading modes and the onboard mode at $N = 1$ corresponds to the image/feature transmission time. The image processing time for the full offloading mode is higher than with partial offloading for all image resolutions, and this difference becomes more prominent with increasing resolution. This result is expected, as the transmission time for images (full offloading) is much higher than for feature transfers (partial offloading), as discussed.

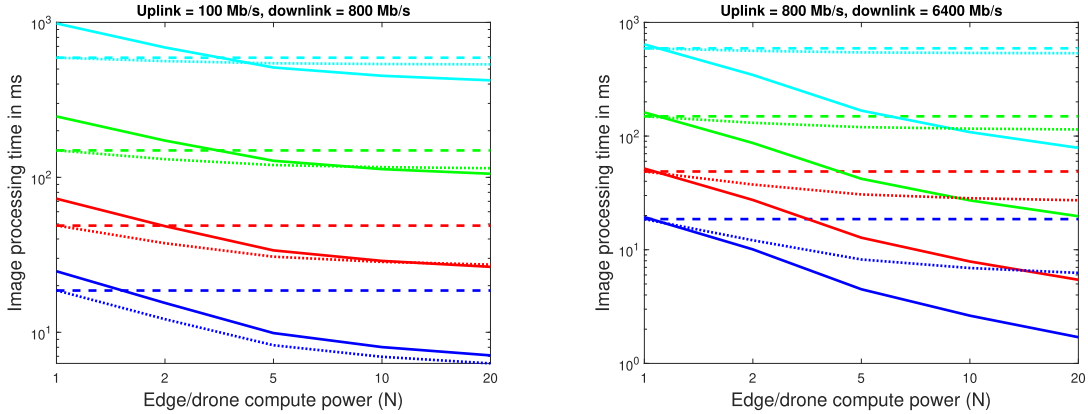
With increasing compute power of the edge server, we witness an improvement in image processing time for both partial and full offloading. If the server's compute power is twice that of the drone's, partial offloading yields lower image processing times compared to onboard processing. This is not the case for full offloading. For r_2, r_3, and r_4, full offloading never outperforms the onboard mode for the chosen N values. We may deduce that with an uplink rate of 40 Mb/s and $N > 1$, transferring features to the edge is beneficial; the image processing time reduces due to the higher compute power of the server for all image resolutions. On the contrary, full offloading requires long image transmission time; the higher edge compute power does not improve the overall image processing time. Fig. 3 shows that it takes more than one second to transfer r_4 uncompressed images to an edge server (comparing the curves for $t_o : r_4$ and $t_f : r_4$ at $N = 1$). This causes the overall image processing time with full offloading to be 400 ms higher than the onboard processing time even for $N = 20$.

2) Impact of Uplink Data Rate: Next, we study the impact of the 5G data rates on the image processing time in each mode. We observe in Figure 4 that the image processing time with full offloading improves in comparison to Fig. 3. For example, the transmission time for r_3 reduces from 240 ms for 40 Mb/s to 12 ms for 800 Mb/s uplink rate. We may deduce that the higher uplink rates can satisfy the image transfer demands of the full offloading mode.

Comparing Figs. 3 and 4 highlights another important point. For the uplink rate of 40 Mb/s, partial offloading outperforms full offloading for the considered resolutions. The full image transmission time outweighs the computational time improvement offered by the edge server. Extracting features onboard and transmitting to the edge offers a higher benefit than transmitting the entire image. The situation changes for higher uplink rates. For instance, in Fig. 4(a), transferring 192 features (dotted green line) takes 95 ms less than transferring the full r_3 image (solid green line). The increasing edge compute power (increasing N) decreases the computation time. For $N \approx 8$, this improvement compensates for the overhead transmission time (t_T). Beyond this value of N, full offloading surpasses partial offloading. This crossover point beyond which full offloading outperforms partial offloading shifts further to the left for higher resolutions; for r_4, this phenomenon occurs at $N \approx 4$. For higher uplink rates, full offloading outperforms the other modes for $1 < N \leq 2$. This is an important result, illustrating that with high enough uplink rates, the transmission time is small enough so that the image transfer does not take longer than transferring features. Offloading the maximum number of tasks may be more beneficial than reducing the overhead due to communication. This result holds for all image resolutions.

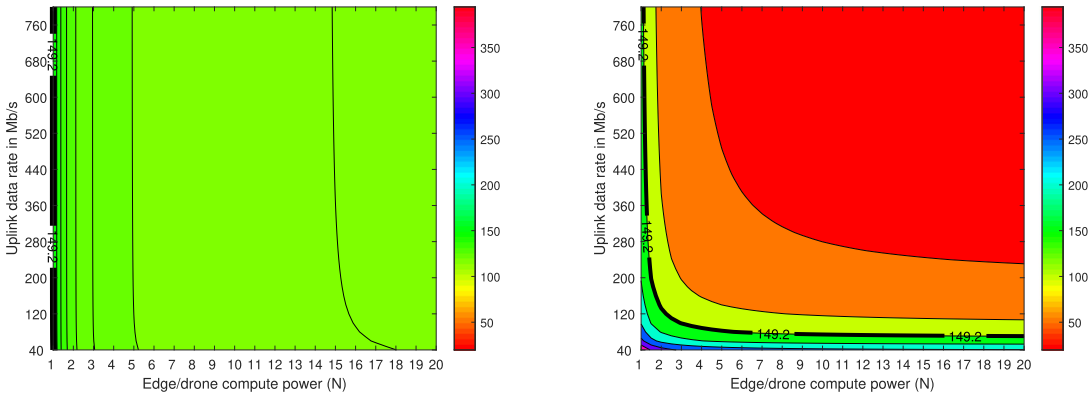
The results also illustrate the advantage of using high 5G uplink rates and high edge compute power together: It may ensure that higher resolution images are processed faster on the edge than lower resolution images onboard the drone. Fig. 4(b) illustrates this. For $N = 20$ with an uplink rate of 800 Mb/s, processing an image with r_3 fully on the edge takes the same time as processing the r_1 image onboard.

Another aspect of the results pertains to the absolute image processing times attained in the different computation modes and, thus, the achievable state estimation frequencies (or, state update rates). Usually, VIO algorithms become more accurate with increasing state update rates and higher image resolutions,



(a) 100 Mb/s (b) 800 Mb/s

Fig. 4. Image processing times at increasing 5G uplink rates. Legend as in Fig. 3.



(a) Partial offloading (b) Full offloading

Fig. 5. Image processing times for r_3 using (a) partial and (b) full offloading modes for different data rate and N values. The color bar indicates the image processing time in ms. The solid black line depicts the onboard image processing time.

leading to faster flights and/or more precise navigation. A state update frequency of 30 Hz (33 ms image processing time) is regarded as desirable.

With onboard computation, we achieve the target 30 Hz state update rate only for r_1 images. Processing images of resolution r_2 at this rate requires a minimum 100 Mb/s uplink rate and an edge compute power of $N > 5$, for both offloading modes. The target rate is achieved for r_3 images only with high edge compute power, uplink rates, and full offloading. The highest image resolution r_4 is out of reach at the target rate: there is simply too much data to compute for the drone's onboard computer (feature extraction for partial offloading) or to communicate to the edge (full offloading). More compute power aboard the drone would ameliorate the situation.

3) What to Offload?: Finally, we use the image processing times for r_3 to illustrate how the results in this letter can be used to decide what to offload for a given N and uplink data rate. Fig. 5 illustrates this. The color bar represents the image processing time in ms. The contour levels are 5 ms apart in Fig. 5(a) and 50 ms apart in Fig. 5(b). The solid black line at 149.2 ms represents the onboard image processing time. Beyond $N = 1$, the partial offloading outperforms onboard processing, offering an increasing gain in image processing time with N. This is true for all uplink data rates. The situation is different for the full offloading mode. Below 80 Mb/s, full offloading performs worse than the onboard processing for all considered N. Higher uplink rates result in performance improvement; above 140 Mb/s, full offloading offers lower image processing time with $N \geq 2$. With increasing uplink rate and N, the gain in image processing time with full offloading is much higher than that offered by partial offloading. At 800 Mb/s and $N = 20$, full offloading results in an image processing time of 20 ms, as against 114 ms achieved using partial offloading. These results illustrate that for values of uplink rates and N below the solid black curve in Fig. 5(b), partial offloading provides the best image processing time for r_3. For values above the curve, full offloading mode offers the best performance.

VI. Conclusions and Outlook

Edge computing can be beneficial for autonomous navigation of drones. We compared the image processing times in three

computation modes: onboard, partial offloading, and full offloading. We profiled a modified, publicly available MSCKF-VIO algorithm and used 5G rates from measurements to extract the communication time needed for offloading tasks to the edge server. The results also consider the envisioned 5G uplink rates to support edge computing.

This work emphasizes the importance of powerful uplinks in cellular networks to support drone applications. They may support seamless high-resolution image transfers to the edge. With adequate data rates and edge computation power, the processing of high-resolution images offloaded to the edge may be faster than the processing of low-resolution images onboard. Up to a specific uplink rate, partial offloading is more beneficial than full offloading for a given edge compute power. The analysis may help choose the optimal offloading mode based on the available uplink rate and the edge server's computation capabilities.

For the robotics community, this work encourages the analysis of complex localization algorithms with more task parallelization options in a similar offloading setting. The work motivates modular algorithm design, where tasks, such as image processing front-end and back-end, are easily separable. With a more modular algorithm design, we expect to witness a more interesting task allocation optimization problem with several offloading possibilities between the drone and the edge server.

The assumptions regarding the compute power N depend highly on the observed scene, the algorithm optimization capability, and the edge server's hardware architecture. If the algorithm is highly parallelizable and the edge server offers sufficient compute resources (e.g., CPU cores, GPU support, etc.), we may achieve much higher values for N. For this work, we chose synthetic N values weakly related to the edge computer's hardware architecture and algorithm optimization. Front-end and back-end task parallelization and edge server's hardware specifications to ensure high N values are important future research questions.

Our future work will focus on improving navigation accuracy with higher image resolutions in an indoor setup using 5G infrastructure and edge computing. We plan to extend this work by investigating the impact of packet loss and other network metrics on the navigation accuracy.

REFERENCES

[1] A. Ollero, J. Ferruz, F. Caballero, S. Hurtado, and L. Merino, "Motion compensation and object detection for autonomous helicopter visual navigation in the COMETS system," in *Proc. IEEE Int'l. Conf. Robot. Automat.*, vol. 1, 2004, pp. 19–24.

[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, pp. 460–466, 2015. [Online]. Available: https://doi.org/10.1038/nature14542

[3] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," Int'*l J. Robot. Res.*, vol. 31, no. 5, pp. 664–674, 2012.

[4] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset," in *Proc. IEEE Int'l. Conf. Robot. Automat.*, 2019, pp. 6713–6719.

[5] M. Mueller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Proc. Conf. Robot Learn.*, 2018, pp. 1–15.

[6] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 1–14, Feb. 2020.

[7] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A. tutorial on UAV communications for 5G and beyond," in *Proc. IEEE IRE*, vol. 107, no. 12, pp. 2327–2375, 2019.

[8] R. Muzaffar, C. Raffelsberger, A. Fakhreddine, J. L. Luque, D. Emini, and C. Bettstetter, "First experiments with a 5G-Connected drone," in *ACM MobiSys DroNet Workshop*, 2020, pp. 1–5.

[9] C. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing," in *IEEE Globecom Workshops*, 2017, pp. 1–7.

[10] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," in *Proc. IEEE Proc. IRE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[11] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *Proc. Asia-Pacific Netw. Operations Manage. Symp.*, 2017, pp. 366–369.

[12] D. Callegaro, S. Baidya, and M. Levorato, "A measurement study on edge computing for autonomous UAVs," in *Proc. ACM SIGCOMM Workshop Mobile AirGround Edge Comput., Syst., Netw., Appl. (MAGESys)*, 2019, pp. 29–35.

[13] T. Bai, J. Wang, Y. Ren, and L. Hanzo, "Energy-efficient computation offloading for secure UAV-Edge-Computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 6074–6087, Jun. 2019.

[14] M. Messous, S. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an UAV network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May 2019.

[15] M. Messous, H. Hellwagner, S.-M. Senouci, D. Emini, and D. Schnieders, "Edge computing for visual navigation and mapping in a UAV network," in *Proc. IEEE Int'l. Conf. Commun.*, 2020, pp. 1–6.

[16] F. Zhou, R. Q. Hu, Z. Li, and Y. Wang, "Mobile edge computing in unmanned aerial vehicle networks," *IEEE Wirel. Commun.*, vol. 27, no. 1, pp. 140–146, Feb. 2020.

[17] R. Muzaffar, A. Hardt-Stremayr, S. Hayat, A. Vogell, and E. Yanmaz, "Autonomous drone navigator for 3D reconstruction of forests," in *Proc. IEEE/RSJ Int'l. Conf. Intell. Robots Syst.*, 2019, pp. 1–5.

[18] R. Jung, G. Rischner, E. Allak, A. Hardt-Stremayr, and S. Weiss, "AAU Synthetic ROS Dataset for VIO," pp. 1–1, May 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3870851.

[19] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proc. IEEE Int'l. Conf. Robot. Automat.*. IEEE, 2007, pp. 3565–3572.

[20] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Proc. Eur. Conf. Comput. Vis.*, 2006, pp. 430–443.