2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on

A Switching Offloading Mechanism for Path Planning and Localization in Robotic Applications

Dimitrios Spatharakis*, Marios Avgeris*, Nikolaos Athanasopoulos†, Dimitrios Dechouniotis*
Symeon Papavassiliou*
**National Technical University of Athens, Greece*
†*School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Northern Ireland, UK*
{dspatharakis, mavgeris}@netmode.ntua.gr, n.athanasopoulos@qub.ac.uk, ddechou@netmode.ntua.gr, papavass@mail.ntua.gr

***Abstract*—Industry 4.0 applications rely on mobile robotic agents that execute many complex tasks that have strict safety and time requirements. Under this setting, the Edge Computing service delivery model allows the robotic agents to offload their computationally intensive tasks to powerful computing infrastructure in their vicinity. In this study, we propose a novel switching offloading mechanism for such robotic applications. In particular, we design opportunistic offloading strategies for the path planning and localization services of mobile robots. The offloading decision is based on the uncertainty of the robot's pose, the resource availability at the Edge of the network and the difficulty of the path planning. Our switching offloading framework is implemented and evaluated using a robot in a real Edge Computing testbed, where the trade-off between execution time and the successful completion of the robot trajectory is highlighted.**

***Index Terms*—IoT, Edge Computing, Computational Offloading, Robotics, Switching Control.**

I. Introduction

Computation offloading in current and next-generation networks is becoming increasingly important due to the proliferation of Internet of Things (IoT) real world applications [1]. These applications introduce a vast number of low-capability, low-energy devices to the networking ecosystem, which regularly need to perform computationally intensive and/or energy-hungry tasks. However, when latency and energy consumption minimization are required, the limited resources of the IoT devices prove inadequate [2]. For example, in Industry 4.0 and especially in collaborative robotics, where humans and robots work together in dynamic environments, computationally heavy algorithms enable IoT devices in sensing and actuating [3]. Consequently, large amount of information has to be processed and complex algorithms need to be executed in real-time.

The increasing availability of networking in the Edge and Cloud supports new approaches, where processing is performed remotely, with access to extensive computing and memory resources. In this direction, Edge Computing (EC) alongside Fog Computing (FC) [4] constitutes a particularly prominent way of dealing with the aforementioned shortcomings of IoT devices. FC offers an attractive alternative providing low-latency and high energy efficient operation, while maximizing system performance. This paradigm is currently more relevant than ever, especially in the context of the much-anticipated Industry 4.0 revolution [5] and Industrial IoT (IIoT), where Fog Robotics (FR) is introduced. FR can be defined as the architecture that distributes computing, storage and networking functions at the Edge/Cloud continuum in a federated manner [6], i.e. where robots and automation systems rely on data or code from a network to support their operation.

Suitable as it may seem, solely utilising remote computational resources is not enough; a number of unwanted phenomena potentially take place in the transmission and processing of the information, such as network latency, variable Quality of Service (QoS), and downtime. For these reasons, autonomous mobile robots often have some capacity for local processing when targeting low-latency responses, and during periods where network access is unavailable or unreliable. Consequently, a major challenge, from a control design, estimation, and network optimization point of view is to combine local and remote resources in an efficient way.

In this work, we propose a computation offloading mechanism for robotic applications. In particular, we realize an IoT-enabled localization and path planning framework and verify the expected gains of computation offloading by utilizing a real Edge Computing setting. To achieve this, we design and implement local and remote localization and path planning controllers, followed by a scheduling mechanism. The offloading mechanisms are treated as switches, leading to different dynamics of the resulting closed-loop system. Specifically, the algorithms involved in the localization process are decided to run remotely, rather than locally, when the uncertainty of the robot's pose is high and at the same time the network and computing resources status at the Edge is favorable. On the other hand, path planning is offloaded when the robot navigates in a part of a map where better planning strategies can be achieved through involved algorithms that can only be executed remotely. These switches compose a switching system that is adaptive and can operate under different scenarios and applications. This architecture perspective, which constitutes the main contribution of this work, offers our framework a degree of contextual awareness; that is the ability to sense and dynamically adapt to the robot's environment, implicitly enhancing to an extent the robustness of its operation, as well as improving the QoS of the supported applications.

Open challenges in this area throughout the literature are concerned with developing adaptive multi-robot/machine control, capturing, modelling, predicting and anticipating the agent's interactions and designing distributed control and path planning algorithms that deliver flexible and safe working environments. Approaches similar to ours include [7], where gesture-based semaphore mirroring with a humanoid robot is split to remotely and locally executed functionality; [8], in which the authors identify a three-layered environment (Robot, Edge and Cloud) to overcome the challenges of network limits in a Deep Robot Learning application and [9] where Dew Robotics is introduced; this concept posits that critical computations are executed locally so that the robot can always react properly, while less critical tasks are moved to the Fog and Cloud, so to exploit the larger availability in computing, storage, and power supply. However, none of the aforementioned offloading decision schemes addresses the dynamic nature of the robot's environment.

The rest of the paper is organized as follows. In Section II the architecture overview is presented, while in Section III the system model and local tracking controller are presented. The algorithms used in the scope of this work for localization and path planning are presented in Section IV. The switching offloading mechanism is presented in Section V. An experimental evaluation in Section VI. Finally, conclusions are drawn and future plans are set in Section VII.

II. Architecture Overview

The scenario addressed in this work involves a mobile robot equipped with sensing, computing, and wireless communication capabilities, which makes its way from a starting position to a target position in an operating ground (e.g. a factory floor), navigating through obstacles. This functionality is a key component to realizing autonomous robotic navigation in Industry 4.0 use cases, e.g. warehousing and logistic robots which automate the process of storing and moving supply chain goods. Tracking the robot location is essential for a robust and safe trajectory planning. However, a common problem in such a scenario is that the uncertainty in estimating the exact pose (i.e. position and orientation) grows over time in motion, due to inaccuracies in sensing, wheel slips, hardware failures, etc., [10]. Thus, the importance of an accurate, dynamically adjusted localization technique is evident.

In our case self-localization through landmark assisted pose estimation is implemented; the robots are equipped with a camera module, while in their proximity unique cylindrical beacons are used as landmarks to assist in the pose estimation process. In the computationally demanding involved algorithms, two offloading opportunities are revealed in, namely, pose estimation and path planning. To this purpose, a small-scale network infrastructure is set up, connecting the robot to a wireless LAN (WLAN) through an Access Point located within the robots' network range, which in turn connects via a wired connection (LAN) to a server in the robot's proximity, the Edge Server.

Locally, the intangible assets include the (i) the Tracking Controller (TC), (ii) the Local Odometry-Based Estimator

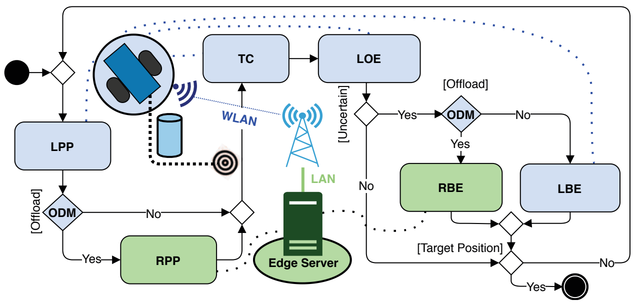

Fig. 1: Architecture Overview. The locally executed components are highlighted with blue color, while the remotely executed ones with green.

(LOE), (iii) the Local Beacon-Based Estimator (LBE), (iv) the Local Path Planner (LPP) and (v) the Offloading Decision Mechanism (ODM) components, all located within the robot; component (i) is responsible for carrying out movement-related decisions, (ii), (iii) and (iv) are the locally executed pose estimation and path planning applications respectively and (v) encompasses the intelligence of our switching system by monitoring the offloading-related metrics and realizing the offloading decisions. On the remote side, containerized counterparts of the path planning and pose estimation applications are co-hosted on the Edge Server; these are namely (vi) the Remote Beacon-Based Estimator (RBE) and (vii) the Remote Path Planner (RPP) which are able to receive offloaded requests from the robot. A more detailed discussion on these components follows in Sections III, IV and V.

In order to outline the sequence of interactions between the main components of the architecture, we showcase a representative scenario in which our solution applies successfully. Fig. 1 depicts an overview of this scenario. Without loss of generality, we assume that only one robot operates in the field. Also, its starting pose, the operating space dimensions and the obstacles' and beacons' positions and shapes are considered known a priori.

A typical activity flow of our scenario, initiates with Local Path Planner component calculating locally a trajectory from the starting position to the target position. This triggers the ODM for the first time; should a quick analysis on the projected trajectory indicate room for significant refinement of the selected path, the Remote Path Planner is invoked. This analysis is based on the trajectory curvature and the degree in which the more elegant remote component is potentially able to smooth it around obstacles; Section V-C provides more insight on this process. Eventually, the resulted trajectory dictates the *intermediate positions* the robot needs to reach. In order to sequentially perform the transition to the each of them, the Tracking Controller component is invoked.

After reaching the next position of its trajectory, an uncertainty indicator of the pose estimation is calculated; this indicator is a scalar that grows with time and actually accumulates the error between the estimated and the reference

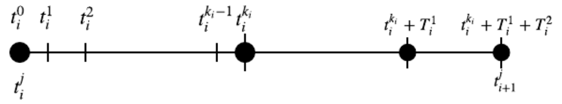

Fig. 2: The timing sequence in the proposed scenario.

pose after each move, as explained thoroughly Section V-A. Here, the second decision occurs; if this indicator measures bellow a predefined threshold, the robot continues to move based on the feedback coming from the Tracking Controller's monitoring process, i.e. the Local Odometry-Based Estimator, which leverages the robot's photoelectric sensors (encoders) attached to each wheel to measure the wheels' angular velocities during a period of time. Else, it invokes the more precise, but computationally heavy, Beacon-Based Pose Estimator, leveraging information coming from the beacons in the environment. That triggers the ODM once again; the Edge Server is queried to provide an estimation on the duration of the potentially offloaded pose estimation task. As described by the mathematical modelling in Section V-B1, this duration is proportional to the availability of the computational resources. Based on this estimated duration, a decision is made on whether to offload the pose estimation task to the Remote Beacon-Based Estimator, or execute it locally. The flow ends with the robot checking if the target position is reached. If not, it reverts to first step.

It is worth highlighting that the tracking controller, as well as the path planning and pose estimation are aperiodic. The position of the robot on the operating ground, is defined by the state vector $\boldsymbol{x^i} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top$. The robot has to move towards the next reference position $\boldsymbol{x^i_{ref}} = [x_{1,\text{ref}}(t_i) \quad x_{2,\text{ref}}(t_i)]^\top$, generated by the path planning algorithms, to approach the target position. Fig. 2 gives a brief insight on the timing sequence in which the rest of the sections will refer to. Let subscript i correspond to the step during which the robot reaches the next reference position in k_i actuation steps, while simultaneously tracking its pose. In particular, at time t_i^0 the robot is in the position x^i. When the next reference position x_{ref}^{i+1} is close, the uncertainty about the current estimation is calculated. Thus, the time duration T_i^1 corresponds to the time spent for localization. When the local odometry-based estimator is used, this time is equal to zero, while the beacon-based estimation algorithm is time consuming. The time duration T_i^2 corresponds to the path planning algorithm running time either remote or local, which generates the next reference position. Similarly, the time to execute the local path planning algorithm is equal to zero.

III. System Dynamics

A. Robot dynamics

The differential drive robot used in this study has two wheels that can turn at different rates, allowing motion by changing the orientation and the position (x_1, x_2) either separately or simultaneously. For the robot dynamics, the 2D coordinates, i.e. position, and the orientation of the robot are denoted by the state variables z_1, z_2 and z_3. Hence we consider $\boldsymbol{z} = \begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^\top = \begin{bmatrix} \boldsymbol{x}^\top & \theta \end{bmatrix}^\top$. The robot is controlled by the angular velocities w_R and w_L, accounting for the right and left wheel respectively. The robot dynamics is defined by the following continuous time system, based on the work in [11], using the aforesaid state-space representation. Specifically, we have for any $t \geq 0$,

$$\dot{z}_1(t) = \frac{r}{2}(w_L(t) + w_R(t)) \cos z_3(t), \tag{1}$$

$$\dot{z}_2(t) = \frac{r}{2}(w_L(t) + w_R(t)) \sin z_3(t), \tag{2}$$

$$\dot{z}_3(t) = \frac{r}{l}(w_L(t) - w_R(t)), \tag{3}$$

where l, r are the distance between the two wheels and the radius of each wheel respectively. The odometry measurements $\tilde{w}_L(t_i^j)$,$\tilde{w}_R(t_i^j)$ are taken at each time instant t_i^j, $i = 0, 1, ...$, $j = 0, \dots, k_i$ of the timing sequence introduced in Section II. The corresponding discretized system using Euler forward method is:

$$\tilde{z}_1(t_i^{j+1}) = \frac{r}{2}(\tilde{w}_L(t_i^j) + \tilde{w}_R(t_i^j)) \cos \tilde{z}_3(t_i^j)(t_i^{j+1} - t_i^j) + \tilde{z}_1(t_i^j), \tag{4}$$

$$\tilde{z}_2(t_i^{j+1}) = \frac{r}{2}(\tilde{w}_L(t_i^j) + \tilde{w}_R(t_i^j)) \sin \tilde{z}_3(t_i^j)(t_i^{j+1} - t_i^j) + \tilde{z}_2(t_i^j), \tag{5}$$

$$\tilde{z}_3(t_i^{j+1}) = \frac{r}{l}(\tilde{w}_L(t_i^j) - \tilde{w}_R(t_i^j))(t_i^{j+1} - t_i^j) + \tilde{z}_3(t_i^j). \tag{6}$$

B. Tracking controller

As previously mentioned, the robot moves towards the next reference position $\boldsymbol{x^i_{ref}}$ to reach the target position. For this actuation phase, given the specific robot dynamics, we propose a tracking controller executed locally on the robot, by fixing the control inputs w_L, w_R to be either equal or opposite. Therefore, the control input is w, while $|w| = |w_L| = |w_R|$. As a result, we restrict the motion of the robot to a straight line, i.e. "translational motion", or a rotation around the center of the wheel axle, i.e. "rotational motion", respectively. This control structure is chosen as it is efficient for tracking purposes, leading to a simple structure of the closed-loop system. Specifically, the closed-loop dynamics for the translational and rotational motion are

$$S_1^{Tran}: \begin{cases} \dot{z}_1(t) = \frac{r}{2}(w(t)) \cos z_3(t), \\ \dot{z}_2(t) = \frac{r}{2}(w(t)) \sin z_3(t), \\ \dot{z}_3(t) = 0 \end{cases} \tag{7}$$

$$S_2^{Rot}: \begin{cases} \dot{z}_1(t) = 0, \\ \dot{z}_2(t) = 0, \\ \dot{z}_3(t) = \frac{r}{l}(w(t)), \end{cases} \tag{8}$$

where S_1^{tran} is used for the translational motion and S_2^{rot} when the robot needs to rotate. Let $R(t_i^j) = \left\| \begin{bmatrix} \tilde{z}_1(t_i^j) \\ \tilde{z}_2(t_i^j) \end{bmatrix} - \begin{bmatrix} z_{1,\text{ref}}(t_i) \\ z_{2,\text{ref}}(t_i) \end{bmatrix} \right\|_2$ be the distance between the robot's current estimation and the reference position and let $\phi(t_i^j) = \tilde{z}_3(t_i^j) - \tan^{-1}\left(\frac{\tilde{z}_2(t_i^j) - z_{2,\text{ref}}(t_i)}{\tilde{z}_1(t_i^j) - z_{1,\text{ref}}(t_i)} \right)$ be the angle between the robot's current estimation of orientation and the line connecting the robot and the reference position. Here, $\tilde{z}$ accounts for the estimation of its current pose calculated by Equations (4) – (6) at the time period of the actuation $t = t_i^j$, $j = 0, 1, \dots, k_i$.

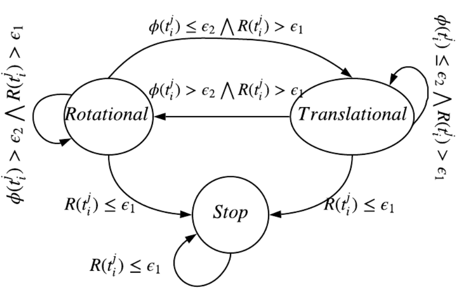
$\phi(t_i^j) \leq \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1$
$\phi(t_i^j) > \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1$
$\phi(t_i^j) > \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1$
$\phi(t_i^j) \leq \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1$
Rotational
Translational
$R(t_i^j) \leq \epsilon_1$
Stop
$R(t_i^j) \leq \epsilon_1$
$R(t_i^j) \leq \epsilon_1$

Fig. 3: The hybrid automaton of our system.

The closed-loop system with the tracking controller can be modeled by a discrete-event systems, see, e.g., [23], as shown in Fig. 3, where the control input can be calculated as follows:

$$w(t_i^j) = \begin{cases} L_1 R(t_i^j), & \phi(t_i^j) \leq \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1, \text{Translational}, \\ L_2 \phi(t_i^j), & \phi(t_i^j) > \epsilon_2 \bigwedge R(t_i^j) > \epsilon_1, \text{Rotational}, \\ 0, & R(t_i^j) \leq \epsilon_1, \qquad\qquad\quad \text{Stop}. \end{cases}$$

The quantities ϵ_1, ϵ_2 are positive constants, while the gains L_1, L_2 are constant control parameters.

The reference position is reached when the estimation of its position is close, and in particular is inside a ball of radius ϵ_1 close to the reference, i.e., centered at $\mathbf{B}(\boldsymbol{x_{ref}^i}, z(t_i^j)) = \{z \in \mathbb{R}^3 : \|z - \tilde{z}(t_i^j)\| \leq \epsilon_1\}$. The effect of the uncertainty is taken into account explicitly in the offloading decision that follows.

IV. Localization and Path Planning

In what follows, we present the algorithms chosen for localization and path planning, with a varying degree of complexity and accuracy, that are implemented locally and remotely accordingly.

A. Localization

The localization problem is equivalent to the pose estimation problem in our setting. Two algorithms of different complexity are implemented, namely, (i) an odometry-based one, and (ii) a camera-based estimation. The first estimation algorithm is light enough to run efficiently on the robotic platform. Roughly, the robot's on-board wheel encoders readings are fed to the motion model (4) – (6). While this is a lightweight and fairly accurate localization technique when it comes to short trajectories, odometry is known to be prone to accumulative errors [12].

The second localization technique is the computationally heavier beacon-based estimator. Details on the technical parts of the algorithm and its software and hardware implementation can be found in [13]. Roughly, the technique is based on a bilateration method using principles of the projective geometry. Distance calculation is based on feature extraction from pictures depicting the landmarks, with the localization algorithm relying on minimum two strategically positioned landmarks. To address this requirement, the attached camera scans the area in front of the robot, capturing pictures and analysing them until two landmarks are detected. Hence, computationally intensive, real time image processing is required to achieve highly accurate results. Relevant works include [14] and [15].

B. Path Planning

Many works exist in the literature addressing the path planning problem; a realistic robot navigation and smooth trajectory planning is a major challenge [16], [17]. Planning algorithms generate a trajectory consisting of intermediate reference positions to reach the final target position. In this work, we select and adapt graph-based methods of varying complexity, see, e.g., [11, Chapter 8]. As a result, the algorithms described below, take as input a graph that represents the real-space grid space along with the target positions, the obstacles and the starting position. This grid has a predefined cell size, that depends on the length of the robot. Each cell corresponds to a possible reference position. In our case, the obstacles are rectangular-shaped, in the sense of simplicity, however, arbitrarily-shaped obstacles could also be included.

On the one hand, a lightweight implementation of the $A^\star$ algorithm [18] acts as the Local Path Planner. Similar to [19], four directions of movement are allowed in the grid. The cells containing obstacles are not connected with the neighboring cells. The $A^\star$ algorithm returns a sequence of positions to reach the target position, according to a heuristic cost function; in our case this is the Manhattan Distance. The implementation is suitable for a robot with minimal computational resources providing a solid and quick solution, however the generated trajectory is not smooth.

The computationally intensive algorithm acting as the Remote Path Planner is deployed on the Edge Server. Similar to [20], the main process of the proposed algorithm is to locate a possible move towards a node that is closer to the target given the aforesaid graph. To this purpose, a multiple sources single destination problem is solved, utilising Dijkstra's shortest path algorithm, which calculates a path from each node towards the target position, offline. These precalculated paths, along with the total cost to reach the desired destination, are stored in a database on server's startup. When the Remote Path Planner is invoked, given the current location of the robot, a neighbour pruning is performed similar to [21]. A node of the graph is considered to be a neighbor of the current position if (i) the distance between them is less than twice the specified cell size and (ii) no obstacle is in the line of sight of the current position to that node. Consequently, to retrieve the set of possible neighbours, it is sufficient to search for avoidance of line clipping (intersection) between the line connecting the current position to each of the adjacent cells and the set of obstacles present in the real-space grid. The optimal path is chosen by comparing all possible neighbours. In particular, the cost to reach each one of them from the current position is added to the cost from each neighbour to reach the desired target. In this way, the algorithm allows "shortcuts' to the neighbouring nodes, while any-angle trajectories are feasible.

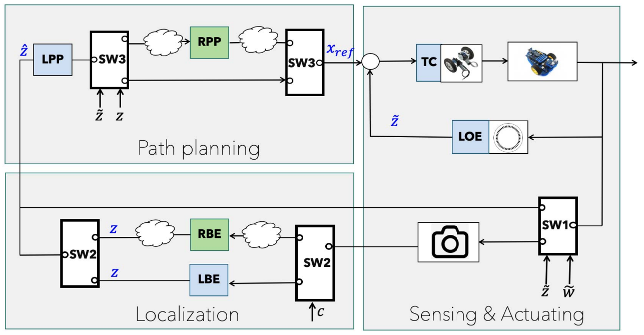

Fig. 4: The block diagram of the switching system. Component abbreviations and colors follow the pattern introduced in Section II.

V. Switching System

In this section, we present the switching mechanisms that are realizing the Offloading Decision Mechanism of our framework. We assume that starting from a position $\boldsymbol{x_0} = [x_1(0) \quad x_2(0)]^\top$, the closed-loop system converges asymptotically to a reference position $\boldsymbol{x_{ref}} = [x_{1,\text{ref}}(t_i) \quad x_{2,\text{ref}}(t_i)]^\top$ when exact measurements are available, i.e., when $\tilde{\boldsymbol{z}}(t) = \boldsymbol{z}(t)$. We identify two offloading opportunities related to the pose estimation and the path planning problem. In Fig. 4 the proposed switching system is presented. In particular, switches $\mathcal{S}_1$ and $\mathcal{S}_2$ relate to the estimation procedure, and switch $\mathcal{S}_3$ concerns path planning part.

A. Sensor selection (Switch 1)

The measurements of the onboard sensors are imperfect, thus the pose estimation error is accumulated. When the error becomes too large, the more precise, yet more computationally intensive remote localization algorithm is invoked. In order to decide when to offload, we introduce the variable $\delta(\cdot)$ that describes the uncertainty in estimation. We set

$$\delta(t_i^{j+1}) = \delta(t_i^j) + b_0 + b_1\tilde{\delta}(t_i^j),$$

$j = 1, \ldots, k_i$, $i \in \mathcal{N}$, where $\tilde{\delta}$ is the deviation between the measurements of the states $\tilde{z}$, computed by the Equations (4) – (6) and the model-based estimations $\breve{\boldsymbol{z}}$, i.e.

$$\tilde{\delta}(t_i^j) = \left\| \breve{\boldsymbol{z}}(t_i^j) - \tilde{\boldsymbol{z}}(t_i^j) \right\|_2,$$

where $\breve{\boldsymbol{z}}(t_i^j)$ consists of:

$$\breve{z}_1(t_i^{j+1}) = \frac{r}{2}(w_L(t_i^j) + w_R(t_i^j)) \cos \breve{z}_3(t_i^j)(t_i^{j+1} - t_i^j) + \breve{z}_1(t_i^j),$$
$$\breve{z}_2(t_i^{j+1}) = \frac{r}{2}(w_L(t_i^j) + w_R(t_i^j)) \sin \breve{z}_3(t_i^j)(t_i^{j+1} - t_i^j) + \breve{z}_2(t_i^j),$$
$$\breve{z}_3(t_i^{j+1}) = \frac{r}{l}(w_L(t_i^j) - w_R(t_i^j))(t_i^{j+1} - t_i^j) + \breve{z}_3(t_i^j),$$

which are the model-based estimation of the dynamics at time instants t_i^j, $j = 1, \ldots, k_i$ and w_L,w_R are the outputs of the tracking controller. At time t_0^0, the model-based estimation is equal to a known initial position, i.e. $\breve{\boldsymbol{z}}_1(t_0^0) = \breve{\boldsymbol{z}}_1^0$. As a result, δ linearly depends on the deviation, and is getting bigger as the robot actuates, especially when the actual motion of the robot differs from what the model dictates.

The offloading mechanism, aiming to reset the uncertainty, is triggered when δ becomes too large, namely larger than a prespecified threshold $\delta^\star$, i.e.,

$$S_1(t_i^{k_i}) = \begin{cases} \text{OFF}, & \text{if } \delta(t_i^{k_i}) \leq \delta^\star, \\ \text{ON}, & \text{else}, \end{cases}$$

where k_i refers to the time instant, when the robot's position, calculated by Equations (4) and (5), is close to the next reference position $\boldsymbol{x_{ref,k}}$. Moreover, ON corresponds to using the beacon-based localization and OFF to proceeding based on the local odometry estimation. In the scope of this work, we assume that the uncertainty becomes equal to zero when the beacon-based localization is used. Hence, when $S_1(t_i^{k_i}) = \text{ON}$, then $\delta(t_{i+1}^0) = 0$, which means we get a valid measurement of the states $\boldsymbol{z}$. Otherwise, $\delta(t_{i+1}^0) = \delta(t_i^{k_j})$.

B. Estimation Offloading (Switch 2)

Switch S_2 decides whether the localization algorithm will be executed locally on the microcontroller mounted on the robot, or remotely on the Edge Server. Although the execution of such a computationally heavy algorithm on a battery-powered IoT device is energy-consuming, it may be preferable in some cases as offloading might result to larger response times due to lack of available resources on the remote server and network congestion.

1) Resource modelling and estimation: We assume that the resources of the localization service on the Edge Server are managed by the resource orchestrator of the infrastructure provider and we can only estimate the allocated resources through measurements. Thus, we model the resource allocation strategy on the Edge Server as a linear dynamical system subject to process and measurements uncertainty disturbances

$$c((k+1)T_s) = c(kT_s) + w(kT_s),$$
$$z(kT_s) = c(kT_s) + v(kT_s),$$

where c accounts for the virtual CPU cores of the container, z is the measurement of c and T_s is a constant sampling time. The terms w, v are the process and measurement noise respectively, both following a normal distribution. Based on previous measurements, we compute a current estimation of the virtual CPU cores allocated to the container, $\hat{c}$, by applying a Kalman Filter [22], which is a computationally light prediction method.

2) Processing time estimation: Having acquired the estimation of the available remote virtual CPU cores $\hat{c}$, the estimated processing time of the beacon-based localization algorithm can be calculated. To this purpose, the processing time, t_p is modeled as a linear relationship of the available resources, $t_p = a\hat{c} + b$. The coefficients a,b are calculated using the least squares fitting method, on a set of pairs $(t_p, \hat{c})$ produced offline while experimenting with a dataset of pictures. Moreover, we consider the wireless network induced delay t_{net} to be constant as a standard network delay in a WLAN network.

3) Localization Offloading: The processing time is related directly to the CPU availability. The local beacon-based localization has an average time t_{loc} to be executed based on the robot's resources. Hence, Switch S_2 is formulated as:

$$S_2(t_i^{k_i}) = \begin{cases} \text{ON}, & \text{if } t_p + t_{net} \le t_{loc}, \\ \text{OFF}, & \text{else}, \end{cases}$$

where k_i refers to the time instant that the robot must decide whether to offload or not the beacon-based localization algorithm. Moreover, ON corresponds to the remote execution of the self-localization algorithm and OFF to the local execution.

C. Path Planning Offloading (Switch 3)

Two path planning algorithms are implemented. By default, the computationally light $A^\star$ algorithm presented in Section IV-B,provides a reference trajectory on the robot. However, whenever a prediction cost indicates a possible amelioration by choosing a more refined path, the remote path planning algorithm is invoked. Both algorithms take as input the current estimation of the position and the reference position and generate a reference trajectory.

The offloading decision for the path planning depends on a cost consisting of two parts; the first part estimates the closeness of the generated reference trajectory to obstacles and the second part evaluates the curvature of the trajectory. Both terms follow theoretical aspects from standard works, e.g., [24]. We define the function $D(\boldsymbol{x})$ that quantifies the "density" of obstacles according to the estimation of the current position $\hat{\boldsymbol{x}}$, either computed by the beacon-based localization or the local odometry measurements.

$$D(\boldsymbol{x}) = \sum_{\hat{\boldsymbol{x}}_{\text{obs}} \in \mathcal{X}_{\text{obs}}} \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}_{\text{obs}}\|\right),$$

and $\boldsymbol{\mathcal{X}}_{\text{obs}}$ is the set of positions that correspond to the centers of the cells that are unreachable, e.g., occupied by an obstacle.

Let $\{\check{\boldsymbol{x}}(i)\}_{i=1,\dots,M}$ be the part of the path sequence consisting of the first M positions, generated by the local path planning algorithm.

The local path planning algorithm takes as input the current position estimation $\hat{\boldsymbol{x}}(t_i^{k_i})$ at $t = T_i^{k_i} + T_i^1$ and creates a reference trajectory sequence $\{\check{\boldsymbol{x}}(i)\}_{i=0,1,\dots,M}$, with $\check{\boldsymbol{x}}(0) = \hat{\boldsymbol{x}}(t_i^{k_i} + T_i^1)$. We define:

$$J_{\text{local}}(\hat{\boldsymbol{x}}(t_i^{k_i} + T_i^1)) = \sum_{i=0}^{M-1} \Big(\left\|\check{\boldsymbol{x}}(i+1) - \check{\boldsymbol{x}}(i)\right\| \Big) - \left\|\check{\boldsymbol{x}}(M) - \check{\boldsymbol{x}}(0)\right\|,$$

as a cost describing the curvature of the reference local trajectory. The offloading strategy can be formulated as:

$$S_3(t_i^{k_i} + T_i^1) = \begin{cases} \text{OFF}, \ \text{if } D(\hat{\boldsymbol{x}}(t_i^{k_i} + T_i^1)) - J_{\text{local}}(\hat{\boldsymbol{x}}(t_i^{k_i} + T_i^1)) \le J^\star, \\ \text{ON}, \ \text{else}, \end{cases}$$

where $t_i^{k_i} + T_i^1$ indicates the time instant after the actuation and pose estimation. The constant $J^\star$ accounts for the degree of difficulty of the next moves in terms of proximity to obstacles and curvature of the trajectory. When S_3 in ON, the remote path planning provides the next step to reach the target position. Otherwise, the robot relies on the local path planning trajectory. It should be mentioned that, contrary to Switch 2, here, we do not include the CPU availability in the offloading decision, as we noticed that the remote path planner chosen is mainly memory intensive.

VI. Experiments and Evaluation

The experiments were conducted in an operating space of 2.5×2.5 meters, divided by 25×25 cells, with a cell size of 10×10cm. The robot chosen was the commercially available AlphaBot[1], equipped with a Raspberry Pi 3 device as the control unit. The length of the AlphaBot is 22cm and the radius of each wheel is 6.6cm. The coloured beacons were placed at the periphery of the grid for the localization procedure described in Section IV. The rectangular-shaped obstacles were placed as depicted with grey colour in Fig. 5. The map is considered known. The Access Point used was a MikroTik wireless SOHO AP, providing up to 100Mbs LAN connection, Single Band (2.4GHz). The Edge Server deployed on the NETMODE, testbed part of Fed4FIRE[2] initiative, was equipped an Intel Atom CPU, up to 1Gbit Ethernet port and 8GB of RAM. The services provided by the edge server were deployed as Docker containers. For each Docker container, one can set constraints, to limit a given container's access to the host machine's CPU cores, by provisioning a percentage of them as the virtual cores of the containers. Thus, containers can be assigned with partial virtual CPUs using decimal values. Using a collection of pictures from the actual experimentation room, from different positions and viewing angles, a dataset was created to estimate the time duration of the remote beacon-based localization. In Table I, the values of the set of pairs $(t_p, \hat{c})$, introduced in Section V-B, are presented. Using the least squares fitting method we calculated the coefficients $a = -1.34$ and $b = 1.675$. Hence, the estimated processing time of the remote beacon-based localization is given by $t_p = -1.34\hat{c} + 1.675$. Provisioning over 1.5 cores resulted in similar computation time, thus, the maximum CPU allocation was set to that value. In our experiments, the allocated cores of the containerized application were updated every 10sec, following a Normal Distribution with a mean value of 0.75 and 0.5 variance. The following values were used for the aforesaid constant values: $b_0 = 1$; $b_1 = 0.2$; $e_1 = 5$cm $e_2 = 5°$, $L_1 = 0.2$, $L_2 = 0.6$, $\delta^\star = 6$ and $J^\star = 3$. Finally, the average network delay of the WLAN was empirically measured to $t_{net} = 1$sec per offloaded picture and the average time for each picture to be processed locally on the AlphaBot was $t_{loc} = 3$sec.

Three experiments were conducted, namely, local only execution, remote only execution and the proposed switching offloading scheme. In Table II the average completion time and the average success rate for 10 experiments of each

[1]https://www.waveshare.com/wiki/AlphaBot
[2]https://www.fed4fire.eu/testbeds/netmode/

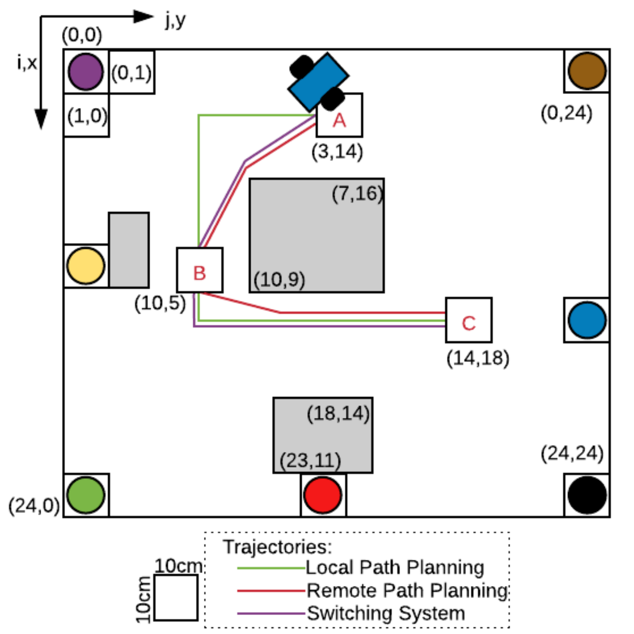

Fig. 5: The experiment setup and the trajectories produced by the three experiments.

Average Time per picture (sec), t_p	Virtual Allocated Cores, $\hat{c}$
2.41	0.25
1.06	0.5
0.56	0.75
0.39	1
0.30	1.25
0.26	1.5

TABLE I: The average time for remote beacon-based estimation per virtual allocated core to the container.

setting is presented. For the rest of the evaluation, we will present the results of the best trials for each setting. Moreover, in Fig. 5 the reference trajectories of these trials for the three experiments, are illustrated, with green colour for local only execution, red colour for remote only execution and purple colour for the switching system. As outlined in Section IV, the local $A^\star$ algorithm allows only four directions of movement, while the remote path planner allows any-angle movements. For better visualization, we uploaded timelapse videos[3] from the conducted trials for each setting. In these experiments, the starting position for the AlphaBot was the already known position $A(3, 14)$, while the desired target reference positions were $B(10, 5)$ and $C(14, 18)$ in sequence. The scale of uncertainty is illustrated as a percentage of $\delta^\star$, i.e. $\delta/\delta^\star$, which is the predefined quantity for Switch 1 to be ON.

1) Experiment A - Local Only Execution: In the first experiment Switches 1 and 3 were ON, throughout the experiment and Switch 2 was never used. This setting results

[3]https://github.com/Dspatharakis/alphabot-ppl/tree/master/timelapsed-videos

Experiment	Average completion time (sec)	Success Rate
Local Only Execution	61	40%
Remote Only Execution	105	100%
Switching System	90	100%

TABLE II: The average completion time and success rate of 10 experiments for each setting.

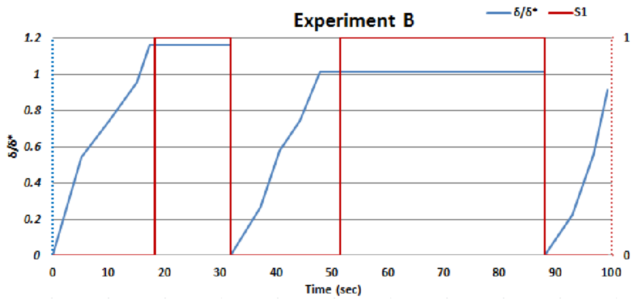

Fig. 6: Experiment B - Remote Only Execution.

to a fast, although not precise navigation with $\delta/\delta^\star$ growing monotonically. The average duration was 61 seconds as the main time consuming process was the actuation. The amount of successful trials was low. Consequently, without a more sophisticated localization algorithm and a more precise path planning technique there is no guarantee the target reference position is reached.

2) Experiment B - Remote Only Execution: In the second experiment, whenever the uncertainty about AlphaBot's pose grew over the predefined threshold $\delta^\star$, beacon-based localization was invoked (Switches 1 and 2 ON) on the Edge Server. Moreover, the reference trajectory was always generated by the remote path planning algorithm (Switch 3 ON). In this setting, the robot always reached the target positions, as shown in Table II, although the completion time was heavily affected, as shown in Fig. 6. Beacon-based localization was executed twice during this experiment and, as a result, $\delta/\delta^\star$ became equal to 0. The setup of the particular experiment underlines the importance of a slower but more precise navigation.

3) Experiment C - Switching System: As described in Section V-C, Switch 3 decides which path planning algorithm solution the AlphaBot will use to generate the next reference position. When, the curvature function of the trajectory calculated by the $A^\star$ algorithm and the obstacle density function exceeded the threshold value $J^\star$, the remote path planning solution was selected; e.g., from the beginning of the experiment until the 25th sec of the simulation and from the 43rd sec till the 67th sec, as illustrated with green dashed line in Fig. 7. In the same figure, with red solid line, $\delta/\delta^\star$ is depicted. Two times during the experiment the more precise beacon-based estimation was invoked to reset $\delta/\delta^\star$. The first estimation attempt, at the 25th sec of the experiment, was executed on the Edge Server, because S2 was ON. The second one, at the 71st sec of the experiment, was executed locally, as S2 dictated (OFF), because the estimation of the CPU availability of the Edge Server, provided by the Kalman Filter, along

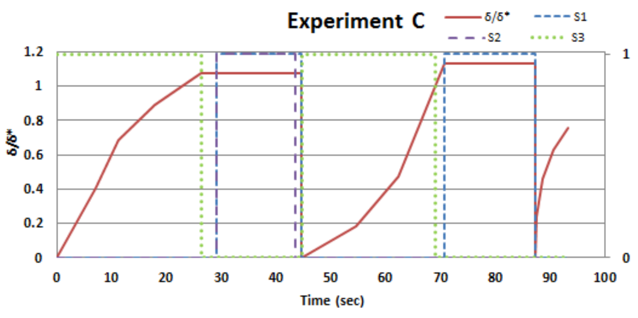

Fig. 7: Experiment C - Switching System.

with the network delay for each picture, at that time, would have provided worse results than the local execution. This setup provided a very precise and robust navigation for the robot, leading to a very high success rate of the experiments, achieving a balance between execution time and trajectory accuracy.

VII. Conclusion

In this study, we introduced a switching offloading mechanism for localization and path planning applications of mobile robots. The offloading decision for localization is based on pose uncertainty and the availability of edge resources, while the offloading decision for path planning depends on the difficulty of the trajectory. The proposed framework achieves more precise navigation than the case of exclusive local execution of the applications, without paying the price of a slower execution time, like in the case of remote only execution of the algorithms. Also, it is modular and applicable to various scenarios, applications and objectives under the robot's dynamic environment. Our future work will focus on extending the proposed mechanism to more sophisticated control algorithms, providing theoretical guarantees for stability and convergence of the proposed robot's dynamics. Furthermore, we plan to develop more precise estimation and planning algorithms in multi-robot scenarios and more sophisticated control algorithms in the co-design setting that will take into account the available resources on the infrastructure side.

Acknowledgment

The research work of Mr. Dechouniotis is co-financed by Greece and the European Union (European Social Fund-ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of the project "Reinforcement of Postdoctoral Researchers - 2nd Cycle" (MIS-5033021), implemented by the State Scholarships Foundation (IKY).

References

[1] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1628-1656, thirdquarter 2017.

[2] M. Avgeris, D. Spatharakis, D. Dechouniotis, N. Kalatzis, I. Roussaki, and S. Papavassiliou. "Where there is fire there is SMOKE: a scalable edge computing framework for early fire detection." Sensors, vol. 19, no. 3 (2019): 639.

[3] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying Fog Computing in Industrial Internet of Things and Industry 4.0," IEEE Trans. Ind. Inf., vol. 14, no. 10, pp. 4674–4682, Oct. 2018.

[4] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: principles, architectures, and applications," in Internet of Things, Elsevier, 2016, pp. 61–75.

[5] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)," IEEE Comm. Stand. Mag., vol. 2, no. 2, pp. 55–61, Jun. 2018.

[6] D. Song, A.K. Tanwani, K. Goldberg and B. Siciliano, 2019. "Networked-, cloud-and fog-robotics", Springer.

[7] N. Tian et al., "A Cloud-Based Robust Semaphore Mirroring System for Social Robots," in 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, Aug. 2018, pp. 1351–1358.

[8] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A Fog Robotics Approach to Deep Robot Learning: Application to Object Recognition and Grasp Planning in Surface Decluttering," in 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, May 2019, pp. 4559–4566.

[9] A. Botta, L. Gallo, and G. Ventre, "Cloud, Fog, and Dew Robotics: Architectures for Next Generation Applications," in 2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Newark, CA, USA, Apr. 2019, pp. 16–23.

[10] L. A. Trinh, N. D. Thang, N. V. D. Hau, and T. C. Hung, "Position rectification with depth camera to improve odometry-based localization," in 2015 International Conference on Communications, Management and Telecommunications (ComManTel), DaNang, Vietnam, Dec. 2015, pp. 147–152.

[11] S. M. LaValle, Planning Algorithms. Cambridge: Cambridge University Press, 2006.

[12] D. Pizarro, M. Mazo, E. Santiso, M. Marron, D. Jimenez, S. Cobreces, and C. Losada, "Localization of mobile robots using odometry and an external vision sensor," Sensors, vol. 10, no. 4, pp. 3655–3680, 2010.

[13] M. Avgeris, D. Spatharakis, N. Athanasopoulos, D. Dechouniotis and S. Papavassiliou, "Single Vision-Based Self-Localization for Autonomous Robotic Agents," in 2019 7th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Istanbul, Turkey, Aug. 2019, pp. 123–129.

[14] D. C. K. Yuen and B. A. MacDonald, "Vision-based localization algorithm based on landmark matching, triangulation, reconstruction, and comparison," IEEE Trans. Robot., vol. 21, no. 2, pp. 217–226, Apr. 2005.

[15] A. Bais and R. Sablatnig, "Landmark Based Global Self-localization of Mobile Soccer Robots," in Computer Vision – ACCV 2006, vol. 3852, P. J. Narayanan, S. K. Nayar, and H.-Y. Shum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 842–851.

[16] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-Angle Path Planning on Grids," jair, vol. 39, pp. 533–579, Oct. 2010.

[17] C. Chen, M. Rickert, and A. Knoll, "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering," in 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, South Korea, Jun. 2015, pp. 1148–1153.

[18] P. E. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans. Syst. Sci. Cyber., vol. 4, no. 2, pp. 100–107, 1968.

[19] J. Peng, Y. Huang, and G. Luo, "Robot Path Planning Based on Improved A* Algorithm," Cybernetics and Information Technologies, vol. 15, no. 2, pp. 171–180, Jun. 2015.

[20] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., Jurišica, L. (2014). "Path planning with modified a star algorithm for a mobile robot," Procedia Engineering, 96, 59-69.

[21] D.D. Harabor, A. Grastien, (2011, August). "Online graph pruning for pathfinding on grid maps,", in Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011

[22] R. Kalman, "A new approach to linear filtering and prediction problems," Journal of basic Engineering vol. 82(1), pp. 35–45, 1960.

[23] C. G. Cassandras and S. Lafortune, Introduction to discrete event systems, Second edition. New York, NY: Springer, 2010.

[24] T. Fraichard and A. Scheuer, "From Reeds and Shepp's to Continuous-Curvature Paths," IEEE Trans. Robot., vol. 20, no. 6, pp. 1025–1035, Dec. 2004.