A Tradeoff Analysis of a Cloud-Based Robot Navigation Assistant Using Stereo Image Processing

Javier Salmerón-García, Pablo Íñigo-Blasco, Fernando Díaz-del-Río, and Daniel Cagigas-Muñiz

***Abstract*—The use of Cloud Computing for computation offloading in the robotics area has become a field of interest today. The aim of this work is to demonstrate the viability of cloud offloading in a low level and intensive computing task: a vision-based navigation assistance of a service mobile robot. In order to do so, a prototype, running over a ROS-based mobile robot (Erratic by Videre Design LLC) is presented. The information extracted from on-board stereo cameras will be used by a private cloud platform consisting of five bare-metal nodes with AMD Phenom 965 $\times$ 4 CPU, with the cloud middleware Openstack Havana. The actual task is the shared control of the robot teleoperation, that is, the smooth filtering of the teleoperated commands with the detected obstacles to prevent collisions. All the possible offloading models for this case are presented and analyzed. Several performance results using different communication technologies and offloading models are explained as well. In addition to this, a real navigation case in a domestic circuit was done. The tests demonstrate that offloading computation to the Cloud improves the performance and navigation results with respect to the case where all processing is done by the robot.**

***Note to Practitioners*—Cloud computing for robotics is very promising for several reasons, like robot's energy saving, larger storage capacity, stable electric power, better resource utilization and the difficulty of upgrading the robots' embedded hardware. The presented application extracts 3D point clouds from the stereo image pairs of a camera situated on the robot. Using these 3D points, a shared control is implemented to help the remote teleop-eration of a robot. That is, the commands sent by a joystick are attenuated when a possible collision is detected (by checking the future commanded trajectory against the 3D points). All of these computationally heavy tasks (difficult to perform by a mobile robot) are done in the cloud. The offloading models proposed in this paper are generic enough to be used in other applications. Obtained results show that further improvement in communica-tion technologies will suppose a significant performance boost for offloading computation.**

***Index Terms*—Cloud offloading, cloud robotics, image point cloud, mobile robots, navigation assistance, shared control.**

I. Introduction

CLOUD computing is an emerging technology for robotics, especially for mobile service robots. The needs come from the huge amount of information that a service robot has to process in order to interact and interpret the environment correctly. For this reason, during the last few years, an important number of research papers and projects are addressing the use of cloud infrastructures in robotics [1]–[4]. Research fields where clouds are of interest are those where computation is very intensive. But, where and how can high computing tasks be identified in mobile robots? According to [5], a robot usually has a layered architecture. Layers in the top level of the hierarchy can contain processes that, for example, perform cognitive tasks similar to humans. In the middle layers, tasks also involve complex processes like path planning, object handling, speech recognition, etc. Finally, in the lowest levels reactive and real-time control operations are performed (e.g., obstacle avoidance, guidance, beacon detection, signal communications processing, etc). The amount of computation is not necessarily proportional to the level. For instance, in [6], it is pointed out that an intelligent mobile robot in an office-like environment can be modeled by the Soar cognitive architecture with only a few milliseconds of computational cost. However, other middle and low-level tasks that feed higher layers in a robot architecture, can be very computing demanding. The main example is that of artificial vision and higher level tasks arising from it, such as object detection, recognition and tracking, surveillance, gesture recognition, etc. Another very promising field is that related to multi-robot cooperation at different levels, where new cooperative algorithms are being developed like multi-robot simultaneous localization and mapping (SLAM) [7], map merging (acquired by several robots), networked information repository for robots [8], amongst others.

The main properties that make cloud infrastructures compelling are the following: high reliability, larger storage capacity, stable electric power, reutilization of hardware, dynamic scalability and better resource utilization. In particular, the dynamic scalability property (adaptation of the computing power at runtime) is extremely useful in Robotics, because it allows the almost instant incorporation of new computation demanding algorithms as soon as they are implemented. Besides, the term “Cloud Robotics” has emerged to include this area, which promises fast development of complex distributed robotics tasks in the forthcoming years.

Consequently, the use of cloud computing is expected to be widespread in the next few years in the service and rehabilitation robotics field. Its use can offer them additional advantages,

such as: supplementing local information collected by the robot with that coming from Intelligent Environments, Ambient Assisted Living applications like monitoring user's health and daily activities, multirobot cooperation and so on. In this respect, it must be noticed that images captured by on-board cameras are to be required by higher robotics levels (like object detection and recognition, gesture recognition, etc., which must run in the cloud), or by a human teleoperator that must make a decision over the robotics system or simply teleoperate the robot itself. Therefore, these images will be transmitted to the cloud in any case, and these transfers would not be a burden for the whole system.

Navigation assistance and human-computer shared control, are common application cases for intelligent wheelchairs. Many wheelchairs incorporate high-ended sensors, like stereo cameras, laser rangers, and Ambient Intelligence aids [9] to address this question. However, despite great advancements in power wheelchair technology, research shows that wheelchair related accidents occur frequently, especially for users with considerable handicaps [10]. Daily wheelchair maneuvers could be challenging due to the users' pathologies, poor maneuvering skills, user's fatigue, and unknown or adverse environments. Therefore, a safe maneuverability of a wheelchair using on-board or external high-ended sensors has great importance in this kind of situations. Moreover, remote teleoperation of the wheelchair by an external caregiver could be sometimes necessary.

In addition to this, the results obtained in our Lab research in advanced wheelchairs control [11], [12] support the necessity of navigation assistance in certain situations. Because of the probable omnipresence of cloud infrastructures for service robots, in this work a prototype is implemented as a first step to analyze and demonstrate the viability of carrying out in a cloud the lower (but intensive computing) levels of a mobile robot navigation assistant. The presented prototype is running over a Robotic Operating System (ROS)-based mobile robot (Erratic by Videre Design LLC) due to its software deploying and debugging ease. However, this idea will be applied to other devices like intelligent wheelchairs or other mobile service robots, extending our previous research. More precisely, the task consists of the teleoperation by a local or remote (which has little influence because the inherent distributed architecture of the developed system) user, who is helped by using the sensing information extracted from an on-board (but processed in a cloud infrastructure) stereo camera. In order to exploit the cloud capabilities, a dynamic parallel algorithm has been implemented, so the solution is able to scale out and back. Hence, the robot gets the results faster when more computation power is required.

The Computation offloading of a robotics task includes several tradeoffs. The software architecture (and its components) of a complex robotic system must cater for a variety of characteristics, distinguish it from other systems. The most relevant are [13]: concurrent and distributed architecture, modularity (several components of high cohesion but low coupling), robustness and fault tolerance, and real-time efficiency. The first two characteristics primarily benefit from cloud offloading, while the third introduces new challenges (network robustness and fault tolerance appear as a new aspect to considerate). Nevertheless, the main issue that offloading must address for a navigation task is the fourth of them. In this respect, a short quantitative comparison of times involved in local versus remote computing follows. This demonstrates the theoretical real-time viability of cloud offloading and it points out new considerations.

The first mandatory bound is the time spent in data transmission to the cloud. If the bandwidth rate of communication technology is BW, and data size to be transmitted and received is D, it is concluded that D/BW must be inferior to the deadline of the task. Nowadays, there are wireless networks whose bandwidths [14] are approaching to 1 Gbps, with a steady incremental ratio of more than 40% per year, or two times every two years (see http://www.wi-fi.org/). For a typical navigation control loop, frequencies around 20 Hz (period $T = 0.05$ s) are usually enough. This gives us a limit of 0.05 Gbits, or equivalently, a 3 Mpixel raw B/W stereo image, per period. Therefore, offloading computing is feasible, which, indeed, has the aforementioned benefits.

A secondary aim is that cloud accelerates timing execution. Let us suppose that the robot computer can execute at a rate of IPS instructions per second (millions of IPS or $MIPS$ is a common magnitude in computer architecture [15]), and that the cloud can speedup an application S times more that this IPS. Therefore, local and remote execution times for N_I computer instructions could be expressed as

$$t_{\text{local}} = \frac{N_I}{IPS}; \quad t_{\text{remote}} = \frac{N_I}{IPS \cdot S} + \frac{D}{BW}.$$

For stereo vision applications, S is very big because the cloud is supposed to have far more computational resources than the local computer, and many operations are performed in parallel. Being that the case, timing comparison gets to a short formula, which indicates whether computation offloading is faster than local execution, that is, $t_{\text{local}} > t_{\text{remote}}$, if

$$\frac{N_I}{D} > \frac{IPS}{BW}.$$

The first term of this inequality is entirely dependent on each application. This means that high intensive computing tasks benefit from cloud computing. With respect to image processing, many of the filters that extract features used in point cloud processing have computational complexity orders higher than $O(N)$, being N the number of pixels. For example, in our experiments, a frame pair is computed by the Erratic in $t_{\text{exec}} = 0.96$ s (see Section V). As the Erratic CPU (Intel Celeron-M) runs at a frequency of $f = 1.4$ GHz and has a Clocks per Instruction (CPI) around 2.0 [16], then, the number of instructions is [15]

$$N_I = \frac{(t_{\text{exec}} \cdot f)}{CPI} = 6.72 \cdot 10^8 \text{ instructions.}$$

Besides, transmitted data in this experiment (see Section V) mainly consists of a color 1024×768 frame pair, that is: $D = 1024 \cdot 768 \cdot 3 \cdot 2 \cdot 8 = 3.77 \cdot 10^7$ bits. Hence

$$\frac{N_I}{D} = 17.8 \text{ instr/bit.}$$

On the contrary, the second member of the inequality is mainly technology dependent. Nowadays, IPS is f/CPI [15], so for the Erratic CPU, it rounds $IPS = 7.0 \cdot 10^8$ instr/s. Therefore, networking bandwidth is crucial to have success in the offloading. Using WiFi IEEE 802.11ac, 400 Mbps data rate

transmissions can be easily achieved and, hence, the second term has a very much lower value (1.75 for actual case) that the first one, which promises a successful offloading. On the whole, it can be concluded that many tasks from the top, middle, and even lowest, level of a common layered robot architecture are presently candidates to be remotely executed, or they would be it in a few years.

The rest of this paper is organized as follows. Section II summarizes several related works. Next, two sections analyze the practical case study: Section III explains the architecture of the system, especially the navigation assistant and a timing analysis. This timing analysis is vital for studying the computation offloading possibilities of the presented solution, which will be explained in Section IV. Experimental results are shown in Section V to quantify the benefits of the cloud approach, and finally conclusions are summarized in Section VI.

II. Related Work

Cloud robotics has two major lines of work. The first one corresponds to the creation of an "Internet for robots" [8], where all robots extend their knowledge using a predefined language and collaborative build and merge/retrieve information [17]–[20]. The second line of work (though not being completely separated of the former) is the one studied in this work: computation offloading. This paradigm is being used by many works and projects that accomplish high-level vision tasks, though they do not have real-time requirements [2], [3], [21]. However, some papers have appeared in the last few years that propose the external processing of several parts of the sensor feedback information, achieving close to real-time performance for these operations. In [22], they combine GPUs and cloud offloading (to a private cloud infrastructure) to perform SLAM, with successful results in different environments. They study different virtual resources configurations as well: 1 virtual machine per bare metal node, several VM's per node, amongst others, finding that virtualization overheads imply a degradation in speedup (they use Xen as virtualization technology). In this sense, the method for identifying the optimal balance between a cloud system overhead and performance presented in [7] can be useful. Executing SLAM in the cloud is also studied in [23], where they develop a cloud mapping framework (C2TAM). They combine both computation offloading and collaborative work, as the framework allows fusing the information obtained from several robots. They work with a 640×480 pixel RGBD camera and an average data flow of 1 MB/s, below 3 MB/s, which is the usual wireless bandwidth and hence the mapping is successfully done (moreover, they work with keyframes, reducing the amount of images to send). Another computation offloading example is proposed in [24], where a high-resolution SIFT-based object detection is speeded up by transmitting on-board preprocessed image information instead of raw image data to external servers. The configuration of these external servers is specific to this work, so some properties of the cloud computing paradigm are not exploited.

The idea of Computation Offloading is studied in [25], and an estimation of the computation and communication times needed for the recognition and object tracking tasks is presented in order to minimize the total execution time (approaching the real-time constraints). Their analysis permits making offloading decisions for object recognition for different bandwidths, background complexities, and database sizes.

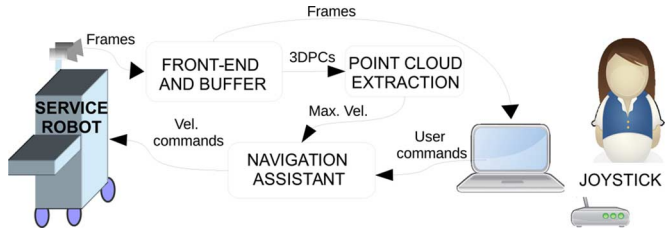

Fig. 1. Block diagram of the Cloud-Based Navigation Assistant for the teleoperation of a mobile robot. The different types of data transfer are the following: Stereo frame pairs, 3DPC, robot velocity commands, and user input commands. Also, bond status messages are required by ROS bond library.

Authors of [26] present an object-tracking scenario for a 14-DOF industrial dual-arm robot using a UDP transport protocol for transmitting large-volume image over an Ethernet network. Thanks to the very low sending and cloud image processing times that are achieved, a stabilizing control law can be implemented, with time-varying feedback time delay.

The work [27] also asks whether the performance of distributed offloading tasks can be compared with those executed on-board. The experiment performed consists of a simple controller that guides the robot in order to follow a line according to the images acquired from a single low-resolution (320×240) camera that points to the floor.

Compared with the described literature, the work presented in our paper is the first that tries to analyze the cloud offloading of such a real-time task as navigation assistance. In addition to this, a thorough explanation of all the offloading possibilities, together with the role of communication technologies, adds more novelty to our work.

III. Overview of the System

The analyzed robotic application consists of a teleoperated mobile robot using a shared control via on-board cloud-based stereo vision (continuing the work presented in [28]) and the robotic software framework ROS (http://www.ros.org). The experience of our Lab in shared control for wheelchairs [29] has inspired this present proposal, whose aim is to enable inexperienced or handicapped pilots to safely drive vehicles in challenging scenarios.

Section III-A summarizes the different modules of the application and Section B explains the stereo vision processing implementation. Section III-C explains how the navigation assistance system works and Section III-D addresses the timing analysis of the computation offloading.

A. Block Diagram of the System

Fig. 1 shows a simple diagram of a vision navigation assisted robot. The robot carries a stereo camera which sends frames to the teleoperator and to those processes responsible for 3D Point Cloud (3DPC) extraction. Once a 3DPC is obtained from a stereo frame pair, it is sent to a navigation assistant node. This navigation assistant node will also receive input commands from the local command interface (a joystick in this case), which are translated to the desired linear and angular speeds (v_j, ω_j) With the previously received obstacle information (that is, the

3DPC), and the historic buffering of robot velocities, the navigation assistant will be able to calculate the correct $v_{\max}$ that avoids an obstacle collision. Once this $v_{\max}$ has been computed, a velocity command (v_c, ω_c) which satisfies the same curvature suggested by the user through the joystick, is sent to the robot. This fusion of information is usually called "continuous shared control" [29], which is frequently preferable for most navigation assistance systems, because the desired commands were smoothly and continuously combined with a collision avoidance criteria [30].

The distributed implementation and the discussion of where should each node run, whether the robot or the cloud, are more deeply discussed in Section IV.

B. Stereo Image Processing

Processing stereo images is currently a very heavy computation task. However, the use of stereo cameras as the sensing technology implies several advantages against other simpler sensors (such as infrared or ultrasonic sensors). First, the information obtained with these cameras is more complete. Second, cameras have usually wider fields of vision. In addition to this, as explained in Section I, the stereo pairs will serve for the teleoperator or will presumably be required by other high-level application. Hence, images need to be transferred out of the robot in any case. Nevertheless, there is always the possibility of combining the visual processed information with that of more basic and reactive sensors. Besides, RGB-D cameras are being developed quickly and at relatively low prices [31]. However, low priced RGB-D cameras are currently aimed for the game market, and, hence, they present several disadvantages. Regarding to our system, their drawbacks are: they do not work well when there is lot of sunlight (even indoors), their FOV is much more limited, maximum and minimum distance detection is bounded, amongst others.

For a successful navigation assistance, the frequency of stereo frame processing must be sufficiently high, whereas its latency small enough. This requirement can become very difficult to meet when more accurate reconstruction of the environment is demanded. Hence, the heavy task of 3DPC extraction must be processed in a powerful computing system. Furthermore, this processing must be designed, not only to be parallel, but also to exploit the special properties that a cloud system offers (more importantly, that of dynamic scalability).

The 3DPC extractors (Fig. 1) are implemented thanks to a ROS package called *stereo_image_proc*, which uses two large-scale, open source libraries for 2D/3D point cloud processing and computer vision libraries: the *Point Cloud Library* (PCL, http://www.pointclouds.org) and *OpenCV Library*. More precisely, this *stereo_image_proc* ROS package offers a node which takes a pair of synchronized stereo frames and, after rectifying the images, produces a point cloud with all the 3D information. ROS implements an inner pipeline (using ROS nodelets) with several stages: image debayer, image rectification, disparity map creation and point cloud building. As each stage is dependent from the previous one, no parallelism can be achieved for an unique frame pair.

In order to make the image processing dynamically scalable, a parallel solution for different frame pairs has been

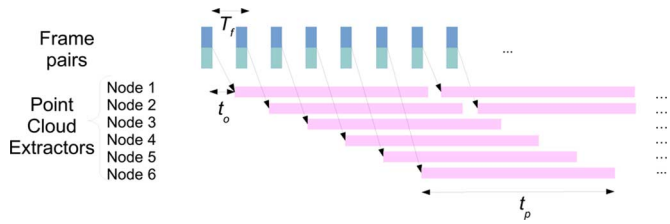

Fig. 2. Stereo frame pipeline process. 3DPC Extractor nodes process (in a pipeline fashion) the frame pairs that the front-end node delivers.

implemented. The idea developed is based on the ability to replicate the 3DPC extractor nodes (see Fig. 2), in several virtual instances in the cloud. Each stereo frame pair will be sent to a different node in a round-robin fashion. If t_p were the time to process a frame pair and T_f the frame acquiring period, the minimum number of nodes would be $\lceil (t_P/T_f) \rceil$ (in practice some additional nodes are added due to the variance of processing times). This scattering method requires a front-end frame buffering node ("Buffer" node in Fig. 1) that will be responsible of determining how many 3DPC extractor nodes are alive at any moment, as well as distributing the stereo stream. This parallel solution increases the performance of the whole system, which is especially evident when bigger frame resolutions are used. The main advantage of this solution is its inherent dynamic nature. If the solution needs to scale out, more 3DPC extractors can be launched at runtime. Once the buffering node detects them (thanks to ROS *bond* library, using bond status messages), they will start receiving stereo pairs for processing. Reversely, if the system must be scaled back, some nodes will be shut down and hence resources would be freed.

C. Overview of the Navigation Assistance

The "Navigation assistant" node receives the 3DPC, which includes the obstacle information. A shared control that continuously filters user commands by means of the obstacle information is implemented as follows. Due that frontal view of cameras only allows controlling small curvature arcs, a simplified version of [29] is used here. Nevertheless, as the user can receive visual feedback of the operated scene, he/she can command pure turns in those areas where no lateral obstacle have been seen by her/him.

The first step is the projection of all those points in the XY plane. After that, for each point (x_p, y_p) in the 3DPC, a trajectory from the driven wheel axle center O (axis origin in Fig. 3) is calculated. This process is done by finding a circumference whose radius has endpoints at O and at (x_p, y_p). This circumference supposes a feasible circular path of the robot, whose center is $(x_c, 0)$ and whose equation is

$$(x - x_c)^2 + y^2 = R.$$

As the circumference has endpoints at $(0, 0)$ and (x_p, y_p), the radius R, curvature value K and x_c can be calculated as follows:

$$x_c = \frac{x_p^2 + y_p^2}{2 \cdot x_p}, \quad R = |x_c|, \quad K = 1/R.$$

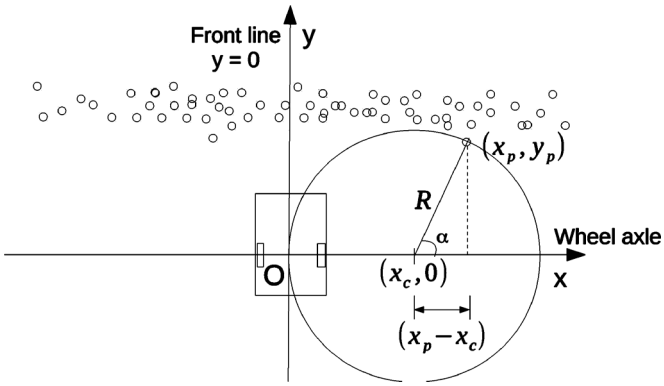

Fig. 3. Plot of the navigation assistant approach. The upper circles are points of the Point Cloud. The circumference that crosses a point and the middle driven wheel axis is calculated.

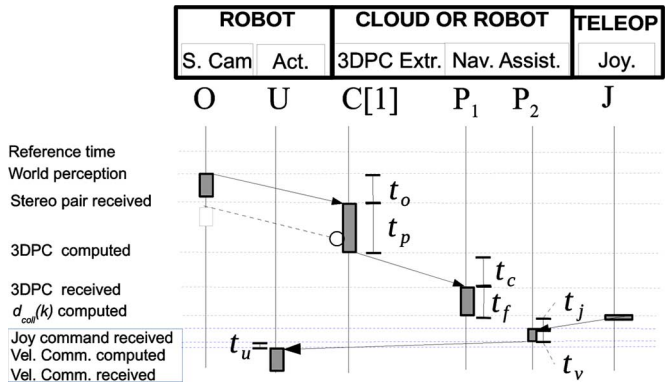

Fig. 4. Time diagram of the system. Camera O captures a frame pair that sends it to the Point Clod extraction C. This extracts the 3DPC and sends it to the Navigation assistant, which fuses this information with the user command coming from the joystick J. Finally, the actuators U receives the velocity commands. Different interval times increment the total latency of the system.

Finally, using the angle α between X axis and the radius, the arc distance from O to (x_p, y_p) is obtained (see Fig. 3)

$$s = \Pi R - \alpha R = R \cdot (\Pi - atan2(y_p, x_p - x_c))$$
$$s = y_p, \quad if\ R \to \infty.$$

For each point (x_p, y_p), a pair (K, s) is calculated, and thus it is possible to interpolate a function $s = d_{\text{coll}}(K)$ with all this information, that is, the distance to collision for each curvature. Every time a new stereo pair is received, $d_{\text{coll}}(K)$ will be updated.

Simultaneously, the user is sending joystick commands, which are proportionally translated to a desired pair of linear and angular velocities (v_j, ω_j), and a curvature $K_j = \omega_j / v_j$. This desired curvature is mapped to a distance value $s_j = d_{\text{coll}}(K_j)$.

Using the uniformly accelerated motion equations, the maximum linear velocity $v_{\max}$ that guarantees a stop without collision can be obtained: $v_{\max} = \sqrt{2 \cdot a_{\max} \cdot s_j}$, where $a_{\max}$ is the maximum braking acceleration of the robot. Every time $d_{\text{coll}}(K)$ is updated, $v_{\max}$ is reviewed and updated. If $v_{\max}$ is less than v_j, the input command cannot be allowed, and the speed sent to the robot is $v_c = v_{\max}$. Angular speed ω_c is calculated so the resulting pair sent to the robot (v_c, ω_c) retains the same curvature asked by the user. This way the teleoperator does not feel that the navigation aids change the desired trajectory. Should $d_{\text{coll}}(K)$ be updated fast enough, this event would occur naturally, and the robot brakes progressively. Otherwise, the navigation assistant will automatically send brake commands at a certain frequency.

D. Communication Time Analysis

When working with real-time navigation, an analysis over the average latency is required. Most of the latency comes from the time the system takes to process the visual information. A simplified timing diagram of system is shown in Fig. 4 (the front-end buffer node is not shown because its delay times are negligible with respect to the other involved times). The robot's on-board computer comprises the stereo camera O and the motor actuation subsystem U. Depending on the offloading model, the following nodes can be either in the robot or in the cloud (Section IV analyzes these possibilities): the point cloud extraction C and the navigation assistant (P_1 and P_2). P_1 is responsible for creating and updating the $d_{\text{coll}}(K_j)$ function explained in Section III-C and P_2 fuses the information from the joystick J with $d_{\text{coll}}(K)$.

Pairs of frames are continuously sent from camera node O to C (via the buffer) at a specified frequency. The transfer time of this message is named t_o. As explained in Section III.-A, the C nodes receive the stereo pairs in a round-robin fashion (in the figure only one node $C[1]$ is represented for simplicity). After computing time t_p, this new extracted 3DPC is sent to the navigation assistant P_1 (taking t_c). There, the function $d_{\text{coll}}(K)$is obtained and updated in a time t_f. On the other side of the figure, each time the joystick J sends a command, a new action will be computed by P_2. Time intervals t_j and t_v involved in this sending, are negligible with respect to the others. This action is sent back to the robot, where module U applies the desired speeds to the actuators. These speeds need to be recalculated until a new action arrives, as the period of the actuation subsystem is inferior to the time to transfer a new frame pair. At the moment, a simple interpolation taking into account the real robot speeds and the last sensed obstacle map is carried.

If, for instance, one new stereo pair tried to enter the *stereo_image_proc* pipeline (see Section III-C) and no nodes were free, this new pair would be discarded. In the case of only one node C, as the processing times t_p in Fig. 4) are usually longer than the period of O, many frames would be discarded (shown like clear rectangles in the figure) until the node were idle again. In order to cut down the number of discarded frames, this timing analysis gives us another bound to the minimum number p of 3DPC extractor nodes. As images are to be sent by a single physical channel (that is, wirelessly), transfer times (t_o) are not overlapped. Due to this, frame acquiring period T_f cannot be inferior than t_o. Furthermore, the mean time to process a pair of frames (t_p/p) must be less than the transfer time (t_o) to send it. In order to get rid of this issue and, as we are working with high scalability systems, p is overestimated. Hence, $t_p/p < t_o$ is always guaranteed.

Another two critical aspects arise from the proper network characteristics: 1) a strict periodical controller cannot be implemented as WiFi networks (proper candidate for mobile robots) have considerable fluctuations in transmission times and 2) the presence of delays in control loops tends to produce oscillations. Nevertheless, these oscillations are overcome by two reasons.

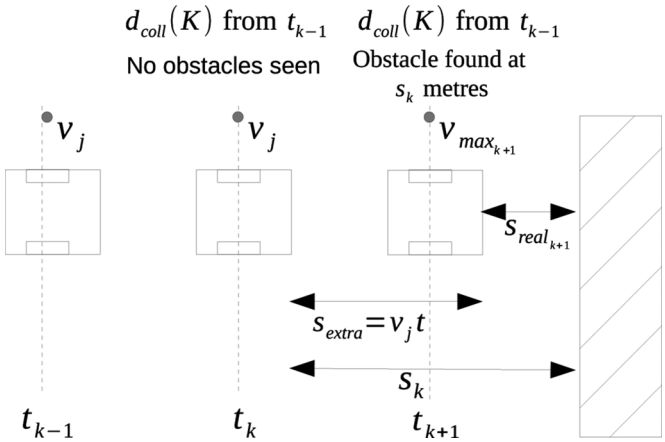

Fig. 5. Example case of delay correction in distance to collision s calculation. As the obstacle information is delayed, time predictive corrections must be applied. At t_{k+1}, the system would calculate that an obstacle is at s_k meters, but the robot is actually closer. A simple predictive correction consists of subtracting s_k an estimated distance $s_{\text{extra}} = v_j \cdot t$, where $t = t_{k+1} - t_k$, and v_j is the commanded speed.

First, the effect of these oscillations are mitigated or counteracted by the user, because he/she is conscious of this kind of problem when remotely driving a slow robot. Second, some delay corrections have been incorporated (see Fig. 5) in the actuation subsystem. As the information obtained from the stereo cameras is delayed, obstacles are actually nearer than $d_{\text{coll}}(K)$ states. Let us suppose that the system calculates at t_{k+1} that an obstacle is at s_k meters. Due to the timing latencies (see Fig. 4), the robot is actually closer. To be conservative, we have supposed that during this latency time the robot has moved at the maximum speed asked by the user, named v_j. Therefore, a simple predictive correction consists of subtracting s_k an estimated distance $s_{\text{extra}} = v_j \cdot t$, where $t = t_{k+1} - t_k$.

One final critical issue must also be considered: the case where the actuation subsystem does not receive any command for a long time. This case is addressed automatically due to the way the interpolation is implemented. The idea is that the robot speeds are gradually reduced at each actuation period to prevent a collision, according to the last obstacle reading. In this case, the robot speeds will tend to zero. There is an even more critical circumstance: when last camera processing did not find any obstacle, so P_2 sends to U an obstacle-free signal, and no other obstacle information is received anymore. Due to this, a deadline internal time is considered by U: if it is exceeded, robot speeds are progressively decremented to prevent a collision.

IV. Computation Offloading Analysis

This section analyzes and debates how the cloud can serve to offload the implied computation of the previously explained robotic application. It should be noted that the discussion presented here is generic enough to be applied to other applications. For example, a cloud offload autonomous robot has similar processing nodes to those shown in this section.

As stated in Section I, there are inherent tradeoffs when moving computation into the cloud: the communication overheads, the amount of computing resources used, along with

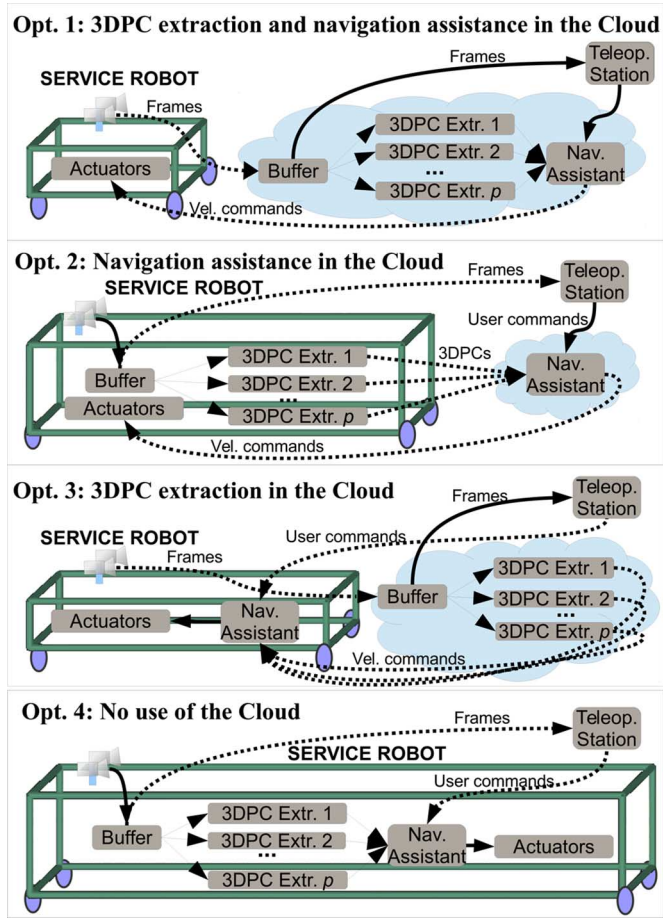

Fig. 6. Possible configurations for Cloud offloading. Mandatory wireless communications (those which origin or destination is the robot) are represented by dotted arrows while communications that can be wired are depicted by continuous arrows.

others. Different options will be presented, justifying which is the best one among them (for the current application).

For this reason, a summary table, which classifies the different options qualitatively, is presented at the end of this section. The selected parameters of this table have been chosen due to their relevance in cloud computing and robotic technologies. They are: a) Best scalability, which indicates if the involved processing can be easily scaled out or back. b) Least communication bandwidth, as bandwidth is a precious resource in cloud computing and a source of timing delays. c) Least virtual computing resources. It indicates the amount of cloud resources used by an option. d) Cheapest cloud pricing, which must be considered when using public clouds, such as Amazon EC2. In addition to this, Section V will include experimental numerical results that quantifies which option is the best for the presented application.

Fig. 6 shows the most interesting ways of distributing the main computation tasks to be carried out in the navigation assisted teleoperation. These tasks are: the Frame buffering process ("Buffer" in Fig. 6), 3DPC extractors ("3DPC Ext." replicated in p nodes in Fig. 6), and the navigation assistant. For the current system, experiments in Section V show that the 3DPC extraction is more computationally complex than the navigation assistant, which basically includes the $d_{\text{coll}}(K)$

TABLE I
QUALITATIVE CLASSIFICATION OF DIFFERENT OFFLOADING MODELS. THE HIGHER THE NUMBER, THE WORSE THE OPTION IS REGARDING THAT PROPERTY

Property	Option 1	Option 2	Option 3	Option 4
Best scalability	1	3	2	4
Least communication bandwidth	2	3	4	1
Least virtual computing resources	3	1	2	-
Cheapest cloud pricing	2	1	3	-

function processing (see Section III). The necessary messages between these tasks are: 1) Color frames from camera to buffer, whose size is approximately proportional to image resolution; 2) Color frames from the buffer to the 3DPC extractors; 3) 3DPCs sent to the navigation assistant, which size in bytes ranges from 2 to 3 times that of raw color frames; and 4) Velocity commands that will be received by the actuation subsystem, which suppose just a few bytes.

For obtaining more reliable performance measures, every node moved to the cloud is isolated in one virtual machine. As explained in Section III, the different nodes of this solution have inherent distributed nature thanks to ROS middleware. Fig. 6 shows p 3DPC extractor nodes, because this processing part was designed to be dynamically scalable (see Section III).

Last option 4 is the classical robot centralized approach, but, when cloud offloading can be considered, other options are conceivable. As moving the camera to the cloud makes no sense (it must stay in the robot), and the buffer must reside in the same platform that the 3DPC extractors to prevent a waste of frame transfers, three additional options appear (see Fig. 6).

Option 1: 3DPC extraction and navigation assistance in the Cloud. This first configuration aims to move all the existing computation to the Cloud. A first advantage is that the robot has practically all its computing power available for other robotic tasks. Moreover, this configuration allows the exploitation of the cloud properties, due to the following fact: should the robot require more computing power, more 3DPC virtual machines can be spun up at runtime, and therefore the quality of the navigation will presumably increase (as it will be able to reach higher frequency rates or to process bigger frame resolutions). In terms of communication bandwidth usage, this configuration requires the transmission of stereo frame pairs (which size in bytes increases linearly with the frame resolution) to the cloud and reception of velocity commands (which size is commonly very short).

Option 2: Navigation assistance in the Cloud. This choice tries to better balance the computing tasks between local and cloud computation, but it is less scalable. Scaling out and back in the robot is far more limited than doing it in the Cloud. In terms of bandwidth usage, instead of sending frames to the cloud, 3DPCs will be sent (whose size ranges from 2 to 3 times that of frames'), and velocity commands (a few bytes) will be received. Therefore, the wireless bandwidth usage is bigger than the previous case, not being the case of virtual computing requirements. For a fully autonomous robot, images must not come out of the robot. However, for our current teleoperated application, video streaming must also be sent to the teleoperator, thus increasing bandwidth interferences and consumption. With respect to the current application, in terms of cloud pricing, this solution is cheaper than the previous one. However, because 3DPC extraction is more computationally complex than the navigation assistance, the robot will probably be unable to respond correctly when a better quality of the navigation is required (for example a bigger frame resolution, as delays in Section V exhibit).

Option 3: 3DPC extraction in the Cloud. In this case, the heaviest processing is moved to the cloud, whereas the navigation assistant node (which could be more reactive if the robot included another simple sensor) stays in the robot. This configuration seems the sensiblest in terms of typical real-time applications. In terms of scalability, it is possible to scale out and back 3DPC nodes depending on the needs. In terms of virtual computing power requirements, it is very similar to the first choice since the main core of computation resides in the 3DPC extraction. However, the main drawback of this configuration for the current application is the big amount of wireless communication that takes place: the robot sends stereo frame pairs and receives 3DPCs. As a consequence, if the communication technologies are not good enough, this solution can yield bad performance results. In addition to this, this solution may be the most expensive in terms of pricing due to the high bandwidth consumption.

Option 4: All processing in the robot. In this case, the only communication overheads will be internal to the robot node, which nowadays are usually inferior to those of remote communication. This choice does not use the cloud, hence it requires the robot's embedded hardware to be powerful enough. For the real application considered here, the next section demonstrates its poor performance results (even for Erratic Robot, which incorporates medium-end hardware). Just like option 2, the use of remote communication would be necessary if robot camera images were sent to the teleoperator.

Table I classifies all the choices according to all the properties analyzed. Taking into account all the advantages and disadvantages, it seems that options 1 and 3 are the most suitable for the described application. Section V proves that option 1 yields the best performance results.

V. EXPERIMENTAL RESULTS

This section presents the performance results for different experiments using the aforementioned example robotic application. The main objective is to prove which cloud computing options are viable for a real-time robotic system. In [28], it was proved that the scalability of the cloud works properly when changing the number of virtual cloud nodes. The system was tested with the hardest computing and communication processing: high definition resolution images (HD 1080i frames), the most consuming bandwidth model (option 3 in Fig. 6), but with the fastest available TCP network (Gigabit). It was showed that a significant speedup (ratio between total time for

1 node and for p nodes) can be obtained. This means that cloud elasticity makes it possible for the robot to change between different computing resources depending on the frequency and image resolution required by the system. Despite the fact that in some applications low-quality images and small updating frequencies could be enough, the stereoscopic formula for the depth error [32] says that this error is proportional to the real size of a pixel (that is, for a given CCD size, the more horizontal resolution, the less depth error exists. As an extension of this idea, it is evident that with perfect lighting and essentially infinite SNR, the highest accuracy is achieved using a combination of high framerate and high resolution, with limits only set by the available computational budget [33].

Section V-A analyzes the performance impact of current communication technologies, which has direct impact in the offloading. In that sense, Section V-B shows the application qualities when choosing the different offloading options presented in Section IV. Finally, Section V-C presents a real navigation experiment, both using the cloud and the robot on its own.

The cloud infrastructure used for the experiments is the following: a cluster consisting of five bare-metal machines (one front-end and four compute machines) with AMD Phenom 965 x4 CPU (with virtual extensions enabled) and 8 GB of RAM each one. They are all connected using Gigabit Ethernet bandwidth and *Openstack Havana* is the Infrastructure-as-a-Service (IaaS) cloud middleware installed. KVM has been chosen as the virtualization technology. Other well known solutions such as Hadoop were not suitable for real-time systems. The Erratic robot hardware has a 1.4 GHz Core 2 Duo Processor with 1 GB of RAM.

When moving computation to the cloud, the following mapping between nodes and virtual machines has been considered: each 3DPC extractor node is executed in a separate virtual machine with 2 Virtual CPUs (VCPU) and 2 GB of RAM (the front-end buffer VM has exactly the same properties). In the case of the navigation assistant node, a larger VM was chosen, consisting of 4 VCPUs and 8 GB of RAM (as the parallelization of function $d_{coll}(K)$ is obvious).

A. Analysis of Communication Technologies

This first experiment is done to compare how different communication technologies affect to the achievable processing rates. Moreover, using the cloud for mobile robot navigation, more realistic technology choices such as IEEE 802.11n WiFi and IEEE 802.11ac WiFi must be considered. The second one is quite recent and promises an expected bandwidth close to that of Gigabit Ethernet. In this experiment, the stereo camera and the image transmission has been carried out by a real mobile robot (in this case Videre Erratic). The cloud offloading configuration chosen is option 3, so this experiment checks if the bandwidth limitations are being a burden in the whole performance of the system.

Table II shows the frequency of the system for different technologies and different frame resolutions. In order to give a proper comparison, the table contains the results of option 4 as well (that is, the robot without cloud). Two conclusions can be extracted from these results. First of all, when using small resolution frames (for extracting 3DPCs), a boost of performance when using an external platform is not obtained with the communication technologies used here. The reasons are the following: the robot hardware is fast enough, and the networks used do not have enough bandwidth (for Infiniband or 10 Gbps Ethernet results might be better). This means that, for other simpler robots, offloading this process might be beneficial. Despite this, when the quality of the frames is increased, the robot hardware limitations arise. Therefore, the Cloud can be used not only for a simple computation offloading, but also for speeding it up.

TABLE II
PERFORMANCE MEASURES FOR DIFFERENT COMMUNICATION TECHNOLOGIES

Average frequency of 3DPC reception (Hz)

	320x240	640x480	1024x768	1920x1080
Gigabit	16.31	6.65	2.22	0.84
Wifi 11n	4.04	2.04	0.29	0.14
Wifi 11ac	4.98	3.02	0.76	0.24
Robot "Opt 4"	7.15	2.61	1.01	0.02

TABLE III
NAVIGATION ASSISTANT UPDATE FREQUENCIES FOR THE DIFFERENT OFFLOADING MODELS OF SECTION V USING 802.11AC WIFI

Frequency (Hz)

	320x240	640x480	1024x768	1920x1080
Option 1	12.55	5.72	3.73	1.06
Option 2	8.14	2.29	0.89	0.03
Option 3	7.48	2.16	0.96	0.25
Option 4	7.15	2.61	1.01	0.02

It must be pointed out that, when changing the hardware platform from that of the robot to a modern laptop, the frequencies obtained for Gigabit Ethernet differ from that of [28]. This fact is easily understood because both the CPU and RAM are four times better in the case of the laptop. Even though the robot has been freed from heavy computations, there are other factors that do affect in the whole performance of the system (peer to peer connection management, frame buffering and sending, 3DPC reception, along with others). The second fact discovered is that current wireless technologies are not enough to handle this process successfully due to the big amount of data transferred (not only the frames are transferred but also the 3DPCs). However, it must be noted that the offloading model chosen was not beneficial for limited communication technologies.

B. Comparison of Offloading Models

In order to compare all the offloading models of Section IV, two tests were carried out: the first one compares the frequencies in which $d_{\text{coll}}(K)$ is updated for different frame sizes, while the second one measures the latencies of the whole application.

Table III shows the results for different frame sizes and models using 802.11ac WiFi. The faster $d_{\text{coll}}(K)$ gets updated, the higher the quality of the navigation will be. Just as expected, the option 1, which has less bandwidth consumption, has the best results, even though option 3 seems a more common configuration for a real-time system. Only when the frame resolution is small, the required computation can be assumable by the robot (option 4) on its own. Due to the communication overheads, the speedup between options 1 and 4 becomes greater when the size of the frame increases, being worth having

TABLE IV
AVERAGE DELAYS OF THE SYSTEM FOR 1024 × 768 FRAMES AND DIFFERENT OFFLOADING MODELS USING 802.11AC WIFI

Mean Delays (s)						
	t_o	t_p	t_c	t_f	Total Comm.	Total
Option 1	0.199	0.382	0.154	0.026	0.353	0.761
Option 2	0.077	1.007	0.652	0.026	0.729	1.762
Option 3	0.511	0.382	1.223	0.064	1.734	2.180
Option 4	0.077	0.966	0.181	0.096	0.258	1.320

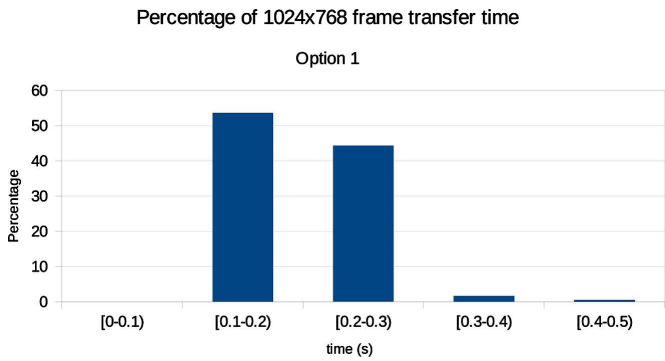

Fig. 7. Histogram of 1024 × 768 frames using option 1 and 802.11ac WiFi.

communication overheads in exchange of higher frequency of the whole system.

The second test measures the latencies of the whole application, that is, the difference between the time the stereo frame pair was taken and the time $d_{\text{coll}}(K)$ was calculated for them. Table IV shows that the main cause of delays in the system are again the communication overheads. Thus, an offloading model that minimizes them is benefited. The robot on its own is unable to get acceptable latencies; this time due to the heavy computation. With these two tests, it is certain that option 1 is the best offloading model for this case, obtaining decent results even with wireless technologies.

Finally, as communication latencies cannot be constant, it is interesting to show a histogram for t_o (in Fig. 7, 1024 × 768 stereo frame pairs were used for option 1 and Wifi AC). Although variance is not very big, some messages (only a 2%) present a high variability of the latency. This is due to four main factors: 1) the ROS middleware used; 2) complexity of 3DPC filters; 3) TCP protocols; and 4) interferences, and packet rejection with backoff periods in MAC layer. As percentage of late packet is very low, a simple predictive timing correction algorithm (see Section III) mitigates possible oscillations in the control loop. Besides, further improvements in *802.11ac* MAC layer like time-division multiple-access (TDMA) protocols, would reduce substantially this latency variance. An alternative to TDMA protocols is the use of the Contention Free Period in the infrastructure mode with fixed size packets. This could not reduce the variability as much as the use of TDMA, but it guarantees a minimum bandwidth reservation, which may be suitable for many real-time systems.

C. Analysis of a Real Navigation Case

Two final tests show that cloud-based robot navigation (option 1) is possible and even can improve its on-board counterpart. The vehicle has been configured with: $v_{\max} = 0,4$ m/s, $\omega_{\max\_\text{robot}} = 1$ rad/s, maximum linear acceleration $a_{\max} = 8m/s^2$, maximum linear brake $a_{\text{brake}} = 10$ m/s^2. Circuit of the first test contains obstacle free areas (where the robot is able to move at maximum speed) and four walls, where the robot should stop to prevent a collision. Figs. 8 and 9 demonstrate the viability of option 1. Fig. 8 shows $d_{\text{coll}}(K)$ when approaching one of these walls. Fig. 9 shows the difference between the joystick linear velocity and the final computed linear speed command when completing a turn of the circuit.

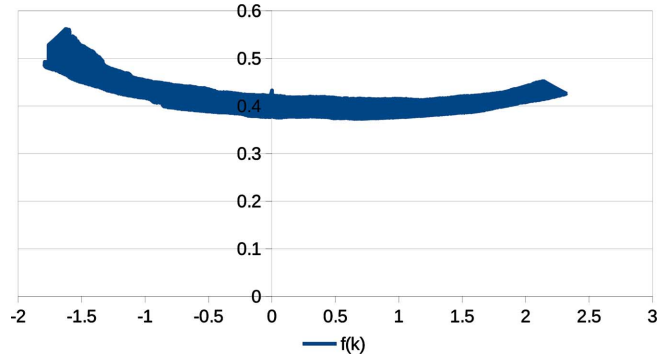

Fig. 8. Example of $s = d_{\text{coll}}(K)$ (in meters).

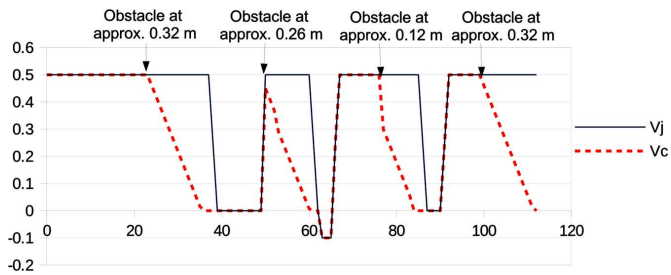

Fig. 9. Response of the navigation assistant to joystick commands.

TABLE V
NAVIGATION RESULTS FOR TEST 2 USING OPTIONS 1, 4, AND TWO FRAME RESOLUTIONS. THE RATIO IS CALCULATED OVER 70 MANEUVERS

Resolution	Option	# Collisions	Ratio
640x480	1	7	0.10
640x480	4	32	0.46
320x240	1	38	0.54
320x240	4	31	0.44

The second test compares the navigation of the robot in an office environment for options 1 and 4, and for low and medium resolutions (320 × 240 and 640 × 480 pixels) at 10 fps. The circuit is completed ten times per case. The number of collisions are measured as a quality magnitude, because, due to the delay variances in the navigation assistant computation, some collisions are difficult to avoid. For each obstacle approaching maneuver, the joystick is pushed to its extreme positions (desired commands are always those of maximum speeds), in order to test the shared control in the worst conditions. As there are seven of these maneuvers for each turn of the circuit, we have a total of 70 possible collisions. Table V shows the number and ratio of collisions for the four tested alternatives. The first conclusion is that for the stereo algorithm used here, low resolutions are not enough to detect properly some of the walls (when the robot is in movement), and collisions are very frequent for any computing option. The second important conclusion is obtained

for the 640×480 pixel resolution. Collision rate is very much reduced when the cloud is working, which is mainly explained because frame processing frequencies are more than double for option 1 than for 4 (see Table III). In this case, the only collisions are produced against an metallic furniture which texture is very much homogeneous. Another additional trouble is the limited FOV of the camera. This means that unavoidable collisions are produced when robot makes brusque turns (this must be solved by the inclusion of more cameras in the future). A demonstration video can be found in [34].

VI. Conclusion

This work analyzes and shows that running a vision based navigation assistant completely on an external cloud is feasible. Two main evidences allow this: a) cloud scalability is pretty well achieved and b) processing times run in parallel to the transmission ones, being the latter the bottleneck of the cloud offloading. In fact, the mean processing frequencies are almost proportional to bandwidth network. It is expected that the extension and natural evolution of wireless networks would improve their bandwidth in the next few years. Experiments demonstrate that the proper computation offloading model must be carefully chosen. In that sense, the computation versus communication tradeoff has to be analyzed. Moreover, the key to success in computation offloading is the scalability of the developed solutions. Finally, the implemented prototype (a teleoperated mobile robot using shared control) can be useful for the teleoperation of wheelchairs of other service robots, which can be done by the user or by an external caregiver. As a better navigation is obtained when using higher resolution images, the necessity of the cloud has been empirically demonstrated. It is also shown that the navigation assistant must contemplate a predictive temporal correction that should prevent a possible collision, caused by the sensoring delay, mainly due to the transmission latencies.

Acknowledgment

The authors wish to thank Prof. D. Cascado for his interesting comments. In addition to this, they really wish to thank the reviewers for all the feedback of our work, together with all the recommendations for future work.

References

[1] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The RoboEarth cloud engine," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2013, pp. 438–444.

[2] E. Guizzo, "Robots with their heads in the clouds," *IEEE Spectrum*, vol. 48, no. 3, pp. 16–18, Mar. 2011.

[3] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2010, pp. 3084–3089.

[4] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Autom. Sci. Eng. (T-ASE)*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[5] *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin, Germany: Springer-Verlag, 2008.

[6] J. Laird, *The Soar Cognitive Architecture*. Cambridge, MA, USA: MIT Press, 2012.

[7] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation sharing in distributed robotic systems: A case study on SLAM," *IEEE Trans. Autom. Sci. Eng. (T-ASE)*, vol. 12, no. 2, pp. 410–422, Apr. 2015.

[8] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *IEEE Robot. Autom. Mag.*, vol. 18, no. 2, pp. 69–82, Jun. 2011.

[9] J. Sevillano, D. Cascado, D. Cagigas, S. Vicente, C. Lujan, and F. del Rio, "A real-time wireless sensor network for wheelchair navigation," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. (AICCSA)*, May 2009, pp. 103–108.

[10] J. Fu, W. Hao, T. White, Y. Yan, M. Jones, and Y.-K. Jan, "Capturing and analyzing wheelchair maneuvering patterns with mobile cloud computing," in *Proc. 35th Annu. Int. Conf. IEEE Eng. Med. Bio. Soc. (EMBC)*, Jul. 2013, pp. 2419–2422.

[11] A. Civit-Balcells, F. Diaz Del Rio, G. Jimenez, J. Sevillano, C. Amaya, and S. Vicente, "SIRIUS: Improving the maneuverability of powered wheelchairs," in *Proc. Int. Conf. Control Appl.*, 2002, vol. 2, pp. 790–795.

[12] S. V. Diaz, C. A. Rodriguez, F. Diaz del Rio, A. C. Balcells, and D. C. Muniz, "TetraNauta: A intelligent wheelchair for users with very severe mobility restrictions," in *Proc. Int. Conf. Control Appl.*, 2002, vol. 2, pp. 778–783.

[13] P. I. Blasco, F. Diaz-del Rio, M. C. Romero-Ternero, D. Cagigas-Muiz, and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development," *Robot. Auton. Syst.*, vol. 60, no. 6, pp. 803–821, Jun. 2012.

[14] E. Charfi, L. Chaari, and L. Kamoun, "PHY/MAC enhancements and QoS mechanisms for very high throughput WLANs: A survey," *IEEE Commun. Surveys Tutorials*, vol. 15, no. 4, pp. 1714–1735, 2013.

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Waltham, MA, USA: Morgan Kaufmann, 2012.

[16] B. P. John, *Effectiveness of SPEC CPU2006 and Multimedia Applications on Intel's Single, Dual and Quad Core Processors*. Ann Arbor, MI, USA: ProQuest, 2009.

[17] M. Tenorth, A. Perzylo, R. Lafrenz, and M. Beetz, "The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2012, pp. 1284–1289.

[18] M. Tenorth, A. Perzylo, R. Lafrenz, and M. Beetz, "Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 3, pp. 643–651, Jul. 2013.

[19] M. Tenorth, K. Kamei, S. Satake, T. Miyashita, and N. Hagita, "Building knowledge-enabled cloud robotics applications using the ubiquitous network robot platform," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Nov. 2013, pp. 5716–5721.

[20] L. Wang, M. Liu, and M. Meng, "Real-time multi-sensor data retrieval for cloud robotic systems," *IEEE Trans. Autom. Sci. Eng. (T-ASE)*, vol. 12, no. 2, pp. 507–518, Apr. 2015.

[21] H. M. Do, C. J. Mouser, Y. Gu, W. Sheng, S. Honarvar, and T. Chen, "An open platform telepresence robot with natural human interface," in *Proc. IEEE 3rd Annu. Int. Conf. Cyber Technol. Autom., Control, Intell. Syst. (CYBER)*, Nanjing, China, May 2013, pp. 81–86.

[22] K. Ayush and N. K. Agarwal, "Real time visual SLAM using cloud computing," in *Proc. 4th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Tiruchengode, India, Jul. 2013, pp. 1–7.

[23] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robot. Auton. Syst.*, vol. 62, no. 4, pp. 401–413, Apr. 2014.

[24] H. Bistry and J. Zhang, "A cloud computing approach to complex robot vision tasks using smart camera systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Oct. 2010, pp. 3195–3200.

[25] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. S. G. Lee, "Real-time moving object recognition and tracking using computation offloading," in *Proc. EEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Oct. 2010, pp. 2449–2455.

[26] H. Wu, L. Lou, C.-C. Chen, S. Hirche, and K. Kuhnlenz, "Cloud-based networked visual servo control," *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 554–566, Feb. 2013.

[27] L. Agostinho, L. Olivi, G. Feliciano, F. Paolieri, D. Rodrigues, E. Cardozo, and E. Guimaraes, "A cloud computing environment for supporting networked robotics applications," in *Proc. IEEE 9th Int. Conf. Dependable, Auton. Secure Comput. (DASC)*, Dec. 2011, pp. 1110–1116.

[28] J. Salmerón-García, P. I. Blasco, F. Diaz-del Rio, and D. Cagigas-Muñiz, "Mobile robot motion planning based on cloud computing stereo vision processing," in *Proc. 41st Int. Symp. Robot. (ISR/Robotik)*, Jun. 2014, pp. 1–6.

[29] P. I. Blasco, F. Diaz-del Rio, S. Vicente-Diaz, and D. Cagigas-Muńiz, "The shared control dynamic window approach for non-holonomic semi-autonomous robots," in *Proc. 41st Int. Symp. Robot. (ISR/Robotik)*, Munich, Germany, Jun. 2014, pp. 1–6.
[30] P. Nisbet, "Who's intelligent? Wheelchair, driver or both?," in *Proc. Int. Conf. Control Appl.*, Sep. 2002, vol. 2, pp. 760–765.
[31] A. To, G. Paul, and D. Liu, "Surface-type classification using RGB-d," *IEEE Trans. Autom. Sci. Eng.*, to be published.
[32] M. Kytö;, M. Nuutinen, and O. Pirkko, "Method for measuring stereo camera depth accuracy based on stereoscopic vision," in *Proc. SPIE/ IS&T Electron. Imag.*, 2011, pp. 1–9.
[33] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?," in *Computer Vision ECCV 2012*, ser. Lecture Notes in Computer Science, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Germany: Springer-Verlag, Jan. 2012, pp. 222–235, no. 7578.
[34] "Cloud-based robot navigation assistant using stereo image processing," (accessed Nov. 12, 2014.) [Online]. Available: http://www.rtc.us.es/cloud-based-robot-navigation-assistant-using-stereo-image-processing/