Server Implementations for Improving the Performance of CAR Systems Based on Mobile Phones

Víctor Fernández, Juan Manuel Orduña, Pedro Morillo

Departamento de Informática, Universidad de Valencia, Valencia, Spain

Abstract

Collaborative Augmented Reality (CAR) systems allow multiple users to share a real world environment, including computer-generated images in real time. The hardware features of most current mobile phones include wireless network capabilities that offer a natural platform for CAR systems. However, the potential number of clients in CAR systems based on mobile phones is much larger than on CAR systems based on other kind of mobile devices, requiring a system design that takes into account scalability issues. This paper presents the experimental characterization of CAR systems based on mobile phones, providing quantitative results about well-known performance metrics in distributed systems. The results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server. Also, the results show that it is limited by the server I/O in some cases. Therefore, this paper also show different server implementations that can improve the system throughput, as well as a comparative study. The results show that the implementation based on UDP messages provides a significant improvement in system throughput with respect to other implementations based on TCP, at the cost of loosing a very small percentage of messages. These results validates this implementation as the most efficient one for large scale CAR applications based on mobile phones.

Keywords: Collaborative interaction, augmented reality, mobile phones

1. Introduction

Augmented Reality (AR) systems are nowadays widely used in applications such as medical procedures, scientific visualization, manufacturing au-

tomation, cultural heritage and military applications. The term Augmented Reality (AR) refers to computer graphic procedures or applications where the real-world view is superimposed by computer-generated objects in real-time [1, 2, 3]. From the beginning of AR systems, the potential of collaborative AR (CAR) systems was exploited for different activities such as Collaborative Computing [4] or Teleconferencing [5]. Wearable devices were used to provide CAR systems, where a wearable AR user could collaborate with a remote user at a desktop computer [6, 7].

On other hand, a lot of different mobile and/or wereable devices comprising a computing embedded system pervade our daily life, and they have been used for CAR systems. Mobile phones have become the most extended example of these devices [8, 9]. They have become an ideal platform for CAR systems, due to the multimedia hardware that they include, like full color displays, integrated cameras, fast processors and even dedicated 3D graphics chips [10].

As an example, Figure 1 shows a CAR system developed for collaborative training in industrial electricity. In this case, the CAR systems show the electrical technicians how the circuit-breakers should be replaced in the electric general panelboards at the construction sites. The figure shows on the left image the execution of the CAR tool on a Samsung Galaxy NOTE mobile phone. The image on the center shows a real image of the the panelboard where technicians collaboratively operate, and the right image shows the execution of the CAR tool on a HTC Nexus One mobile phone.

Figure 1: Example of a CAR application developed for training in industrial electricity.

Nevertheless, the wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the performance of a large-scale CAR system, in terms of

system latency, frames per second or number of supported clients with certain latency levels. These effects should be taken into account when implementing CAR systems based on mobile phones, in order to avoid a performance degradation in terms of both system latency and throughput.

In a previous work, we characterized the behavior of different mobile phones when used in Collaborative Augmented Reality applications [11]. The purpose of that work was to determine the performance that can be obtained in a CAR application when this kind of devices is used as client terminals. Also, we have carried out a preliminary performance characterization from the server side, measuring the system response time and system throughput when varying different systems parameters like the number of clients in the system, the number of clients in the work space (i.e., the number of neighbors to which messages should be sent), and the cycle time of clients [12]. Since the results showed that the CAR system throughput could be limited by the server I/O in some cases, we developed different server implementations [13].

In this paper we present, in a unified manner, the characterization of CAR systems based on mobile phones and the performance improvement that can be achieved by using different implementations of the system server. The characterization results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server, although it is not a fixed value and it is inversely related to the number of processor cores. Also, the results show that it heavily depends on the kind of client devices, and it is limited by the server I/O in some cases. Therefore, we have developed different server implementations that improve the system throughput. The comparison study shows that the implementation providing the best performance is the one based on UDP messages, providing a significant improvement in system throughput with respect to other implementations based on TCP and supporting more than one thousand clients at interactive rates (twice the number of supported clients of the TCP implementation). This improvement is achieved at the cost of loosing a very small percentage of updating messages. However, the effects of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the best option for large-scale CAR systems based on mobile phones.

The rest of the paper is organized as follows: Section 2 shows some related work about CAR applications on mobile phones. Section 3 describes the

characterization setup and the evaluation results provided by a CAR system using the kinds of clients considered in [14] with a system server implemented by means of TCP connections. Then, Section 4 describes different server implementations considered for improving the performance results shown in the previous section, and it presents a comparative study of the performance obtained with each server implementation. Finally, Section 5 presents some conclusion remarks.

2. Related Work

Augmented Reality superimposes multimedia content - 3D object, text, sound, etc - to real world through a display or screen. In order to locate digital contents on a specific image of the real world point, some references within the image are needed. These references are known as markers, and two methods are usually used for managing markers: natural feature tracking and fiducial marker tracking. The former method uses interest point detectors and matching schemes to associate 2D locations on the video with 3D locations [15]. This process can be grouped in three big phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database [16]. The latter method uses fiducial markers to find a specific position of the real world. This process can be divided in three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns [15].

Any CAR application needs a device equipped with an on-board camera, CPU and display. The most common devices used for CAR applications are Tablet-PCs or mobile phones. We will focus on mobile phones, because they are more suitable for CAR applications [10, 17].

There are few solutions based on fiducial marker tracking over mobile phones. In 2003, ArToolKit [18], one of the most well-known software libraries for developing Augmented Reality (AR) application, was released for Windows CE, and the first self-contained application was developed for mobile phones [19]. This software evolved later as the ArToolKitPlus tracking library [15]. A tracking solution for mobile phones that works with 3D color-coded marks was developed [9], and a version of ArToolKit for Symbian OS was developed, partially based on the ArToolKitPlus source code [10]. The research teams behind these works have worked on fiducial marker tracking, but not from the collaborative point of view. Also, there are many other works that focus on natural feature tracking [15, 20, 21, 22].

Although real-time natural feature tracking over mobile devices has been currently achieved [15], fiducial marker tracking is more widely used, because it allows simultaneous computational robustness and efficiency. A large number of locations and objects can be efficiently labeled by encoding unique identifiers on the markers. Additionally, the markers can be detected with angles near to 90 degrees [15].

The first CAR applications improved the conference system highlights, giving the feeling of real presence to remote collaborators [5]. The Rekimoto's Transvision system showed how to share virtual objects through handheld displays [23]. Also, Schmalstieg created a software architecture to develop CAR applications [24].

3. Server Characterization

3.1. Characterization Setup

In order to analyze the behavior of CAR systems based on mobile devices, we have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in our previous work [14]. We have time-stamped all the messages generated within this CAR system, in order to measure the performance of every device. The system configuration will consist of one server, and a certain amount of mobile devices that are scanning the visual space of their video camera looking for a marker that will be converted into a 3D object in their display. The main performance metrics in distributed systems are throughput and latency [25]. However, in order to avoid clock skews when measuring the system latency in distributed systems, the same device should measure the initial and final time. Therefore, we consider round-trip times instead of system latencies.

Since we are considering collaborative systems, after each updating of the object location, the mobile device will send a location update message (containing the new location) to each of its neighbor devices through the server (that is, it sends the location update message to the server, and then the server re-sends the message to the appropriate clients). We have denoted the set of neighbor devices of a given client as the working group, and we have defined this set based on a distance criterion (defined by the Area of Interest (AOI) size [26]), since all the clients working in the same task should be displaying the same real image, and therefore they must be separated each other by a short distance. The working group size (WGS) determines the number of update messages that each client device must send in each action cycle.

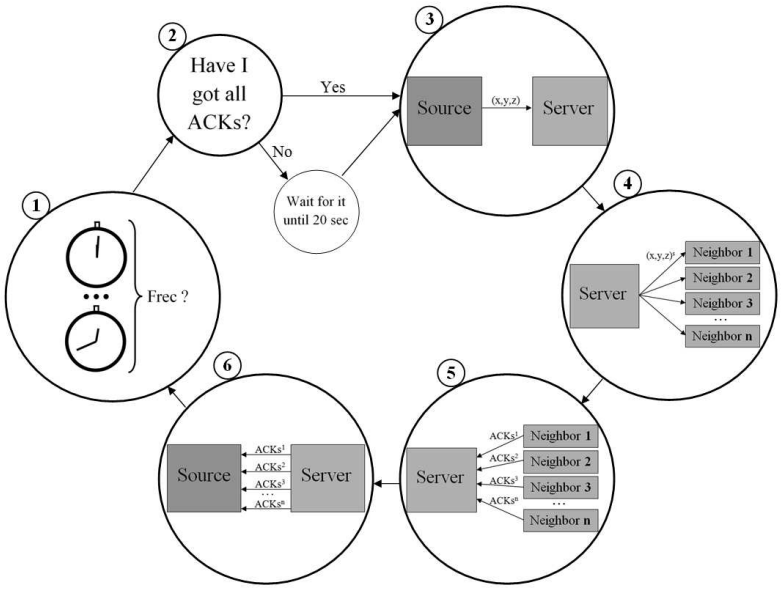

Figure 2: Stages of the action cycle in each mobile device

For characterization purposes, the destination clients return an acknowledgment message (ACK) to the server, which, in turn, forwards it to the source client. When the source client has received the ACK messages corresponding to the location update from all the clients in its working group, then it computes the average system response for that location update. Figure 2 illustrates the action cycle that takes place for each of the mobile clients in the system.

Once the message with the location update is sent, the action cycle performed by each client is composed of the following steps: first, it performs one new image acquisition followed by a marker detection stages. Then, the client waits until the cycle period (determined by the action frequency, a system parameter) finishes. Next, if the acknowledgments from all the neighbors have been received, a new message with the new marker location is sent. If not all the acknowledgments have been received, then it waits until a maximum threshold of 20 seconds, and then a new round of messages (with the latest marker location) are sent to the neighbors through the server. The neighbors simply returns an ACK message to the sender device through the server. The server simply forwards the messages to the corresponding des-

tination clients. It must be noticed that the mobile devices will not send a new round of messages with a new location update until it has received the acknowledgment message from all its neighbors, even although new marker detection stages have been completed in the device.

This characterization setup considers that all the required static content in the scene has been loaded. According to recent works [27], in these cases the network bandwidth required is less than 50 kbps for performing this information exchange. Since we are using a Gigabit Ethernet switch, we ensure that network bandwidth does not become a system bottleneck.

The system latency provided for each location update is computed by recording a timestamp when the first message is sent to the server. Next, a second timestamp is recorded with the last ACK message for that location update received from the server. The system response time is computed by subtracting these two timestamps. The server response time is computed by timestamping not only each message forwarded from each client, but also the reception of the corresponding ACK message from the destination client. Also, the percentage of CPU utilization is measured both in the server and the mobile devices every half second.

We have implemented a multithreaded server, where each server thread manages a working group of clients within a given AOI. Thus, for example, with a system configuration of 500 mobile clients and an working group size of 10 clients, we have 50 server threads, each of them supporting 10 clients. We have considered a maximum configuration of 1000 clients, resulting in 100 server threads. For system characterization purposes, we have considered a single server. Nevertheless, the system performance greatly depends on the server implementation. Therefore, in order to make a robust characterization of CAR systems, we have implemented two different versions of the server, denoted as *passive* and *active* server. The *active* server forwards each location update message to the corresponding neighbor clients, and it collects the ACK messages received from the neighbor clients. When all the ACK messages corresponding to a location update message have been received, then a single ACK message is sent to the corresponding source client. The *passive* server simply forwards each location update message to the corresponding neighbor clients and the ACK messages received from the neighbor clients to the source client. In this sense, neither computations nor data structures are needed in the server, although more messages are exchanged between the server and the clients (the network traffic increases).

The previous work [14] showed that Google phone HTC Nexus One was

the fastest device, with a period cycle of 167.11 milliseconds, while the Motorola Milestone was the slowest one, with a period cycle of 698.34 milliseconds. We have considered these values as the limits for characterization purposes. Also, we have considered four different values for the working group size: 5, 10, 20, and 25 neighbor clients. Finally, we have considered a number of clients in the system ranging from 100 to 1000. It must be noticed that usually, actual CAR applications do not contain more than a hundred clients (for example, more than a hundred persons within the same lounge using collaborative Augmented Reality for studying art masterpieces), due to the size of the augmented models. Thus, reaching thousands of clients clearly exceed the worst case for this kind of applications.

In this characterization, we have considered a TCP implementation of the server, where the simulator starts generating a *Server Process*, and for every 50 clients it generates a *Client Process*. Figure 3 illustrates the general scheme of the Server Process. This process starts listening connections, and for each connection it generates a new array of X TCP sockets, where X is the number of clients that will be within a given working group. When all the clients have connected to the Server Process (the population size is a simulation parameter) then the Server Process generates as many *Server Threads* as needed. Each Server Thread is in charge of managing all the clients within a working group. Concretely, it starts the simulation by sending a welcome message to all the client sockets. When the simulation finishes, it collects statistics from all the clients in its working group. But the most important task performed by server threads is the generation of two threads for each of the clients in the working group: the *Server Receiver Thread (SRT)* and the *Server Processor Thread (SPT)*. The SRT associated to client i receives the location update messages from the client i. Next, it computes the correct destination clients (the neighbor clients, that is, the clients within the same working group) and it generates messages that will be stored in the queues of the Server threads managing these neighbor clients. The SPT associated to client i extracts the queued messages that the SRTs associated to other clients may have generated for client i, and it sends them to this client. Additionally, the server process collects and processes the statistics generated by the server threads, and it also measures the percentage of CPU utilization.

Figure 4 illustrates the general scheme of the Client Process. This process generates 50 client threads (we have assumed a maximum population size of 1000 client devices), and it also computes the percentage of CPU utilization, client latencies, etc.. Each Client Thread generates two threads for each

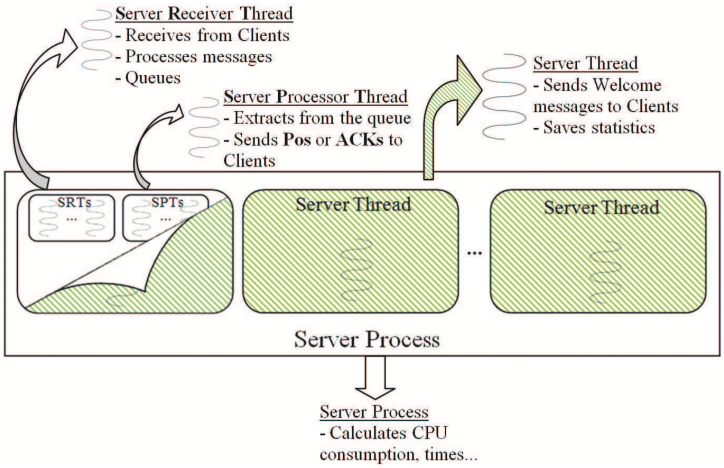

Figure 3: General scheme of the server process in the TCP implementation.

client: the *Client Receiver Thread (CRT)* and the *Client Processor Thread (CPT)*, and when the welcome message from the Server Thread arrives to the associated socket, then the Client Thread starts the simulation, which consists of sending a given number of position update messages and receiving the corresponding acknowledgments from the neighbor clients. The frequency of the location update messages is a simulation parameter (since it determines the actuation rate of clients to be simulated). In each simulation cycle (that can be much shorter than the client actuation cycle), the Client Thread checks if a new location update message should be sent. If so, it then checks if all the acknowledgments of the previous message have arrived from the neighbors clients. If not, then it waits for them until a maximum timeout of 20 seconds. The value for this timeout has been empirically obtained. Although it is not shown here for the sake of shortness, we have performed experiments with the simulator, concluding that unless the system reaches deep saturation and collapses, the maximum latency for obtaining all the ACKs from the neighbors have been 20 seconds. After this timeout, the new location update is sent. The CRT is continuously checking the client queue. When a location update message arrives to this queue, it sends back an acknowledgment to the corresponding server thread. If an acknowledgment of a previous message arrives to this queue, this acknowledgment is computed.

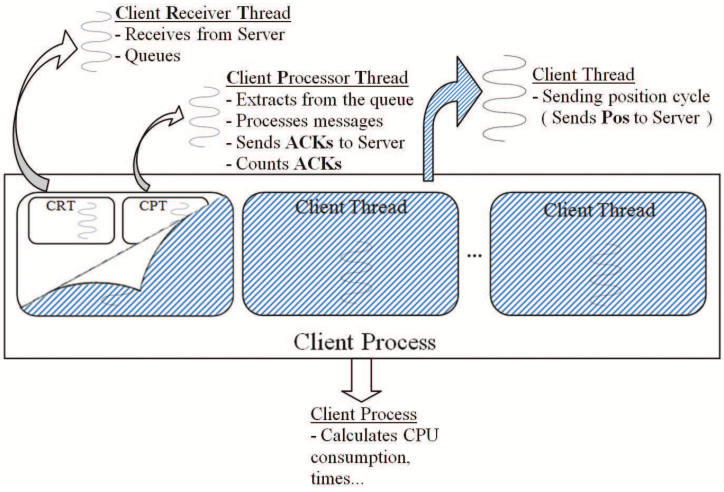

Figure 4: General scheme of the client process in the TCP implementation.

3.2. Server Evaluation

We have measured the average system response time for all the location updates sent by all the clients in the system. In this sense, we have considered the system response time (in milliseconds) for each update as the time required for receiving all the acknowledgments (for each message sent by a client device, an acknowledgment from each client in the same working group should be received). Also, we have measured the average response time in the server (in milliseconds) as the time required by the destination clients to answer the server messages. Additionally, we have computed the standard deviation for the response times, and the percentage of the CPU utilization in the system server, since it can easily become the system bottleneck. The computer platform hosting the system server is a Intel Core 2 Duo E8400 CPU running at 3.00 GHz with 4 Gbytes of RAM, executing an Ubuntu Linux distribution with the 3.0.0-14-generic x86_64 operating system kernel.

Table 1 shows the performance evaluation results for the active server implementation (denoted with the "a-" prefix in the tables) and considering a working group size of 10 neighbors for each client. The most-left column in this table is labeled as "WGS" (standing for "Working Group Size), and it shows the number of clients in the system and the number of neighbor clients in the working group. The values in this column ranges from 100 to 1000 clients in the system. Next, there are two groups of three columns, which shows the results for the fastest (HTC Nexus One, whose results are labeled as

Table 1: System performance for the active server implementation with an working group size of 10 neighbors

	a-One			a-Miles		
WGS 10	RT	Dev	CPU	RT	Dev	CPU
100	70.27	23.61	10.9	72.89	28.02	7.1
200	68.85	23.14	21.8	69.35	23.92	13.0
300	73.27	18.07	36.4	73.67	18.43	17.8
400	74.81	17.48	44.0	78.13	23.07	18.8
500	79.71	21.34	57.0	75.9	19.04	26.0
600	83.58	25.59	70.0	80.35	26.04	31.7
700	83.85	27.61	82.3	81.73	26.06	31.0
800	179.75	89.37	84.2	78.69	23.82	40.8
900	222.94	102.97	84.0	76.32	20.47	43.9
1000	255.67	116.19	85.2	78.29	21.99	45.1

"One") and the slowest client device considered (Motorola Milestone, whose results are labeled as "Miles"). The first two columns in each group of columns show the average system response time (in milliseconds) and its standard deviation. The third column shows the percentage of the CPU utilization in the system server.

Table 1 clearly shows different behavior and performance for different client devices. Thus, comparing the system response time (column RT) for the a-One (i.e., active server implementation and all clients using a Nexus One device) and a-Miles configurations, it can be seen that the RT slightly increases as the number of clients in the system increases in the case of the a-Miles, ranging from 72.89 milliseconds to 81.73 milliseconds. However, in the case of the a-One, when the population reaches 800 clients there is a huge increase in both the RT value and in its standard deviation with respect to the values for smaller population sizes. For higher population sizes, the increase in the RT values is also significant.

Table 2 shows the performance results for a CAR system with the same configuration except that the server implementation is now the passive server (denoted with a "p-" prefix). This table shows a similar behavior of the CAR system to the one shown in Table 1 for the active server. The only difference is the huge increase in the RT values of the "p-One" for a smaller population

Table 2: System performance for the passive server implementation with a working group size of 10 neighbors

	p-One			p-Miles		
WGS 10	RT	Dev	CPU	RT	Dev	CPU
100	78.23	30.41	16.0	77.03	27.63	9.9
200	74.02	25.06	25.2	78.16	26.07	26.0
300	79.45	24.57	37.0	83.6	28.61	25.0
400	81.02	24.05	59.6	82.54	23.76	23.8
500	93.61	28.56	68.7	88.27	26.76	31.3
600	147.73	100.48	83.8	87.66	27.45	33.0
700	195.47	76.24	83.2	88.45	28.58	33.7
800	237.71	91.09	85.9	94.43	34.05	62.0
900	270.25	104.97	84.2	84.8	25.7	69.3
1000	300.96	132.04	84.2	87.72	27.56	46.5

size (600 clients instead of 800 clients in the "a-One" configuration). These results show that the system reaches saturation for a population size of 800 clients in the case of a-One, and for a population size of 600 clients in the case of the p-One. However, in both cases the saturation starts when the percentage of CPU utilization in the system server reaches around 84% (83,8% in the passive server and 84,2% in the active server implementation). The configurations "a-Miles" and "p-Miles" do not reach saturation, since the RT remains almost constant as the population size increases. The reason for the different system behavior, when changing the devices, is the greater period cycle of the Motorola Milestone, which imposes a lower system workload since it generates new location updates at a much lower rate.

Regarding the acceptable limits for the RT values, some works consider 250 milliseconds as the response time limit for providing interactivity to human users [28]. If we consider such threshold value, then we can state that the throughput limit with the HTC Nexus One is 900 clients when using the active server, and 800 when using the passive server. That is, the system performance is surprisingly improved when the server reduces the traffic exchanged with the source clients (it collects all the ACK messages and only sends the source client a single ACK) at the cost of performing more computations (counting the ACK received for each update message).

Table 3: Server Response Times

	p-One				
WGS 10	RT	Dev	CPU	SRT	MaxSRT
100	78,23	30,41	16	18,2	20,15
200	74,02	25,06	25,2	18,5	24,9
300	79,45	24,57	37	25,78	28,62
400	81,02	24,05	59,6	27,71	30,4
500	93,61	28,56	68,7	31,83	40,5
600	147,73	100,48	83,8	40,93	44,37
700	195,47	76,24	83,2	49,85	52,39
800	237,71	91,09	85,9	58,08	62,23
900	270,25	104,97	84,2	64,79	68,9
1000	300,96	132,04	84,2	68,77	75,48

In order to find the reason for this behavior, we have measured the server response times for all the configurations considered. Table 3 shows the same results for the "p-one" configuration shown in Table 2, but now adding the corresponding average response times in the server (the average response time required by clients for sending back their ACK messages to the server, measured from the server) and their maximum values.

Table 3 shows that the average response times do not reach 70 milliseconds for the p-One device regardless of the population size, while the maximum values for the server response times do not reach 80 milliseconds. These values indicate that the system bottleneck (the reason for the significant average system response times shown in Tables 1 and 2) is located in the server. Since the comparison between these two tables shows that the active server achieves better performance than the passive server, these results suggest that the server I/O with the source client is the system bottleneck. It is important to notice that each source client in the passive server must not only receive all the ACKs for each message sent, but also it must send an ACK for every location update received from its neighbors. Although the active server adds more computational workload to the server for the same system workload than the passive server, it alleviates the server I/O with each client, and therefore the performance is improved.

In order to show that the behaviors shown in Tables 1 and 2 are consistent

Table 4: System performance for the active server implementation with a WGS of 20 neighbors

	a-One			a-Miles		
WGS 20	RT	Dev	CPU	RT	Dev	CPU
100	77.37	23.26	23.8	78.65	20.81	12.8
200	77.13	19.95	38.0	77.69	17.25	16.0
300	79.59	17.17	56.6	84.26	17.78	35.4
400	83.07	25.81	74.2	96.71	24.51	38.0
500	195.92	63.28	82.9	81.19	21.24	40.0
600	246.46	86.01	82.8	103.62	39.43	46.0
700	297.3	101.41	82.8	97.92	26.56	47.0
800	350.88	122.83	84.2	82.63	26.52	62.4
900	394.66	142.36	86.0	93.31	30.57	56.5
1000	410.14	181.39	87.0	88.52	26.54	66.3

for other workloads, Tables 4 and 5 show the results corresponding to a system configuration when all the clients have a working group size of 20 neighbor clients. These tables show behaviors that are very similar to the ones shown in Tables 1 and 2, except for the fact that the RT values are in general higher, and the system reaches saturation for lower population sizes. The results for the Milestone device show that the system does not reach saturation (there is not a significant jump in the RT values as the population size increases) in any of the tables. However, the results shown in Table 4 for the a-One show a jump in the RT values for a population size of 500 clients, when the percentage of CPU utilization is 82.9%. In the case of the p-One configuration (Table 5), the disruption occurs for a population size of 400 clients, when the percentage of CPU utilization is 84.2%. That is, again the system saturation point is a percentage of CPU utilization in the system server of around 84%.

In order to illustrate the behavior obtained for another workloads, we have measured the average system response times provided by the considered configurations for different WGSs. Concretely, Figure 5 shows the average system response times obtained for a working group size of 5 neighbors. Each plot in this figure correspond to a configuration of either the passive or the active server implementation, and using the HTC Nexus One or the Motorola

Table 5: System performance for the passive server implementation with a WGS of 20 neighbors

	p-One			p-Miles		
WGS 20	RT	Dev	CPU	RT	Dev	CPU
100	88.22	32.16	27.6	93.88	26.9	20.2
200	94.57	29.06	45.5	94.98	25.74	21.2
300	128.16	28.88	73.3	98.01	24.79	29.7
400	195.92	63.42	84.2	129.24	30.8	37.7
500	273.58	89.08	84.0	101.96	30.91	43.0
600	326.79	106.94	86.2	121.53	39.5	50.5
700	400.45	142.76	87.0	148.84	66.52	83.9
800	447.02	155.46	87.0	110.41	39.07	65.3
900	473.99	223.98	86.0	101.4	32.33	67.3
1000	467.04	286.28	86.9	106.66	37.8	74.7

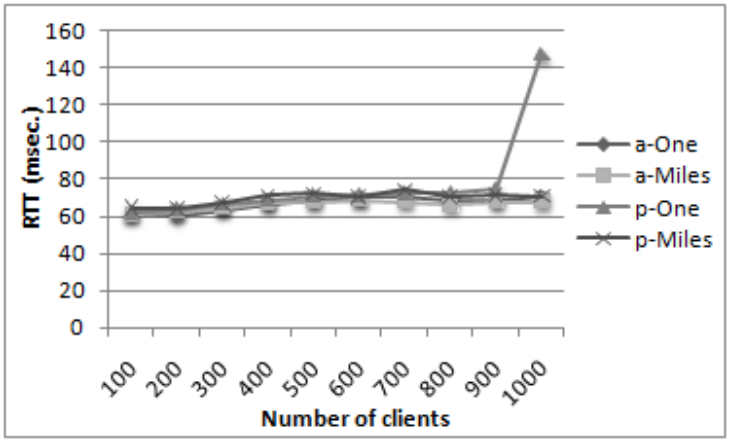

Figure 5: Average system response times for a WGS of 5 neighbors

Milestone as client devices. The X-axis shows the number of clients in the system, while the Y-axis shows the average system response times in the simulation.

Figure 5 shows that the only configuration that approaches saturation is the "p-One" (passive server with Nexus One devices as clients). Effectively, this plot significantly increases for 1000 clients, although the average response

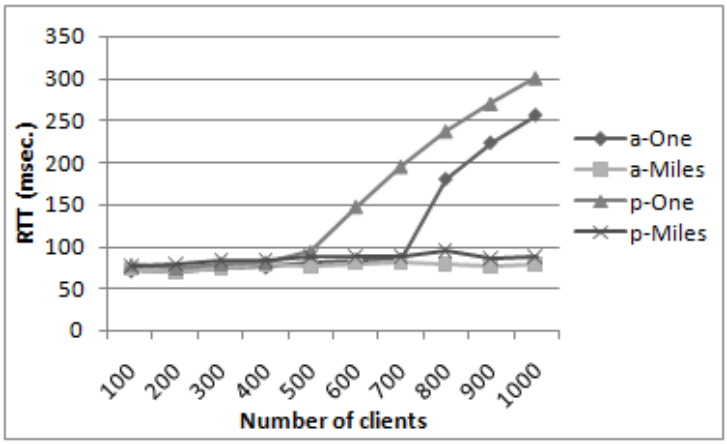

Figure 6: Average system response times for a WGS of 10 neighbors

time is far away from the threshold of 250 milliseconds.

Figure 6 shows the average system response times obtained for a working group size of 10 neighbors, that is, it graphically shows the results shown in the column labeled as "RT" in Tables 1 and2. This figure shows that in this case the two plots corresponding to the simulations performed with Nexus One clients reaches saturation, although the plot for passive server implementation starts to increase with fewer clients (600 instead of 800) and reaches higher response times than the active server implementation (300 milliseconds instead of 255). For 1000 clients, both plots exceed the limit of 250 milliseconds.

Figure 7 shows the same results for the case of a WGS of 20 neighbor clients. This figure shows how the system throughput decreases as the working group size increases. Concretely, this figure shows that, again, the plots for the Motorola Milestone have a flat slope, while plots for the HTC Nexus One start to significantly increase for population sizes of 400 clients (p-One) and 500 clients (a-One). In this case, the passive server implementation shows a throughput (the number of supported clients without exceeding the threshold value of 250 millisecond for the average system response) of 400 clients, while the throughput for the active server is 600 clients. Also, the maximum system response times obtained with this working group size exceed 450 milliseconds for the p-One implementation.

Finally, Figure 8 shows the same results for the case of a working group size of 25 neighbor clients. The behavior is very similar, except that the

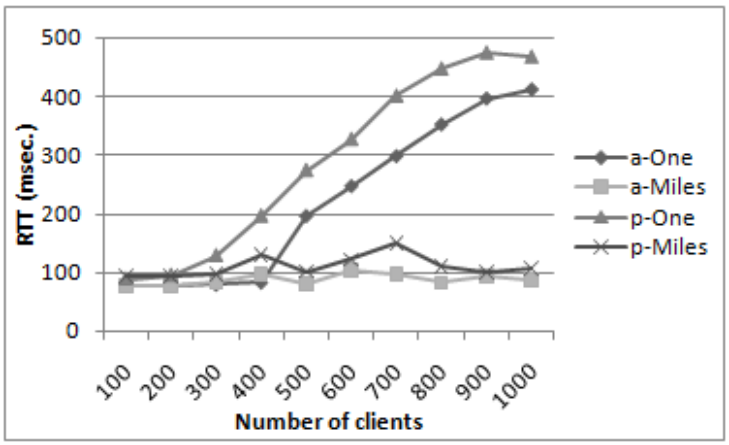

Figure 7: Average system response times for a WGS size of 20 neighbors

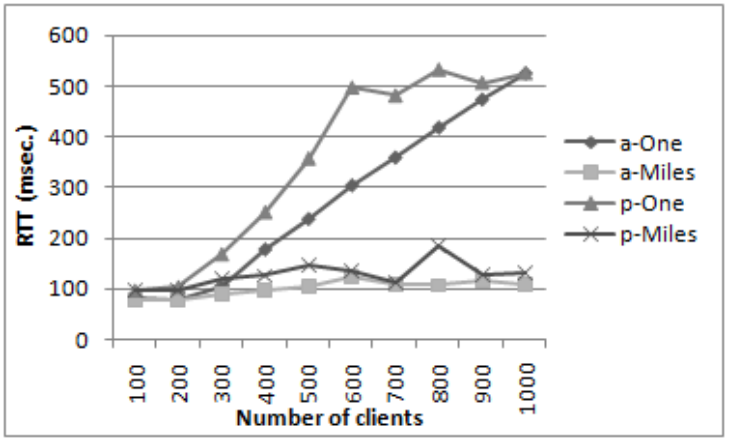

Figure 8: Average system response times for a WGS size of 25 neighbors

plots starts to increase with smaller population sizes (300 and 400 for the passive and active server implementations, respectively). For this working group size, the maximum response times obtained exceed 500 milliseconds for a population size of 1000 clients.

When comparing these figures among them, it can be seen that the configurations based on the Milestone device does not reach saturation in any figure. Also, the system throughput achieved by the p-One plots are the lowest ones in all the figures, while the response times shown in these plots are the longest ones. Additionally, (as it could be expected) the system through-

put decreases as the working group size (that is, the workload generated by the same number of client devices) increases, supporting less devices without reaching saturation. These results confirm that the server I/O becomes the system bottleneck in this CAR configuration; although the active server adds more computational workload to the server for the same system workload than the passive system server, it alleviates the workload of the server I/O, and therefore the performance is improved.

Finally, we have studied the saturation point of the CAR configurations considered. Although they are not shown here for the sake of shortness, we have obtained a percentage of CPU utilization in the server around 84% as the system saturation point for all the configurations shown in Figures 5, 6, 7 and 8. However, similar distributed systems like Distributed Virtual Environments (DVEs) were reported to reach saturation when any of the server reaches around 95-98% of CPU utilization [26]. The difference between 85% and 98% of CPU utilization for reaching the saturation point can be explained by the shared memory architecture of current multicore processors (the dual core processor in the computer platform used as simulation server). The synchronization of the kernel calls, together with the synchronization among threads in the application, prevent the CAR system from fully exploiting the computational power of all the processing cores at the same time, reaching saturation for a lower overall percentage of CPU utilization. The performance characterization of DVEs was performed on single core processors, and therefore those systems reached around 98% before reaching saturation. In order to show that this is the reason for this behavior, we have performed the same tests not only with the computer platform described above (dual core processor), but also with a different multicore computer platform, consisting of an Intel(R) Core(TM) i7 960 CPU (eight cores) running at 3.20GHz with 8 Gbytes of RAM and executing a SuSE Linux 2.6.37.6-0.5-desktop operating system. A representative example of the results obtained in such tests is shown in Table 6, which shows the results for the p-One configuration with a working group size of 25 neighbor clients.

Table 6 shows that the average system response time hugely increases when the population size is composed of 500 clients for the case of the Intel i7 platform, passing from 98.65 milliseconds to 228.98 milliseconds, while the standard deviation for this parameter also double its value. That is, the system saturation point for this server platform is a population size of 500 clients, and for this saturation point the overall percentage of CPU utilization for this execution is 42.46% (fourth most-left column in table 6). However,

Table 6: Performance evaluation for two different computer platforms

	8 cores (i7)			2 cores (Core Duo)		
WGS 25	RT	Dev	CPU	RT	Dev	CPU
100	90.91	26.91	24.1	96.03	25.91	60.4
300	98.65	20.63	33.6	167.58	50.19	86.1
500	228.98	42.36	43.5	357.13	126.19	92
700	357.95	75.79	45.3	480.87	156.16	85.3
900	478.95	103.34	48.3	505.89	300.11	89.0

the results obtained with an Intel Dual Core processor show that the system reaches saturation with 300 clients, where the average system response time jumps from 96.03 milliseconds to 167.58 milliseconds (this result is also shown in the p-One plot in figure 8). For this system response time, the overall percentage of CPU utilization is 86.1%. That is, as it could be expected, the system throughput is increased when a more powerful processor is used as the system server, but the saturation point of the CAR system is not a fixed value for the overall percentage of the CPU utilization, and it depends on the number of processor cores. Although they are not here for the sake of shortness, the system reaches the saturation point when the CPU utilization is around 43% for all the configurations tested with this platform.

4. Improving Server Performance

The characterization shown in the previous section (the one shown in [12]) considers a server using a TCP implementation. In that implementation, the server is based on TCP sockets, one for each agent in the simulation. Thus, one of the potential limitations of the TCP implementation is the server overhead due to the huge number of threads, for those cases when the population size increases. This overhead could be reduced by using the `Select` function of BSD sockets [29]. Moreover, the TCP connection overhead could be fully avoided if UDP sockets were used. In this section, we show the performance of these two variation in the implementation of the server.

4.1. TCP-Select Implementation

In this server implementation, each Server Thread has a single SRT and a single SPT for managing all the clients in each working group, instead of

one SRT and one SPT for each client. Using the `Select` function, the SRT receives messages from all the clients and it processes them. We have tested different options regarding the best number of SPTs for managing all the clients in each working group. Although they are not shown here for the sake of shortness, the experimental results showed that no significant improvements were achieved when using more than a single SPT for managing, in terms of the obtained latencies.

4.2. UDP Implementation

We have also considered a connectionless oriented implementation for the CAR system, in order to study the effectiveness of UDP connections in a distributed environment like a CAR system. Although the UDP protocol can loose messages and the effects and size of these losses should be studied, usually connectionless oriented protocols like UDP show a better network performance for frequent but short network traffic [25].

The UDP implementation is very similar to the TCP-Select implementation, in order to void the overhead of having as many sockets as clients in the simulation. The only difference is that in this implementation we have used UDP sockets. Since this implementation can drop messages, it also counts the number of dropped or lost messages (since both the number of iterations and the number of clients in each working group is known, each client can compute the number of message that should arrive).

Figure 9 shows the main differences between the TCP implementation and the TCP-Select and UDP implementations. These differences are in the Server Threads (STs) that are executed in the server. Each server thread in the TCP implementation generates one SRT and one SPT for each client in the workgroup size. Each SRT establishes a new socket with the corresponding client, having therefore as many connection sockets as clients in the simulation. In both the TCP-Select and the UDP implementations, there is a single SRT for each working group, an therefore a single socket. All the clients within a given working group establish a connection with the same server socket in both the TCP-Select and the UDP implementations. The only difference between the TCP-Select and the UDP implementations is that the latter one uses UDP sockets instead of TCP sockets.

4.3. Performance Comparison

We have performed different measurements on different simulated systems using the considered server implementations. Again, we have performed sim-

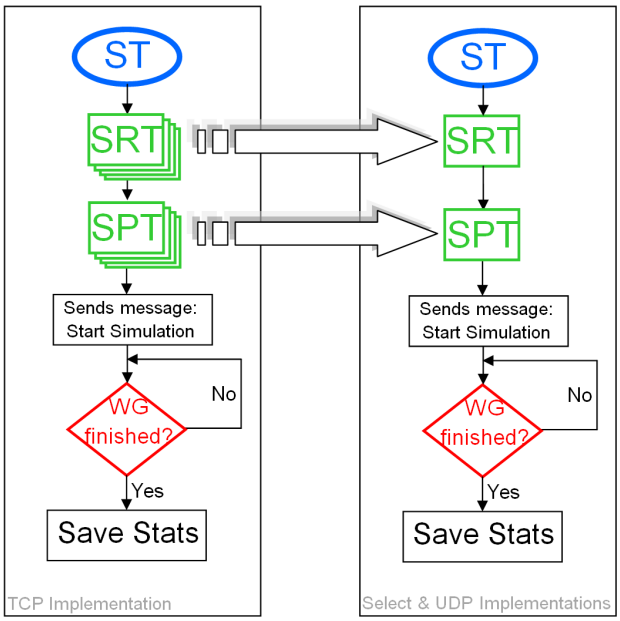

Figure 9: Differences among the different server implementations.

ulations with different number of clients and we have measured the response time provided to these clients (the round-trip delay for each updating message sent by a given client to the clients in its working group). In this way, we can study the maximum number of clients that the system can support while providing a response time below the threshold value of 250 ms. [28].

We have considered the system response time (in milliseconds) for each updating message sent by a given client to its neighbor clients as the time required for receiving the acknowledgments from all the clients in the working group of this given client. In order to measure the dispersion of this metric, we have measured the standard deviation for all the updating messages sent, as well. Additionally, we have computed the percentage of the CPU utilization in the system server, since it can easily become the system bottleneck. The computer platform hosting the system server is a Intel Core 2 Duo E8400 CPU running at 3.00 GHz with 4 Gbytes of RAM, executing an Ubuntu Linux distribution with the 3.0.0-14-generic x86 64 operating system kernel.

In order to study the system behavior for different levels of workload, we have repeated simulations with working group sizes of 10,15,20 and 25 clients. Although not all of them are shown here for the sake of shortness,

we show the result for the smallest (5 clients in each working group) and the biggest sizes (25 clients in each working group).

Table 7 shows the results for a CAR system whose client devices are all of them HTC Nexus One, and where the working group size contains five neighbor clients. This table shows the results for the three considered implementations, organized as three subtables with ten rows each, and labeled with the name of the implementation (TCP, TCP-Select and UDP). The most-left column in these subtables shows the number of clients in the system, that is, the population size. The values in this column range from 100 to 1000 clients in the system. The next two columns show the average value of the response times (in milliseconds) provided by the system to all the clients (labeled as "RT"), as well as the corresponding standard deviation values (column labeled as "Dev"). The fourth column (labeled as "CPU") shows the percentage of the CPU utilization in the server. milliseconds) of the response time in the server for all the messages exchanged during the simulation.

	TCP			SELECT			UDP		
Size	RT	Dev	CPU	RT	Dev	CPU	RT	Dev	CPU
100	62.37	22.68	9.9	65.77	21.56	8	4.80	7.06	38.4
200	63.77	22.21	15	67.12	22.71	11.2	3.76	4.95	34.7
300	66.71	22.66	22	67.52	22.6	19.8	9.57	9.74	26.0
400	68.68	22.5	32.7	67.64	22.88	28	3.60	5.18	33.7
500	71.04	23.56	45	69.12	23.23	31	4.59	6.41	41.6
600	71.5	24.18	48.6	69.14	23	39.6	7.34	13.17	47.0
700	72.37	25.01	59	69.37	23.45	47	5.28	8.24	53.0
800	72.85	26.01	68	75.75	26.63	54.5	7.10	18.52	84.1
900	75.01	28.98	79.2	70.24	24.81	59.6	5.85	11.69	66.3
1000	147.33	101.71	85	71.05	27.53	67	7.15	15.47	69.5

Table 7: Results for a working group size of 5 neighbors

Table 7 shows that none of the values in the RT column reaches the threshold value of 250 milliseconds in any of the considered implementations, showing that the system can efficiently support up to one thousand clients while interactively displaying the Augmented Reality content. Nevertheless, there are significant differences in this column among the considered implementations. Thus, the TCP implementation shows a huge rise in the response time when the system reaches one thousand clients, passing from around 75 milliseconds to more than 147 milliseconds as an average. The

standard deviation of these values are also more than three times the value shown for nine hundred clients. These values show that for that population size the system is approaching saturation. On the contrary, the SELECT implementation does not show an increase in neither the column RT nor the column "Dev" for a population of one thousand clients. Moreover, the UDP implementation shows RT values that are one order of magnitude lower than the ones shown by the other two implementations.

The fourth column in Table 7 shows that the CPU utilization increases as the number of clients in the system increases. In the case of the TCP implementation, the system approaches saturation when the server reaches 85% of CPU utilization. For lower percentages of CPU utilization the response times do not significantly increase. It is worth mention that the UDP implementation provides RT values that are one order of magnitude lower than the ones provided by the TCP implementations, even for CPU utilization of around 70%. These values show that the latency provided by CAR systems greatly depends on the connection or connectionless scheme followed by the system to exchange information with the clients.

These results show that the best latencies when the system is far from saturation are provided with the UDP implementation. However, UDP is a connectionless-oriented protocol, and therefore it may drop messages when the system approach saturation. In order to study these effects, as well as the general behavior of the system for a high workload, Table 8 shows the results for a working group size of 25 clients. In this table, we have added a new column for the case of the UDP implementation, labeled as "% dr.", that shows the percentage of messages dropped by the system when it approaches saturation. It has been computed by subtracting the number of messages received by all the clients in a simulation (measured in the simulation itself) from the theoretical number of messages that clients should exchange for a given population size.

Table 8 shows that for this level of workload the system enters saturation in the two TCP-based implementations. Effectively, the RT column shows that the TCP implementation reaches the maximum threshold value of 250 ms. with 400 clients. From that population size up, the response time provided to clients is unacceptable. The SELECT implementation reaches (and exceeds) this threshold value for a population of 500 clients. However, the UDP implementation does not reach even half of this value for the maximum population size considered, one thousand clients.

The "% dr." column shows that for the UDP implementation the per-

	TCP			SELECT			UDP			
Size	RT	Dev	CPU	RT	Dev	CPU	RT	Dev	CPU	% dr.
100	96.0	25.9	60.4	90.8	24.7	23.2	9.9	6.8	72.5	0.83
200	103.2	38.7	83.0	89.9	21.1	47.0	21.7	14.7	82.0	1.18
300	167.6	50.2	86.1	123.9	32.4	72.0	26.0	21.9	79.6	0.69
400	250.5	77.7	85.0	209.2	35.9	87.2	39.4	30.7	81.9	0.83
500	357.1	126.2	92.0	268.2	44.4	86.0	48.7	39.7	83.8	0.74
600	496.6	218.8	94.1	331.4	50.6	87.0	62.8	45.1	83.8	0.74
700	480.9	156.2	85.3	383.9	70.6	93.1	79.70	97.8	85.1	0.76
800	531.9	210.0	86.0	454.6	125.2	90.1	87.2	200.9	84.0	1.04
900	505.9	300.1	89.0	491.6	106.2	91	93.6	66.4	83.9	0.93
1000	524.9	372.6	87.1	566.4	133.3	93.1	122.4	85.4	85.0	0.90

Table 8: Results for a working group size of 25 neighbors

centage of lost messages is not higher than 1.2%. The effects of loosing some messages will consist of delaying the location update of the artificial artifacts in the display of the destination clients, that is, they will produce some jitter in the display of the clients. However, these percentage values ensure a reasonable quality in the visualization of the CAR system. Therefore, we can state that the UDP implementation provides the highest throughput and the best latency for large scale CAR applications, at the cost of a very limited effects on the visualization in some clients.

In order to ensure that the effects of the UDP implementation in terms of dropped messages are consistent for all the workload levels considered, Figure 10 shows the average number of packets dropped for each working group size considered.

Figure 10 shows that for working group sizes of 5 and 10 neighbor clients there are no packet losses. For a working group size of 20 neighbors, the amount of lost packets reaches 8581 for a theoretical total number of packets sent of 1.9 million packets. Analogously, for a working group size of 25 neighbors, the amount of lost packets reaches 21593 out of 2.4 million packets sent. Therefore, in the worst case the number of lost packets only represent a 1'18 % of the total amount of packets sent. This value represents only a small image flicker on some clients, and in very limited periods of time. Since the information is sent more than once per second (since the action cycle of the HTC Nexus One is 167.11 ms.), this value can be considered an insignificant flickering.

Although they are not here for the shake of shortness, we repeated the

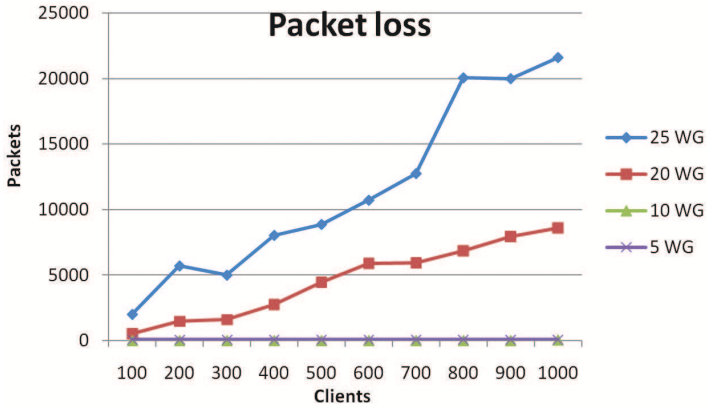

Figure 10: Number of packets dropped in the UDP implementation.

same tests shown in this section here using a different client device, the Motorola Milestone, and we obtained analogous results. Those results were less interesting because of the bigger action cycle of the Milestone (698.34 ms.). For that action frequency, the system saturation point was not reached even in the worst case of a working group size of 25 neighbors and a population of 1000 clients. We have shown here the results for the Nexus One as the worst case for the server implementation.

5. Conclusions and Future Work

In this paper, we have proposed the experimental characterization of CAR systems based on mobile phones. The results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server, although it is not a fixed value and it is inversely related to the number of processor cores. Also, the results show that it heavily depends on the kind of client devices, and it is limited by the server I/O in some cases. Therefore, we have also developed different server implementations that improve the system throughput and we have performed a comparison study. The results show that the implementation based on UDP messages provides a significant improvement in system throughput with respect to other implementations based on TCP, supporting more than one thousand clients at interactive rates (twice the number of supported clients of the TCP implementation). This improvement is achieved at the cost of loosing a very small percentage of updating messages. However, the effects

of these small quantities of dropped messages cannot expand beyond some jitter (bounded within a short period of time) in a reduced number of clients of the CAR application. These results validate the proposed UDP-based implementation as the most efficient approach for the design of large scale CAR applications based on mobile phones.

As a future work to be done, we plan to repeat the whole characterization and improvement work for the case of natural feature tracking [30, 31], since this kind of libraries has widely extended as a common tool for developing AR applications. Also, we plan to study the use of mobile's Graphics Processor Units (GPU), since it is a common hardware feature in current mobile phones.

References

[1] R. Azuma, A survey of augmented reality, Presence: Teleoperators and Virtual Environments 6 (1997) 355–385.

[2] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, B. MacIntyre, Recent advances in augmented reality, Computer Graphics and Applications, IEEE 21 (2001) 34 –47.

[3] S. Cawood, M. Fiala, Augmented Reality: A Practical Guide, Pragmatic Bookshelf, 2008.

[4] M. Billinghurst, I. Poupyrev, H. Kato, R. May, Mixing realities in shared space: an augmented reality interface for collaborative computing, in: IEEE International Conference on Multimedia and Expo (ICME 2000), volume 3, 2000, pp. 1641–1644. doi:10.1109/ICME.2000.871085.

[5] M. Billinghurst, H. Kato, Real world teleconferencing, in: Proc. of the conference on Human Factors in Computing Systems (CHI 99), 1999.

[6] T. Hallerer, S. Feiner, T. Terauchi, G. Rashid, D. Hallaway, Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system, Computers and Graphics 23 (1999) 779–785.

[7] W. Piekarski, B. H. Thomas, Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality, 2002.

[8] A. Henrysson, M. Ollila, Umar: Ubiquitous mobile augmented reality, in: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia, MUM '04, ACM, New York, NY, USA, 2004, pp. 41–45.

[9] M. Mahring, C. Lessig, O. Bimber, Video see-through ar on consumer cell-phones., in: ISMAR'04, 2004, pp. 252–253.

[10] A. Henrysson, M. Billinghurst, M. Ollila, Face to face collaborative ar on mobile phones, in: Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on, 2005, pp. 80 – 89.

[11] V. Fernández, J. M. Orduña, P. Morillo, How mobile phones perform in collaborative augmented reality (car) applications?, The Journal of Supercomputing 65 (2013) 1179–1191.

[12] V. Fernández, J. M. Orduña, P. Morillo, On the characterization of car systems based on mobile computing, in: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, 2012, pp. 1205 –1210. doi:10.1109/HPCC.2012.177.

[13] V. F. Bauset, J. M. Orduña, P. Morillo, On the implementation of servers for large scale car systems based on mobile phones, in: International Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, GRAPP 2013, 2013.

[14] V. F. Bauset, J. M. Orduña, P. Morillo, Performance characterization on mobile phones for collaborative augmented reality (car) applications, in: Proceedings of the 2011 IEEE/ACM 15th DS-RT, DS-RT '11, 2011, pp. 52–53.

[15] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, D. Schmalstieg, Pose tracking from natural features on mobile phones, in: Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 125–134.

[16] S. E. Lee, Y. Zhang, Z. Fang, S. Srinivasan, R. Iyer, D. Newell, Accelerating mobile augmented reality on a handheld platform, in: Computer

Design, 2009. ICCD 2009. IEEE International Conference on, 2009, pp. 419 –426. doi:10.1109/ICCD.2009.5413123.

[17] M. H. M. B. B. Thomas, Emerging Technologies of Augmented Reality: Interfaces and Design, IGI Global, 2007. doi:10.4018/978-1-59904-066-0.

[18] D. H. Kato, Artoolkit, 2011. Available at http://www.hitl.washington.edu/artoolkit/.

[19] D. Wagner, D. Schmalstieg, First steps towards handheld augmented reality, in: Proceedings of the 7th IEEE International Symposium on Wearable Computers, ISWC '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 127–135.

[20] S. Srinivasan, Z. Fang, R. Iyer, S. Zhang, M. Espig, D. Newell, D. Cermak, Y. Wu, I. Kozintsev, H. Haussecker, Performance characterization and optimization of mobile augmented reality on handheld platforms, in: Workload Characterization. IISWC 2009. IEEE International Symposium on, 2009, pp. 128 –137. doi:10.1109/IISWC.2009.5306788.

[21] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, D. Schmalstieg, Real-time detection and tracking for augmented reality on mobile phones, Visualization and Computer Graphics, IEEE Transactions on 16 (2010) 355 –368.

[22] D. Wagner, D. Schmalstieg, H. Bischof, Multiple target detection and tracking with guaranteed framerates on mobile phones, in: Proceedings of ISMAR 2009, 2009, pp. 57 –64. doi:10.1109/ISMAR.2009.5336497.

[23] J. Rekimoto, Transvision: A hand-held augmented reality system for collaborative design, in: Virtual Systems and Multi-Media (VSMM)'96, 1996.

[24] Z. Szalavri, D. Schmalstieg, A. Fuhrmann, M. Gervautz, studierstube: An environment for collaboration in augmented reality, Virtual Reality 3 (1998) 37–48.

[25] J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks: An Engineering Approach, IEEE Computer Society Press, 1997.

[26] P. Morillo, J. M. O. na, M. Fernández, J. Duato, Improving the performance of distributed virtual environment systems, IEEE Transactions on Parallel and Distributed Systems 16 (2005) 637–649.

[27] T. Kantonen, Augmented Collaboration in Mixed Environments, Master's thesis, Helsinky University of Technology, 2009.

[28] T. Henderson, S. Bhatti, Networked games: a qos-sensitive application for qos-insensitive users?, in: Proceedings of the ACM SIGCOMM 2003, ACM Press / ACM SIGCOMM, 2003, pp. 141–147.

[29] M. T. Jones, BSD Sockets Programming from a Multi-Language Perspective, Charles River Media, Inc., Rockland, MA, USA, 2003.

[30] D. Wagner, I. Barakonyi, I. Siklossy, J. Wright, R. Ashok, S. Diaz, B. MacIntyre, D. Schmalstieg, Building your vision with qualcomm's mobile augmented reality (ar) platform: Ar on mobile devices, in: Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 1–. URL: http://dx.doi.org/10.1109/ISMAR.2011.6092355. doi:10.1109/ISMAR.2011.6092355.

[31] Z. Balint, B. Kiss, B. Magyari, K. Simon, Augmented reality and image recognition based framework for treasure hunt games, in: Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on, 2012, pp. 147 –152. doi:10.1109/SISY.2012.6339504.