DOI: http://dx.doi.org/10.1109/EESMS.2013.6661705

An Android-based Platform for Augmented-Reality Remote Inspection Systems Prototyping

Stefano Ferrari and Gheorghi B. V. Pentchev
Department of Computer Science
Università degli Studi di Milano, Milan, Italy
Email: stefano.ferrari@unimi.it

Abstract—**A framework for developing prototypes of remote controlled inspection robots with semi-autonomous behavior and augmented reality enriched real-time video feedback is presented. This platform is constituted of very common and reusable hardware, equipped with open source software. The resulting system is very flexible and customizable, with a low set-up time, which provide an affordable fast prototyping framework.**

I. Introduction

Remote visual inspection systems allow to provide monitoring capabilities in dangerous or inaccessible environment [1][2][3]. In their basic set-up, these systems are composed of a robotic unit which carries one TV camera to capture a video stream of the explored environment. This video is then sent to the human operator which drives the remote unit. More complex systems can carry other sensors, have actuators for interacting with the environment (e.g., to repair or rescue), and have semi-autonomous capabilities.

The design of such systems requires to consider several issues, encompassing the environmental operational conditions, the activity the robot should realize (with a degree of autonomy), the interface for the human operator, which usually results in contrasting specification constraints. Hence, during the development of such a systems, several design choices have to be considered, which can imply the realization of several prototypes, in order to experiment the compliance of the solution to the specification constraints, and to start testing the usability of the system. Realizing a prototype and a replica of the working environment [4][5] can be expensive, both economically and in terms of time.

In this paper, a framework for developing prototypes of remote controlled inspection robots is presented. This platform is constituted of very common and reusable hardware, equipped with open source software. The resulting system is very flexible and customizable, with a low set-up time, which provide an affordable fast prototyping framework. Besides, the real-time video stream captured by the on-board camera can be enriched by Augmented Reality (AR) contents [6].

AR is commonly used for a plethora of applications, ranging from didactics (e.g., in museums or archaeological sites [7]), to maintenance and repair [8]. It has been proven that enriching the visual feed-back can improve the precision in remote operation [9]. In the present work, a simple implementation of AR is used. It makes use of fixed geometrical patterns, an approach often used in literature (e.g., magic cards [10]), which allow to easily mark special points of interest in the experimental environment, in order to emulate obstacles or dangerous objects.

The paper is organized as follows: in Section II, the overall system architecture is introduced, in Section III the functionalities of the main modules are presented, and in Section IV some implementation choices are discussed.

II. The system architecture

The system is structured in three main modules (Fig. 1):

- the robotic module,
- the on-board unit,
- the controller.

The robotic module is composed of the actuators, the basic sensors, and a low-level processing unit. It is responsible for the low-level decisions, which can be triggered by the on-board sensors, and realizes the semi-autonomous behavior. The robotic module carries the on-board processing unit, which is responsible to capture the scene and provides some processing in order to produce and stream the video with the augmented reality enriched scene. It also communicates with the robotic module in order to collect the sensor readings and to transmit the commands for the actuators. The controller provides the user I/O interface. It receives the video streaming from the on-board unit and displays it, and collects the user input and pass it to the on-board unit.

To realize this architecture, we chosen to implement the system using two Android devices for the on-board unit and the controller, while the robotic module has been implemented using the Lego® Mindstorm® NXT set. The communication between the controller and the on-board unit is realized via Wi-Fi, while the communication between the on-board unit and the robotic module is realized through Bluetooth® (BT) technology, as described in Fig. 1.

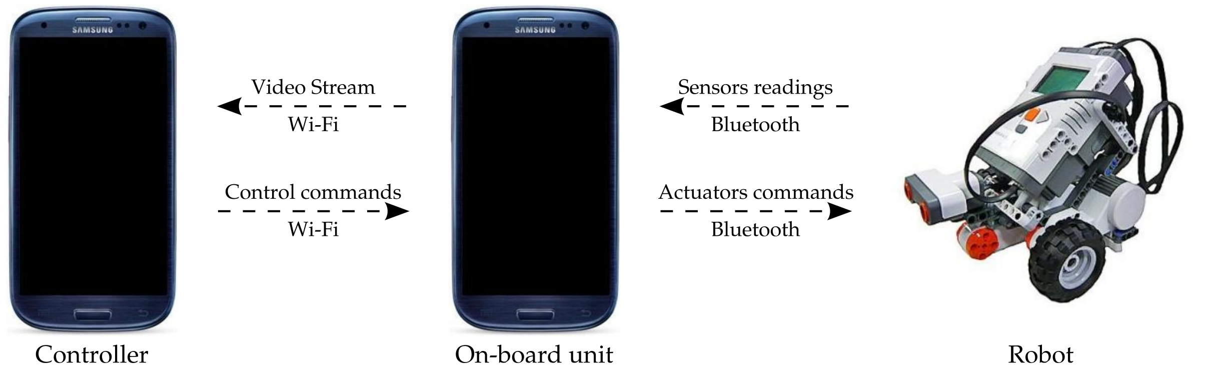

Fig. 1. High-level view of the system architecture.

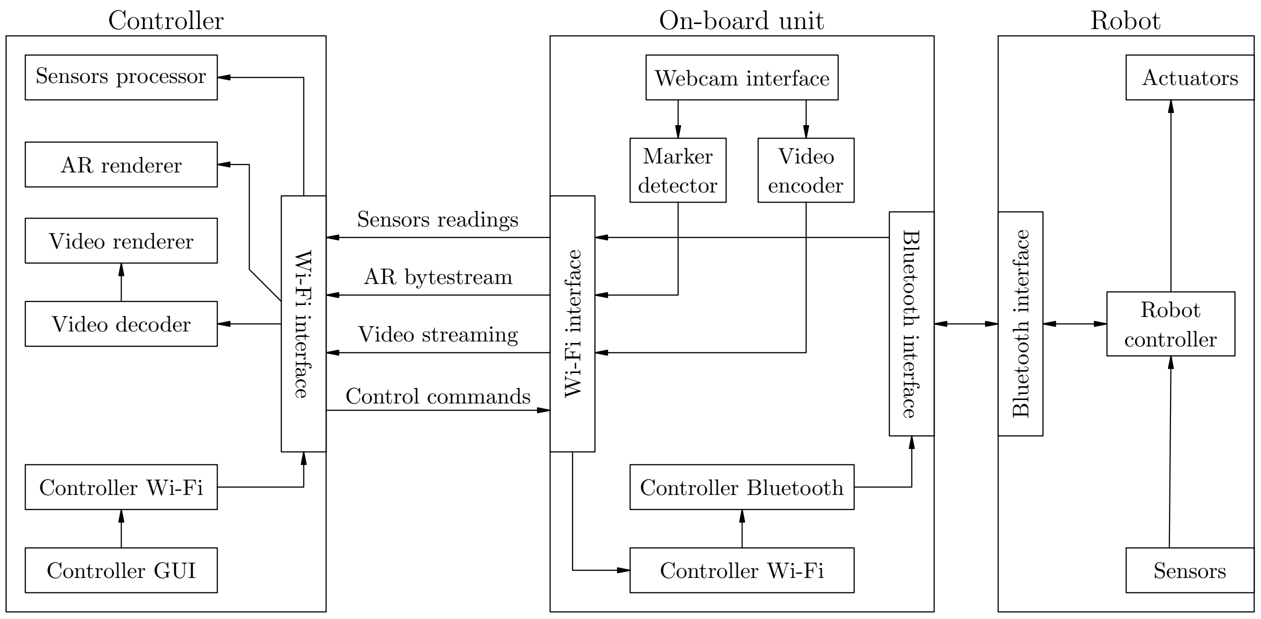

Fig. 2. The architecture of the application.

A. Android Operating System

The Android Operating System is structured in three main layers, as represented in Fig. 3. The Android OS is built over a Linux kernel (3.x, from Android 4.0) which provide the device drivers for interacting with the hardware components and the typical OS functionalities, such as memory, process, file, and power management, and networking.

Over this layer there is the Android native libraries layer and the Android Runtime. The libraries are written in C/C++, but can be accessed by a Java interface. They provide the basic functionalities for the applications (such as building the graphical interfaces, access to the web, record and play a media content). The Android runtime is composed of the Dalvik Virtual Machine (a Java Virtual Machine optimized for embedded systems) and the Core libraries (Core Java APIs which provide a basic development platform).

Next up there is the Application Framework layer, which provides the building blocks for the Android applications. In particular, at this layer level, there are some tools for recording multimedia data (mainly from microphone and camera) and playing the recorded contribution. These functions are provided mainly by two components of the multimedia framework, MediaRecorder and MediaPlayer, which provide respectively the recording and the playing. These components can operate on both audio and video data, performing coding and decoding using the most common multimedia formats. However, the multimedia application framework has not been suited for real-time streaming until its more recent versions. In fact, since buffering several frames is required to optimize the encoding of the stream, real-time streaming can be provided using these libraries only forcing the release of short blocks of frames, e.g., by periodically starting and stopping the coding and the transmission of the encoded multimedia data. Besides being very rigid, this way to operate is also inefficient, due to the overhead required for starting the interaction with the capturing devices.

Since the version 4.1 of Android (Jellybean, API 16, released in July 2012), a new set of powerful low-level media APIs is now available: the MediaCodec library. It provides to

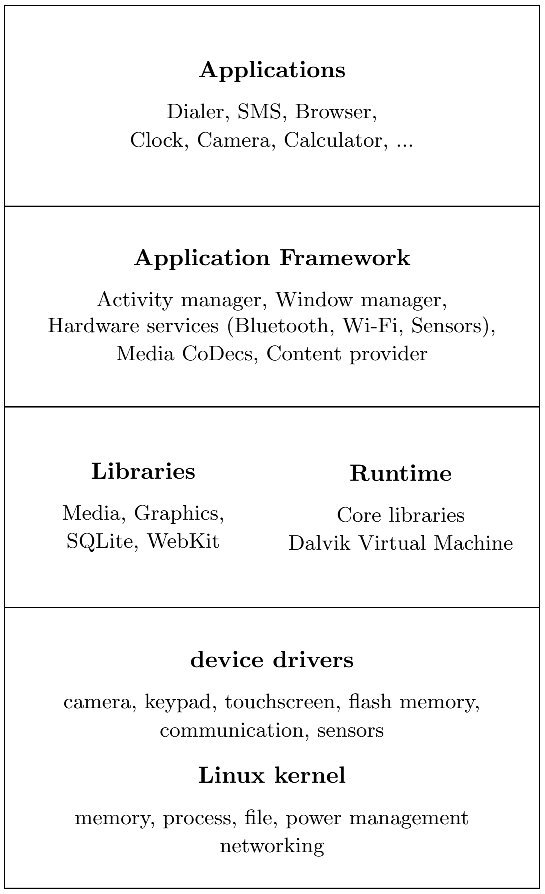
Applications
Dialer, SMS, Browser, Clock, Camera, Calculator, ...
Application Framework
Activity manager, Window manager, Hardware services (Bluetooth, Wi-Fi, Sensors), Media CoDecs, Content provider
Libraries
Media, Graphics, SQLite, WebKit
Runtime
Core libraries
Dalvik Virtual Machine
device drivers
camera, keypad, touchscreen, flash memory, communication, sensors
Linux kernel
memory, process, file, power management networking

Fig. 3. The architecture of the Android Operating System.

the applications the possibility of accessing to the hardware codecs directly from Java. This library allows to set-up the parameters for the video encoding (e.g., height, width, color space, frame rate). However, since different devices can have different hardware, the only guaranteed color space is the YUV with a 4:2:2 chrominance subsampling schema, which has been used in this work.

B. The software layers

Several open source AR frameworks are available for Android. Among them, at least AndAR [11] and NyARToolkit [12] should be cited. AndAR is a Java wrapper for ARToolKit [13], which is one of the most known open source AR toolkit and is coded in C. Since the Android applications are executed on a Java virtual machine, the ARToolKit library cannot be run directly, but the Java wrapping of AndAR, ARToolKit based applications can be run on Android. However, the simplicity of use of the ARToolKit functionalities provided by AndAR imposes a certain rigidity in the framework: the acquisition, the AR enrichment and the visualization have to be operated on the same device. This is a strong limitation when the application requires that the processing is distributed on several devices which have to operate in real-time.

NyARToolkit for Android is a project maintained by the Japan Android Users Group [14]. Since it is a porting in Java of the ARToolKit's main functionalities, it allows the development of AR applications on the Android platform.

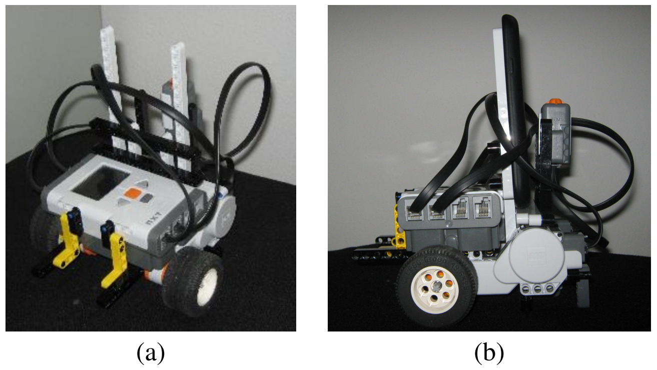

Fig. 4. The robot module. Its main scope is to provide mobility to the on-board unit, but it can also realize some basic data collection and low-level autonomous decision. It is composed of a processing unit, two motors configured as a differential drive, and some sensors (color and touch) (panel (a)). In panel (b), it is shown carrying the on-board unit.

Besides, it decomposes the AR processing chain in primitive operations, giving to the developer the flexibility to use them to build the application fulfilling the specifications in more complex scenario than just a single application on a single device. The NyARToolkit is available also for other languages (Java, C++, C#), which can be useful for extending the here presented application on devices based on other software platforms.

III. The application framework

A. The robot

The robotic module (Fig. 4) has been realized using a Lego Mindstorm NXT 2.0 set. It is composed of an intelligent brick (a programmable micro-controller at which can be connected up to four sensors and three actuators), two motors, and two sensors (one touch and one color sensor). In order to provide mobility, the two motors has been configured as a differential drive, due to its simplicity. In this schema, two wheels are attached to two motors, independently; the robot trajectory results from the rotational speed difference of the two motors. The color sensor provides a RGB reading of the ground under the robot, while the touch sensor can detect collisions with the environment and other objects.

The information collected by the sensors can be be passed to the user controller through the on-board unit, but can also processed by the intelligent brick for taking some autonomous decision. For instance, when a collision is detected the robot can stop itself, avoiding potential damages, and waiting for the suitable recovering instructions that the user can send to the robot.

Several firmwares are available for the intelligent brick; they provides the basic functionalities of an operating system, such as allowing the execution of programs and managing the I/O. Programs can be written in several languages (e.g., the Lego firmware supports NXC, a simplified C, while LeJOS [15] supports Java) and run in a multitasking environment. In particular, the Lego firmware provides also a Bluetooth communication protocol that allows some basic interaction with the connected peripherals: sensor readings can be obtained and command to the actuators can be forwarded by sending (and receiving) suitable messages on a Bluetooth channel. Hence,

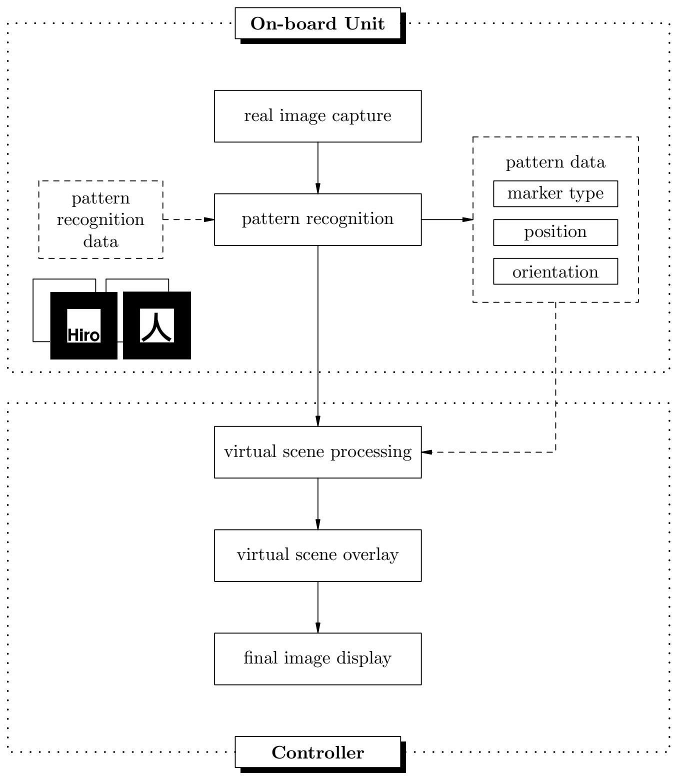

Fig. 5. The processing scheme for obtaining an augmented reality scene. In the application here considered, the first two steps are provided by the on-board unit, while the last three steps are accomplished by the controller.

for simple applicative contexts, where semi-autonomous behavior in not required, there is no need of running programs on the intelligent brick, and all the commands can come from the user through the on-board unit. However, for implementing a more complex behavior, programs that run on the intelligent brick can take care of auxiliary processing and communicate in a more proactive fashion with the on-board unit and the user.

B. *The on-board unit*

The main scope of the robotic module is to provide mobility to the on-board unit, which has the task of capturing the scene and sending it to the controller. It has been implemented using an Android 4.1 smartphone equipped with a camera.

During the start up phase, three channels are opened toward the controller: one for the video streaming, one for the AR information, and one (optional) for the sensors readings. One channel, instead is opened from the controller to the on-board unit for sending the control commands.

Once the application get access to the camera, the video encoding starts. For this task, the codec H.264 [16] has been chosen, since it guarantees a high compression ratio, which results in a better quality of the transmitted video stream: compared with other codec, H.264 can perform with a lower bit-rate. This is an advantage in situations when the network capacity is reduced, due to the distance or the environmental topology [17]. The encoding is set on a frame-rate of 20 frame-per-second (fps).

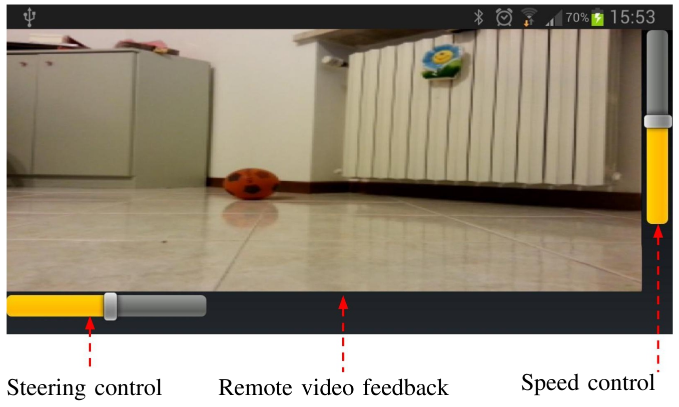

Fig. 6. A screenshot of the controller interface.

The second main task of the on-board unit is the processing for AR (Fig. 5). This is realized through the NyARToolkit framework: each frame captured by the camera is converted in gray-scale, and then in binary format through thresholding. Then, for each marker previously registered in the AR subsystem, the pattern recognition procedure searches for the presence of the marker in the scene, and, when found, output its position and orientation with respect to the coordinate system of the frame. These are the data required for adding the virtual object to the scene for the visualization. In the case of the present application, this task is realized on the controller module, and hence these information are bytestreamed to that device. It worth noting that the encoding of the frame and the AR processing are executed by two different process, and hence, depending on the hardware equipment of the on-board device, these two tasks can be accomplished concurrently.

C. *The controller*

The controller module is the interface between the remote module of the system and the user (Fig. 6). It collects and displays the information of sensors and camera (after enriching it with the AR information) and forward the commands for moving the robotic platform.

As above described, the controller receives from the on-board unit both the video stream captured by the on-board camera and the AR information. The video stream is processed by the H.264 decoder, which decode the frames and render them in RGB format on an OpenGL texture. The AR information, instead are used to render the virtual object corresponding to the recognized marker in the position and with the orientation previously computed. The rendered model is then composed with the real scene frame applying an overlay layer to the same OpenGL surface used for displaying the real scene. Also in this case, the two main tasks of the visualization of the scene can be processed concurrently.

The control of the motion of the remote module is realized through two sliders: one (horizontal) for steering, and one (vertical) for controlling the speed. The value of these bars is captured by the controller GUI module (Fig. 6), which passes these values to another module (the “Controller Wi-Fi” in Fig. 2) that is responsible for translating these values to commands that can be processed by the robot controller to drive the actuators. In fact, depending on the architecture of the propulsion system, different commands have to be sent to the robot.

For the present work, since we we chosen the simple differential drive, the steering is caused by the different rotation of the wheels, which are directly connected to the motors, and hence depends on the power supplied to each motor. The power supplied is computed composing the value of the speed slider (which provide the power reference value) and that of steering (which further modulates the power reference value). In practice, given $s \in [-1, 1]$ and $d \in [-1, 1]$ the position of the speed and the direction sliders, respectively, the power of the right and left motors, p_r and p_l, respectively, are computed as:

$$p_r = \begin{cases} s\,(1 - 2\,d), & d > 0 \\ s, & d \leq 0 \end{cases} \tag{1}$$

$$p_l = \begin{cases} s\,(1 + 2\,d), & d < 0 \\ s, & d \geq 0 \end{cases} \tag{2}$$

The computed p_r and p_l values are in $[-1, 1]$ and can be easily remapped in the $[-P_{\text{max}}, -P_{\text{min}}] \cup [P_{\text{min}}, P_{\text{max}}]$, where P_{min} and P_{max} represents the minimum and the maximum values of the power the motors can be set.

IV. Discussion

Some design choices can be questioned, the main of which is the use of marker-based AR: although it can be useful for other entertainment related applications, does it worth for a remote inspection? In an operative scenario, since the environment is supposed being artificial, some objects or building details (e.g., signals) can be easily recognized by lightweight procedure and their position can be visually enhanced for a better perception of the remote operator. Besides, in case of exploration of a building already mapped, some additional information can be added to the operator's display. The AR however, can be useful also in the developing phases. For instance, the characteristics of a hostile environment can be virtually reproduced, and their effects can be simulated on the operation of the robotic module (e.g., a marker can represent a puddle and if the robot travel across it, some loss in gripping can be simulated by adding a certain drift to the robot's motion).

From the implementation point of view, the choice of deploying the AR recognition subsystem on the on-board unit can be discussed, since it burden the computational load of this unit. Typically, in fact, this device would be smaller and, since it is battery powered, has limited computational power. From a logical point of view, this processing could instead be performed on the controller module, which can have less constraints both in term of weight and portability. Hence, since it does not have any a-priori constraint about the power source, it could provide a higher computational performance. However, the video encoding process can introduce artifacts which can interfere with the pattern recognition process.

In fact, the video stream is partitioned in small chunks of successive frames, called groups of pictures (GOPs). The first frame of each GOP is completely encoded, while the others are encoded as difference with respect to the previous one (in particular, they are approximated as a composition of blocks of pixels of the previous frame, suitably shifted). Hence, encoding errors can be propagated until the next GOP. Although this approximation can be acceptable by the human observer, the artifacts introduced can have disruptive effects on the binary pattern recognition, while performing this operation directly on the frames captured by the on-board camera allows to avoid this problem. The alternative, which consist in transmitting the video stream without any lossy compression (option allowed by the MediaCodec library), is unfeasible due to the saturation of the communication channel, which would force a lower resolution in time (i.e., a lower frame rate) and hence would provide a jerky video stream. Besides, this can involve a delay in the recognition of the marker or also a potential miss in the marker recognition due to a frame drop.

The performance of the AR streaming system depends mainly on the device used for implementing the on-board unit. We used a Samsung Galaxy Young Duos GT-S6312, with a single-core Cortex A5 1 GHz processor and GPU Adreno 200, Android 4.1.2 version, kernel 3.4.0, with 768 MB RAM and a 3 megapixels camera. With this hardware, the estimated visualization delay has been 0.42 s.

V. Conclusion

In this paper, a framework for developing prototypes of remote controlled inspection robots with semi-autonomous behavior and augmented reality enriched real-time video feedback has been presented.

It requires very common and reusable hardware, equipped with only open source software.

These properties and its modular design allow to obtain a very flexible system, with a low set-up time, which provide a fast prototyping framework which can be also used a didactic tool.

References

[1] H. Roman, B. Pellegrino, and W. Sigrist, "Pipe crawling inspection robots: an overview," *Energy Conversion, IEEE Transactions on*, vol. 8, no. 3, pp. 576–583, 1993.

[2] J.-B. Izard, L. Gargiulo, D. Keller, and Y. Perrot, "Hardening inspection devices to ultra-high vacuum, temperature and high magnetic field," *Applied Superconductivity, IEEE Transactions on*, vol. 20, no. 3, pp. 1767–1772, 2010.

[3] M. Friedrich, G. Dobie, C. C. Chan, S. Pierce, W. Galbraith, S. Marshall, and G. Hayward, "Miniature mobile sensor platforms for condition monitoring of structures," *Sensors Journal, IEEE*, vol. 9, no. 11, pp. 1439–1448, 2009.

[4] O. Netland and A. Skavhaug, "Prototyping and evaluation of a telerobot for remote inspection of offshore wind farms," in *Applied Robotics for the Power Industry (CARPI), 2012 2nd International Conference on*, 2012, pp. 187–192.

[5] M. Bengel, K. Pfeiffer, B. Graf, A. Bubeck, and A. Verl, "Mobile robots for offshore inspection and manipulation," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 3317–3322.

[6] M. Gervautz and D. Schmalstieg, "Anywhere interfaces using handheld augmented reality," *Computer*, vol. 45, no. 7, pp. 26–31, 2012.

[7] V. Vlahakis, N. Ioannidis, J. Karigiannis, M. Tsotros, M. Gounaris, D. Stricker, T. Gleue, P. Daehne, and L. Almeida, "Archeoguide: an augmented reality guide for archaeological sites," *Computer Graphics and Applications, IEEE*, vol. 22, no. 5, pp. 52–60, 2002.

[8] S. Henderson and S. Feiner, “Exploring the benefits of augmented reality documentation for maintenance and repair,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 10, pp. 1355–1368, 2011.

[9] K. Chintamani, A. Cao, R. Ellis, and A. Pandya, “Improved telemanipulator navigation during display-control misalignments using augmented reality cues,” *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 40, no. 1, pp. 29–39, 2010.

[10] O. Demuynck and J. Menendez, “Magic cards: A new augmented-reality approach,” *Computer Graphics and Applications, IEEE*, vol. 33, no. 1, pp. 12–19, 2013.

[11] AndAR. [Online]. Available: http://code.google.com/p/andar/

[12] NyARToolkit. [Online]. Available: http://nyatla.jp/nyartoolkit/

[13] ARToolKit. [Online]. Available: http://www.hitl.washington.edu/artoolkit/

[14] NyARToolkit for Android. [Online]. Available: http://sourceforge.jp/projects/nyartoolkit-and/

[15] LeJOS, Java for Lego Mindstorms. [Online]. Available: http://lejos.sourceforge.net/

[16] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h.264/avc video coding standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[17] Y. Baguda, N. Fisal, S. Syed, S.-K. Yusof, and R. Rashid, “H264/avc features and functionalities suitable for wireless video transmission,” in *Wireless and Optical Communications Networks, 2008. WOCN ’08. 5th IFIP International Conference on*, 2008, pp. 1–5.