ROS, Android and Cloud Robotics: How to make a powerful low cost robot

Joao Paulo de A. Barbosa*, Fabio do P. de C. Lima†, Leonardo dos S. Coutinho†, Joao Paulo R. Rodrigues Leite†, Jeremias Barbosa Machado†, Carlos Henrique Valerio† and Guilherme Sousa Bastos†
* Graduate School of Aeronautics Engineering
Aeronautics Institute of Technology - ITA, Sao Jose dos Campos - SP, Brazil
Email: barbosajoaopa@gmail.com
†Institute of Information Technology and Systems Engineering - IESTI
Federal University of Itajuba - UNIFEI, Itajuba - MG, Brazil

***Abstract*—With the constant development in the robotics field, many solutions require a high hardware complexity to be executed resulting in an expensive robotic system. Furthermore, these solutions are usually closed box implementations that are hard to be reproduced. Hence, this project proposes a system structure to reduce these difficulties in robotics development. This system consists of a low cost robot with an embedded Android phone in a ROS environment. For testing this implementation, two Android phones, a laptop and a robot with an Arduino microcontroller have been used. The results of the study demonstrated several advantages of using this system, such as incorporating processing and sensing from the smartphone on the robot, having external heavy processing in a Server outside the robot and being able to connect the robot to the cloud. Therefore, using a popular Android device running a Rosjava application can be very useful in robotics applications and development, mainly due to the possibilities provided by this simple system.**

***Keywords*—ROS; Android; robotic systems; cloud robotics; Rosjava**

I. Introduction

The researches made on the robotics field are increasing frequently. The search for automatic systems performing complex activities and the replacement of people by robots on tasks that involve safety issues for example, are some of the motivations for studying it. The growing use of robots on industry can be explained by that, being mainly used for carrying heavy objects, solving logistics problems and reducing the amount of time that does not add any value to the process [1].

Another interesting point about the advances in technology is the popularity of smartphones all over the globe. However, most of the time, people do not even know the real potential of their devices. It contains many sensors, a great camera and usually a good processor, which can be used to substitute microcomputers and a large number of attached sensors, providing a low cost and cleaner solution to many applications. For academic purposes, using smartphones can become very appropriate, since the goal is to develop research using affordable equipment, an option that has already been proposed on other studies [2].

Also, the usage of cloud computing is becoming more and more popular [3], [4], [5], [6]. The idea of storing a file on a virtual drive that does not consume storage of a computer or smartphone and being able to read these files from anywhere at any time are some of its advantages. However, the cloud service is bigger than that, cloud computing allows a large amount of remote servers to be connected by the network sharing the same information and data storage. It is also possible to access Internet applications, increasing the possibilities of using complex programs that have already been developed and are running in powerful servers [3], [4], [5], [6].

978-1-4673-7509-2/15/$31.00 2015 IEEE

Furthermore, the infeasibility of processing heavy information on a single hardware instigates people to develop better ways of doing it. With the release of multicore computers, many solutions started to be implemented with the idea of breaking down a problem and distributing its small parts across multiple cores of a single computer. However, embedding this kind of complex hardware in a robot can be unaffordable, thus, for robotics purpose, external processing is a considerable solution for heavy computational problems.

The Robot Operating System (ROS) is a great tool that offers many resources to gather all these functionalities mentioned before, making them possible to be implemented [7], [8], [9], [10]. For this purpose, a practical application will be presented showing a robot, with an Android device embedded, being controlled by another Android device. A laptop is also used to demonstrate the possibility of external processing. Besides these two functionalities, video exchange between the smartphones and cloud computing for robotics (or cloud robotics) were also approached in this application. Therefore, during this project, the idea is to explore these advantages with a low cost solution in a practical way, demonstrating how an Android device in a ROS environment can be useful for robotics applications.

II. Theoretical Background

A. Robot Operating System

The Robot Operating System, also known as ROS, is a different operating system than the ones we are used to. The idea is not to schedule or manage processes, instead, it provides a structured communications layer above the host operating systems of a heterogeneous compute cluster [7].

The number of studies and researches made on robotics field involving ROS has been increasing recently as people

notice its benefits. Many systems with multiple robots and cooperative robots are being developed based on this framework [8]. The big community continuously developing ROS helps in the process of starting to use its tools, since several packages have already been developed and there is always someone able to help and answer questions.

In order to make different machines communicate with each other using this framework, it is necessary for all of them to be connected in the same peer-to-peer topology network. Once it is done, we need to establish a master, in order to allow the processes to find each other at runtime. Then, the communication is done by broadcasting through topics on a publish-subscribe model or through services, which works in a request-response model, similar to web services.

The master node works as a DNS, and it is usually started by the *roscore* command. It provides a name and registration for the nodes, enabling them to find each other. The usage of topics and services is only possible because of the tracking system created by the master. It manages the communication between processes and stores a list of topics, services and connected nodes.

With the growth of ROS usage, the need to adapt it to new programming languages and platforms has increased. Due to that, Rosjava was developed, making possible the use of Java, besides previously supported languages like C++, Python, Octave and LISP [11]. With Rosjava, using ROS on an Android app became possible. An Android device has several sensors onboard, which are exceptionally useful to robots. Thus, we can actually connect those robots to Android devices to take advantage of those features. In addition, Android devices typically have wireless access. As a result, when it becomes an integral part of the robot, it links the machine to the cloud, giving the ability to access unlimited CPU, memory and storage.

The advantages of using ROS are the possibility to distribute computational-intensive tasks to multiple machines, saving processing time, being able to make programs written in different languages, communicate with each other, and exchange information. The high amount of algorithms already designed for ROS is also an advantage when it comes to saving coding time, since there are many codes distributed as packages, such as robot navigation, mapping and vision.

With the use of ROS, many opportunities are generated for continuously developing robotics system. Heavy tasks can be finished faster, since, in some cases, they are done in parallel by multiple processes and multiple machines. The exchange of information between agents contributes for cooperative operations, extremely reducing limits of performing complex robotics activities.

B. Cloud Robotics

When using a robot on its own, we are limited to its processor, memory and storage. This means that complex applications require a very expensive robot. The goal is to make possible for everyone to have a robot and use it in daily activities. To achieve this goal, personal robots need to be affordable, and by connecting a robot to the Internet, we can take advantage of powerful computation, memory, storage and a massive data resource. As described before, an Android device can also be used as a link to the cloud.

Several robotics problems cannot be solved due to robots limitations. Even some expensive robot research platforms cannot store all information needed for certain tasks. However, a robot connected to the cloud can make use of some cloud services that are very useful for robotics. The Google Maps API is a great example on how to explore the amount of data present in a server in the cloud [12], [13]. Using this app, a robot can have access to all the Google Maps information, such as an address of a point of interest in the map. A robot that does not have any reference map stored before, can use Google Maps to know where it is and how to get where it needs to go. All this information is stored on a Google server (not in the robot) [12].

Another robot limitation is the processing capacity. A robot has a fixed computation power and it is not scalable, which makes it limited and with not enough processing for very heavy tasks. In this case, an interface with the cloud also becomes useful. By reducing processing on the robot, we reduce its battery consumption, its weight and the most important, its cost. The cloud enables smarter and affordable robots. This is very important to increase opportunities in robotics and, consequently, increase development in this area.

C. Embedded Processing and Sensing in a Low Cost Robot

Robot perception makes possible the machine interaction with the environment, being able to see, hear and feel physical quantities such as temperature, magnetic field or illuminance. This perception becomes possible by attaching multiple sensors and cameras to a processor that will retrieve all the information, make decisions and send back signals to actuators.

To have a device with the features mentioned above, we can either buy a robot that already comes with all these sensors and a powerful processor (which can be very expensive), or buy each individual part and build a system by our own (which is also an expensive option, since good sensors, cameras, and processors are not cheap). Besides that, assembling everything demands a lot of knowledge, making it a hard option.

Considering the difficulties to have a complete system, an alternative for a cheaper, but still powerful robot, is to integrate smartphones with an open hardware platform (*e.g.* Arduino [14]). The cost of this solution is low, since nowadays almost everyone has a smartphone and the price of an Arduino is very affordable. The Arduino platform is used to acquire and publish sensors data, subscribe to processed data and send commands to actuators.

Besides that, by using smartphones as the Brain of the robot, we eliminate the need of some external sensors and cameras, since they are already embedded. For example, Android phones usually have at least one good camera, accelerometer, gyroscope, GPS, speakers, barometer, microphone, temperature and illuminance sensors, which can be used by a robot through an Android application [11]. Other important advantages are the possibilities to connect to wireless networks and the Internet via Wi-Fi or via mobile internet data. There are some commercial robots that already take advantage of using smartphones, like Romo [15] and SmartBot [16].

III. Practical Application

The practical example of this project is based on a low cost robot and two smartphones running Android, one on the robot as its *Brain* and another as the robot *Controller*, and a laptop being an external processor. The goal is to demonstrate the advantages of the model proposed. Thus, we have implemented functions to control the robot using the *Controller*, to exchange video between the *Brain* and the *Controller*, to connect the robot to the cloud and to execute an external processing. The schematic for the communication flow is shown in Fig. 1.

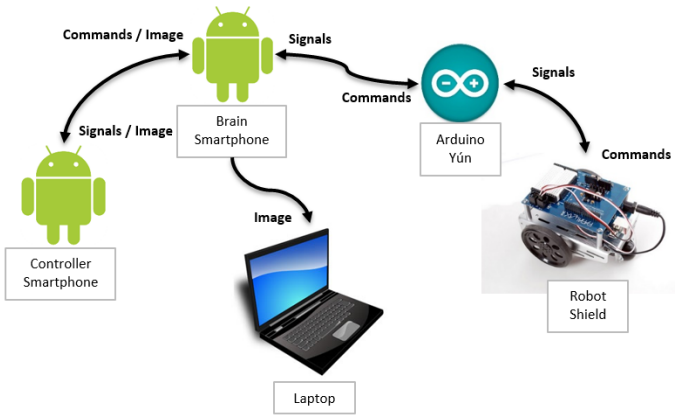

Fig. 1. Schematics of the communication flow.

Two different Android applications using ROS were developed. The *Brain* is the application installed on the smartphone placed on the robot. It is responsible for communicating with the *Controller* and the laptop, publishing information relative to the robot and receiving data that need to be sent to actuators. It is also responsible for communicating with the robots Arduino, sending messages that will be translated to drive the wheels and receiving data that need to be published. The *Controller* is the application installed on the smartphone used to drive the robot, which does not exchange any data directly with the Arduino. Instead, it publishes and subscribes to topics shared with the *Brain* application, in order to receive status from the robot and to actually interact with it.

IV. Platform and Communication Structure

A. Platform and Software Specification

1) Hardware: For a better evaluation of the advantages of the system proposed in this project, we used a simple and low cost educational robotic platform. This platform has two servomotors driving the front wheels and a rear caster. A shield board is used to control the motors, which can be connected to an Arduino board where the low-level program is located.

Once the application presented here intends to be as neat as possible and the entire system uses a local network with internet access, all communication between the agents (robot, mobile phones and server) should be done by using Wi-Fi. Therefore, the Arduino board that fulfilled these requirements for connecting the robot to the network was the Arduino Yn [17]. This board has two microcontrollers, one for the Arduino application as itself and the other with a Linux environment for managing the Wi-Fi, Ethernet, USB host and SD Card interfaces.

At this point, a simple and cheap robot can connect to the network by using Wi-Fi. However, it is still a simple robot with poor processing, memory and sensing capacities. So, as explained previously, embedding processing and sensing from a mobile phone in a robot can be very useful for applications that are more complex. That is the reason why an Android phone is used attached to the robot, as shown in Fig. 2. Besides these advantages in processing and sensing, this smartphone is also responsible for interfacing the robot with the entire system through the ROS framework. As cited above, the interface between this mobile phone and the Arduino is through the Wi-Fi.

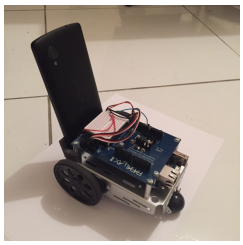

Fig. 2. The robot with the *Brain* smartphone embedded.

Other considerable advantage of using ROS is the possibility of interfacing with other devices in the same network. Another Android phone and a laptop were used for testing this functionality and showing how it works. This *Controller* smartphone controls the robot movements when running the application *Controller*. In addition, the laptop works as a server for image processing.

2) Software: With the hardware described, it becomes necessary to specify the software tools and languages used to develop the practical application. Starting from the lowest level, to program the Arduino, a basic C language was used. In addition, libraries such as the *Bridge* library, used in the communication, and the *Servo* library, used to control the servos, were helpful for the development of the Arduino program.

Other language needed was Java, which was used to make the Android applications. For this purpose, we used the Android Studio IDE [18]. However, in order to use ROS in these apps, Rosjava library was required. Rosjava is a client library that enables ROS communication in Java programs. In addition, other important tool used in the development of the Android applications was the Google Maps API, which was used to demonstrate how a robot could have access to a cloud service.

As stated previously, other important advantage of using ROS is the possibility to have some computational process done outside the robot. For demonstrating this situation, a software in C++ has been developed aiming to show an image processing in an external sever.

Both Android and Server applications used ROS as the main framework, and the version used in this case was the Hydro distribution.

B. Communication

Communication is a crucial part of this practical application. Firstly, because embedding a mobile phone in a robot

requires that the robot microcontroller communicates with the Android application. Secondly, because a robot with no external communication cannot be connected to the cloud or have an external processing, which are important features for a complete robot environment.

There are several different methods for connecting the robot microcontroller, to the *Brain* smartphone embedded on the robot. A wired connection is possible using the micro USB or the audio ports, however, we decided for a wireless connection. Therefore, the solution implemented in this case is a Web Socket interface where the Arduino is the Server and the smartphone is the Client [19]. As seen in Fig. 3, for setting up the communication in the *Brain* application, it is only needed to change the Arduinos IP address in the initial App screen.

Fig. 3. The initial screen of the *Brain* app.

Fig. 3 also demonstrates how the smartphone apps interact with the ROS environment. In the field Robot URI, it is necessary to provide the device address where the ROS Master is running. However, it is also possible to run the Master node in the smartphone application, but in this case, all other devices should be aware of its address. Therefore, the master is responsible for managing the connection between all the devices that are in the same ROS environment and to set up this system it is only necessary to inform the Masters IP address on each device.

V. Development and Results

Since we have set our environment, we need to build our two Android apps, the *Brain* and the *Controller*. The apps are very similar to each other when we take into account their structure as ROS Android apps. To publish and subscribe using ROS on Android, we had to implement publisher and subscriber nodes in Rosjava.

The publisher nodes implements a class called *NodeMain*, which simply gives a main loop entry point to all nodes. This main loop takes a *NodeConfiguration*. That configuration contains information such as the URI for the Roscore, which is the DNS node [20].

As sensor and camera information are published from both smartphones, both apps have the following publisher nodes: Fluid pressure; Illuminance; Imu (accelerometer); Magnetic field; NatSatFix (GPS); Temperature; and Camera.

The *Controller* application also has an *AddressPublisher* class, which publishes the robot address by using Google Maps API. At the other side, the *Brain* application has a *BatteryPublisher* class, which publishes battery information from the robot.

A. Robot Movements Control

As stated before, our application consists of controlling a low cost robot using two Android devices. The *Controller* will send its orientation to the *Brain*, and the *Brain* will receive that information and send to actuators. Graphics shown in the Fig. 4 displays the relation between the *Controller* smartphone inclination and the speed assigned to each wheel of the robot. When just action *roll* is executed, as in the region A and D of the graphics, both wheels tend to follow the angle of the smartphone. However, when action *pitch* is also executed, the speeds of the wheels change in order to make the robot turn left or right. The region B shows an inclination of the smartphone to the left side, resulting in the decrease of the left wheel speed. This difference between the speeds in each wheel makes the robot turn to the left side. The same behavior can be observed in the region C, however, in this case, the robot turns to the right, once the smartphone was tilted to the right side.

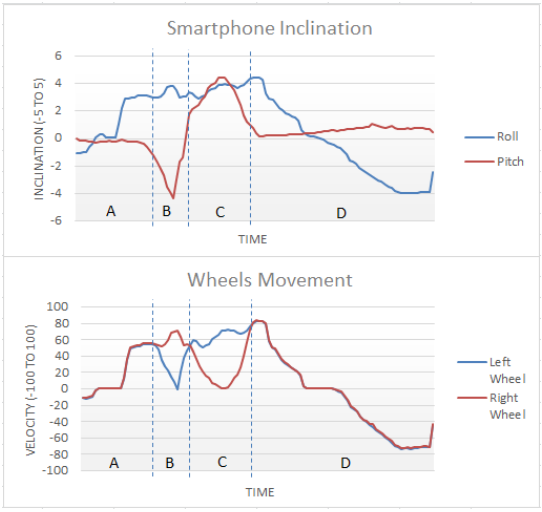

Fig. 4. Smartphone inclination and the correspondent movement of the wheels.

The *Controller* device publishes its orientation through the *ImuPublisher* node in a topic called *robot_name/control/imu*, as illustrated in Fig. 5. The message type that is being published is *"sensors_msgs/Imu"*, which holds data from orientation, angular speed and linear acceleration of the device. In this application, it was used the orientation. The orientation is constructed by the four quaternion parameters, *xyz* first, and then rotation *w*. The *Brain* device subscribes to *robot_name/control/imu* topic and process the information, so the orientation data can be used to control robot speed and direction. Then, the processed information is sent to Arduino, which sends to actuators.

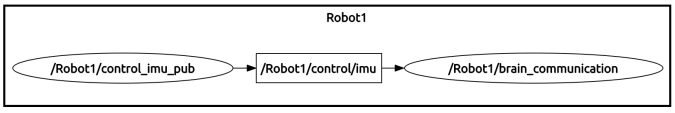

Fig. 5. Communication through the IMU topic.

B. Video Exchange

The Brain smartphone publishes its image as it was the eyes of the robot, thus, the subscriber can see what is in front of it, being able to avoid obstacles, follow objects or just explore a simple room. The topic to which it publishes is called *robot_name/brain/image/compressed*, and the type of the message is *sensor_msgs/CompressedImage*. It also subscribes to the images published by the *Controller*. Fig. 6 shows both smartphones exchanging their video stream. Since images are sent in real time with high quality, the transfer presents a short delay, which varies according to the strength of wireless signals.

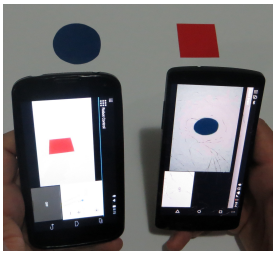

Fig. 6. *Brain* and *Controller* smartphones exchanging their video stream.

The *Controller* smartphone publishes its image so it can be used to interact with a person close to the robot when it reaches its destination for example. The topic used to publish it is called *robot_name/control/image/compressed*, and the type of the message is *sensor_msgs/CompressedImage*. Fig. 7 illustrates this communication.

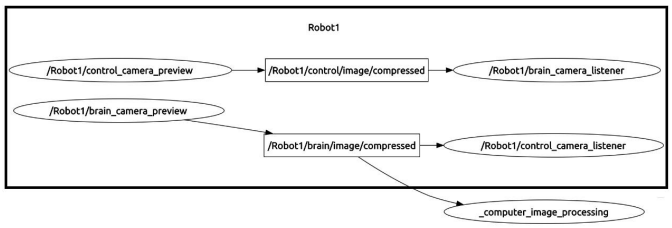

Fig. 7. Image topics communication between the devices.

For all of this exchange be possible, both applications must have a publisher node to send its own image, and a subscriber node to receive each others image. This subscriber node is linked to a *View* object within the Android app, so it can actually display the video on the screen of each smartphone.

C. Connecting to the Cloud

To illustrate how a robot can be connected to the cloud and the advantages of doing that, Google Maps API was used in this project. We decided to use Google Maps, since it is an application that provides a lot of useful information for robotics. By using Google Maps, a robot can know exactly where it is and where other robots are.

In this project, the *Brain* device, which is attached to the robot, publishes its GPS data on *robot_name/brain/fix topic*, as shown in Fig. 8. The *Controller* device subscribes to that topic, and send the information to Google Maps to determine the robot current location. On the *Controller* screen, the robot current location and the *Controller* location are displayed, as can be seen in Fig. 9. The access to Google Maps serves, data downloading, map display and response to map gestures is automatically handled by the API [12].

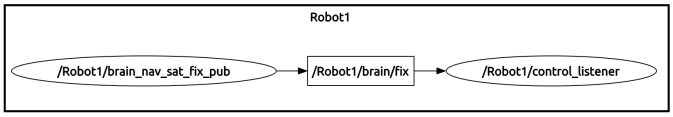

Fig. 8. GPS topic communication.

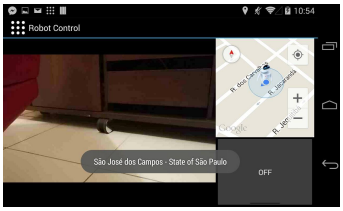

Fig. 9. The *Controller* application interface (Google Maps tool in the upper right corner).

Besides displaying the map, a button on the *Controller* smartphone screen was made to show robot current address. Using Google Maps API, it is possible to have a complete address for the robot (street, number, city, state, country, zip code) as soon as a button is pressed, depending only on the request/response time between client and server.

D. External Processing

In this project, we also developed a system to process information externally to show how powerful this option could be when using ROS. It is known that image processing is a very heavy task when it comes to computing time, since it requires a lot of comparisons and iterations. Our system already publishes images continuously, so dealing with computer vision processing makes more sense. To do that, we wrote a C++ code to run on a powerful computer capable of processing the image in a short period of time, which shows how flexible ROS is when using multi-language support. To show a portion of what can be done with image processing, we developed a program, using OpenCV, capable of locating a red object on the scene and returning its position, as displayed in Fig. 10.

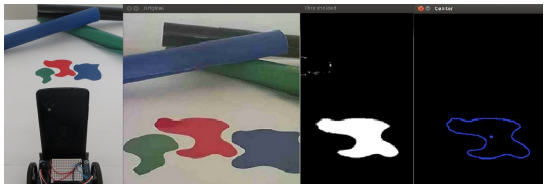

Fig. 10. From left to right: the robot getting an image to be processed, original image, image with threshold, processed image.

This processing can be described in different steps. The first one is implementing a subscriber to the *Brain* camera image topic, so that we can capture the robot viewed image.

After capturing the image, a kernel filter is used to blur it, eliminating most of the noise. This step is very important,

since computer vision is sensitive to light and background variations. The smartphone camera captures the frames in the RGB format, however, for working with image processing, it is easier to deal with the HSV format. The advantages of using this format are the good compatibility with the human intuition and the possibility of using just the hue matrix for color segmentation purposes [21].

With that explained, to eliminate all the colors that are not interesting, in this case, everything that is not red, we first converted the RGB image to HSV and then used a threshold filter, generating one black and white matrix where the white pixels represent the red object. The last step was to use an edge detection algorithm so we could get the boundaries of the object and consequently its centroid. To finish the external processing example, the last step is publishing the centroid found, which is done by the node on the topic named *robot_name/computer/obj_pos*, as shown in Fig. 11. Publishing the image and retrieving the centroid is done almost instantly, since the computer processes the data in a matter of milliseconds, and the communication relies on the image transfer time.

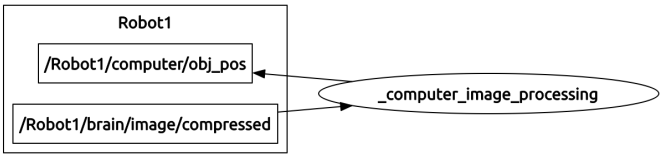

Fig. 11. The external node reading from the image topic and publishing to the possition topic.

VI. Conclusion and Future Works

As demonstrated in the studies presented in this project, the use of smartphones on robotics applications can be very useful, even more when using the framework ROS, which has several packages already developed for robotics purposes. Thus, the practical application created in this research was able to illustrate the innumerous possibilities when applying this model. With a simple low cost robot connected to a smartphone embedded on it, it was possible to have important robotic functionalities as communication to the cloud, external processing, besides several sensors, camera and a considerable processor coming from this smartphone.

To expand the usage of ROS and explore even more its features, the next steps are to implement it outside a local network and create multi-robot systems. Using the Internet to publish and subscribe to topics would make possible for any robot at any place in the world to share information and interact with each other, generating a broad robotics network. Since every smartphone can join a wireless network or have a data plan, the concept of ROS for mobile devices could be even more explored. Having a multi-robot system enables agents to share lists of tasks and manage the best ways of performing their actions. Working on similar activities simultaneously and cooperatively turns long and complex problems into easy and fast tasks.

After all, the results of this study showed that is possible to achieve a good processing power and sensing level with low cost equipment, and at the same time, take advantage of all the possibilities generated by the usage of ROS and the idea of cloud robotics.

Acknowledgment

The authors would like to thank Casimiro Montenegro Filho Foundation (FCMF), Aeronautics Institute of Technology (ITA) and Embraer for supporting our participation in the ICAR 2015 conference. We also thank FAPEMIG for fomenting crucial parts of this research. Finally, but not less important, our sincere thanks to ROS community for sharing their work and documentation.

References

[1] G. S. Bastos, C. H. C. Ribeiro, and L. E. de Souza, "Variable utility in multi-robot task allocation systems," in *Robotic Symposium, 2008. LARS'08. IEEE Latin American*. IEEE, 2008, pp. 179–183.

[2] N. Oros and J. L. Krichmar, "Smartphone based robotics: Powerful, flexible and inexpensive robots for hobbyists, educators, students and researchers," *IEEE Robotics & Automation Magazine*, 2013.

[3] L. Turnbull and B. Samanta, "Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure," in *Southeastcon, 2013 Proceedings of IEEE*. IEEE, 2013, pp. 1–4.

[4] S. Jordan, T. Haidegger, L. Kovács, I. Felde, and I. Rudas, "The rising prospects of cloud robotic applications," in *Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 327–332.

[5] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.

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

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[8] S. Cousins, "Exponential growth of ros [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 1, pp. 19–20, 2011.

[9] J. Kerr and K. Nickels, "Robot operating systems: Bridging the gap between human and robot," in *System Theory (SSST), 2012 44th Southeastern Symposium on*. IEEE, 2012, pp. 99–104.

[10] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing software with ros [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 12–14, 2010.

[11] D. Kohler and K. Conley, "rosjava–an implementation of ros in pure java with android support," 2011.

[12] "introduction to the google maps android api v2". [Accessed: 18 Apr. 2014]. [Online]. Available: "https://developers.google.com/maps/documentation/Android/intro"

[13] G. Svennerberg, *Beginning Google Maps API 3*. Apress, 2010.

[14] "arduino". [Accessed: 18 Apr. 2014]. [Online]. Available: "http://www.arduino.cc/"

[15] "romo robot". [Accessed: 28 Apr. 2015]. [Online]. Available: "http://www.romotive.com/"

[16] "smartbot: Smartphone robot". [Accessed: 28 Apr. 2015]. [Online]. Available: "http://www.overdriverobotics.com/"

[17] "arduino yun". [Accessed: 18 Apr. 2014]. [Online]. Available: "http://arduino.cc/en/Main/ArduinoBoardYun"

[18] E. Hellman, *Android Programming: Pushing the Limits*. John Wiley & Sons, 2013.

[19] I. Fette and A. Melnikov, "The websocket protocol," 2011.

[20] R. Hickman, D. Kohler, K. Conley, and B. Gerkey, "Google i/o 2011: Cloud robotics," *On-Line. Access date: 12 Fev. 2014*, vol. 9, 2012.

[21] K. N. Plataniotis and A. N. Venetsanopoulos, *Color image processing and applications*. Springer Science & Business Media, 2000.