# Development of API for autonomous navigation of robotic platform

**Antonio Valerio Netto[1], Guilherme Orlandini[2], Luiz Eduardo G. Martins[3]**

[1]DT CNPq
Brasília, Brazil

[2] Integrated Colleges of Ourinhos
Ourinhos, São Paulo – Brazil

[3] Institute of Science and Technology
Federal University of São Paulo, São José dos Campos, SP – Brazil

antonio.valerio@pq.cnpq.br, guilherme.orlandini@gmail.com,
martinsleg@hotmail.com

***Abstract.*** *The purpose of this article is to present the development of an Application Program Interface (API) based on the OpenCV library for RoboDeck. RoboDeck is a mobile robotic platform of Brazilian manufacture for the areas of education and research. This API was developed to facilitate to the students and researchers, the creation of new applications for the control of the mobile robot. To test this API was developed an application for the autonomous navigation using computer vision.*

## 1. Introduction

Throughout the years, human beings dared thinking and dreaming about the day when machines would have features and skills so near to those found in humans, which would reach the point of do tasks like them. Since operational tasks as sweep a floor, drive a car; organize objects in a house; or tasks related to human skills (sensitive and cognitive) as listening, seeing, speaking and moving.

Multidisciplinary knowledge has been giving support to researchers to accomplish that aims. Sciences as computing, electrical engineering, mechanics, social sciences and those that study the human behavior have been assisting to robot's creation. However, even with all advances reached by these sciences, the human being did not reach the goal of creating a machine which has all physical and cognitive skills inherent to a human being. That still seems far.

Although that seems far, important advances had been obtained in the latest decades. There are machines with skills (still with considerable limitations) of listening, take decisions, speaking, seeing and interaction with objects and human beings. Among the skills mentioned previously, the one related to robot vision (specifically computer vision) have been target of several researches in the computer science field [Lopes 2010] [Rivera-bautista et al. 2012] [Zhao and Jia 2009] [Nehmzow 2003]. Starting with the knowledge of how the human vision works, researchers have been trying to create solutions to turn that skill possible to robots in a way of a tridimensional objects perception of a robot in an environment and the association of those objects to a previous knowledge is able to accomplish nicely. The referred task is quite complex, this task not only consists in to capture images through retina in ocular globe, but also associates all knowledge acquired for the human being throughout the whole life.

Vision-based navigation of robots has been an active field of research in the past decade. There are many challenges on developing vision computer systems due the needs of a comprehension of the environment in which it is placed. That challenges can also vary considering indoor or outdoor environments. Outdoor environment is more complex to modeling and implements a good vision-based navigation due a lot of variables, but small structures that exists in that environments such as roads, lane markings, etc.; help reduce the model complexity [Xie and Lu 2013]. On the other hand, indoor environments are relatively easier to model considering the inherent structure in buildings, thus becoming mathematically tractable as compared to the outdoor environments. The real challenge in indoor navigation is the localization of the camera system, which involves the vision system estimating its location in the environment in which it is navigating [Quigley et al. 2009].

Navigation may be vaguely defined as the processing of finding a suitable and safe path between a start and a terminal point for a robot to traverse [Liang et al. 2016]. Visual navigation as specific application to mobile robots has brought countless contributions. This is mainly due the rise of possibilities for their application in autonomous mobile robot navigation. Traditionally, navigation solutions primarily based on vision are typically used to the Autonomous Ground Vehicles (AGV) [Linder e Arras 2016].

Several tools have been built aiming to make easier the building of computer vision applications. One of them is the OpenCV library [Budiharto 2014] [Culjak et al. 2012] [Shrivastava 2013] [Szabo and Gontean 2015], which has functions of computer vision, motion detection, tracking, image patterns recognition and camera calibration that contributes to create computer vision applications without requiring the use of low-level code [Suarez et al. 2014]. That library was adopted in this work.

On the other hand, there is the mobile robot. A mobile robot is an automatic device, which is able to move inside a pre-defined environment and interact with that, so the way that the robot will move in the environment must be considered. That device needs software, which makes possible reach some autonomy. In this work, the mobile robot object of studies was the RoboDeck. RoboDeck [XBOT 2015] is a mobile robot wheeled that can be expanded by addition of robotic parts as grabs, mechanical arms, sensors and other devices. The robot was developed and released in 2010 by a Brazilian company called XBot (*www.xbot.com.br*), aiming to provide the educational development, mainly in the field of mobile robot research.

Is intended in this work the development of computer vision applications using the OpenCV library, and also integrate these applications with RoboDeck, providing a partial level of autonomy and bringing some contributions related to research in software development and control of mobile robots.

In the context of vision-based navigation the problem of obstacles is often mentioned. As the RoboDeck is a relative new platform, software applications are necessary to provide autonomy for the RoboDeck to perform some tasks. Although that robot has several resources, the fact of it does not have applications that can explore its resources results in a poor use of the robot. The approach of this works is related to the use of the OpenCV library in order to create computer vision applications [Choset 2005], it will help researchers to use that knowledge on future research with RoboDeck, and also on others robotic platforms [Yeoun-Jae et al. 2011].

In this work, was also designed and developed an API, which provides high-level robotics command for communication with the MAP (High Performance Module) of RoboDeck, in order to make the development of applications for the RoboDeck more productive, simple and fast. Although the robotic hardware chosen for the developing of this work was RoboDeck, the reached results related to use of elements such the OpenCV library, histograms, appearance-based match, can be easily reused for others robotic platforms [Yang et al. 2010] [Fernández-caramés et al. 2014].

The main goal of this work is to present and discuss the development of computer vision applications based on the histogram technique for the RoboDeck platform using the OpenCV library, as well to integrate those applications with that robot, providing then some autonomy for the robot. The specific goals for this work are:

- To validate the RoboDeck for the use of computer vision application techniques;
- To create an API that supports the development of applications that communicates with the RoboDeck High Performance Module;
- To create applications for autonomous navigation based on the color recognition and captured objects from the acquired image using the created API.

The developed activities begun with studies related to Computer Vision. From those studies, was observed the OpenCV library would be indicated for computer vision applications development. Based on the collected materials about the OpenCV library, studies and developing of simple applications (using Visual Studio 2008 IDE) were performed. Analyzing the results, it was noticed that the OpenCV had features that would allow the development to be more practical and fast. After advances related to the development of computer vision applications, the source code was modified to perform at Linux operation system (Debian).

Once understood the concepts related to computer vision, was necessary to research about mobile robots and vision-based navigation. It was performed also studies about RoboDeck (mobile robot object of study aiming the development of computer vision application). It was consulted the RoboDeck manual for an understanding of the robot´s operation and how could it be integrated with computer vision applications. It was also accomplished practical experiments with that robot.

Based on the understanding of that robot´s architecture and its protocol communications, was developed an API using C++ language aiming make easier the task of send robotic commands to MAP, facilitating the integration of computer vision applications with the robot. That API was firstly tested with MAP running on simulated mode. Preliminary tests were conducted based on the mockup hardware, which was a board that has hardware with a way of operation very similar to RoboDeck. After concluded the mentioned activities, the following steps were performed:

- To use the Open-Loop approach for image capture and robotic command sending to the robot;
- To use a reactive approach, where the robot acts according to the information received by sensors;
- The vision-based navigation would be in an indoor way, without map, and using appearance-based match;
- The appearance-based match would be viable generating histograms from images and performing comparisons between histograms.

## 2. Computer Vision and OpenCV Library

For human being, the visual world perception seems to be simple. We are able to understand and discern easily colors, textures, shadows, luminosity and other visual elements. In a family photo, humans are able to discern where the people are (whether they are or not their relatives), the apparent age, and also to imagine the emotional status of a person based on his facial expression. That task is not so simple when is thought about computers. The same image, saw under a different luminosity, can be interpreted in a completely different way by a computer.

The images contained in the Figure 1 are related to the same door; however, the right door has at the bottom a brightness caused by incidence of sunlight. In spite of those images are under different luminosities, is simple for a human being conclude that is the same image, by the reason that human beings generally has a full knowledge about images, variations and learning related to, realizing easily being that images related to the same door. However, for a computer, those two doors are entirely different, what means, those are different "images". While human has an entire knowledge about objects, a computer captures through a camera only images (those images needing to pass by a process that convert it to a numerical data, it being able to be interpreted by a computer).

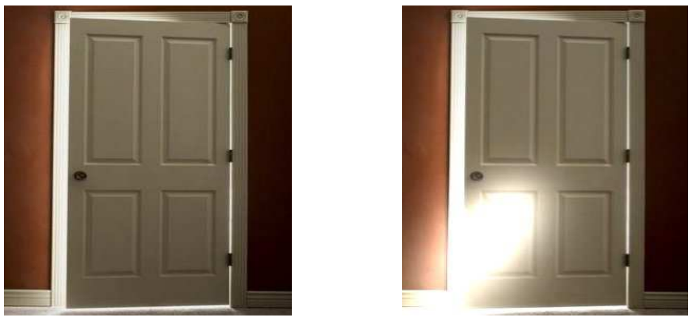


**Figure 1. Door without light reflex and door with light reflex**

According to Bradisk and Kaehler (2008), Computer Vision is defined by the data transformation of an image originated by a camera in some kind of decision or in a new way of representation. Transformations are performed based on input data to accomplish some particular aim, as: Find out whether there is a human being in the scene, how many objects there are in the scene, capture a human face image and verify if that human has authorization to be in that refereed place, and others information about. A new representation of the image based on transformations can be done using several techniques, such as change a colored image in gray scale, to smooth the image in order to take out the luminosity, obtain the camera motion done from the image sequence, etc.

In other definition, Shapiro and Stockman (2001) affirm that the computer vision goal is to take decisions about physical real objects from scenes based on captured images. When the image is digitized, it is stored in a computer memory as a data matrix, where each element represents an image pixel. That numerical matrix is so, processed by computer to extract the wanted information from the image [Dawson-howe 2014]. The data matrix related to the image could have its values changed for a digital image processing operation when necessary. Once changed that values, the image will be consequently changed. There are many operations related to digital image processing that can be used at several situations and needs. Among those operations, we can mention the filtering and smoothing operations, and operations related to histograms.

In computer vision, an obstacle that must be considered about the recognition of objects in a scene is external factors as lighting, where a simple solar reflex that focus on an object can become more difficult that recognition. Thus, the main goal of filtering and smoothing techniques is to process a determined image resulting in an image more suitable than the original image for a specific application. Those filters are categorized in high-pass and low-pass filters [Russ 2011].

The histogram of an image registers the distribution of frequency of the levels of gray of an image. Imagining an 8-bit-image, it´s possible think in a table with 256 entries, indexed from zero to 255. The darker gray level is represented by zero, and the clearer gray level is represented by 255.

The Figure 2(a) shows a data matrix related to a 25-pixel-image. In that case, the picture has gray level pixels going from zero to three. In the Figure 2(b), there is the count and classification of those pixels. As is possible to realize, in h(I) there is 6 pixels with 0 gray level, 9 pixels with 1 gray level, 4 pixels with 2 gray level and 6 pixels being of 3 gray level. In the Figure 2(c), a graphical histogram was generated based on counting and classification of pixels. In that histogram, there are four bins (gap or segment of the histogram), which are related to each gray level found in the image. The gray level more frequent has the 1 gray level, thus, the bin 1 is showed having a biggest length in the histogram.

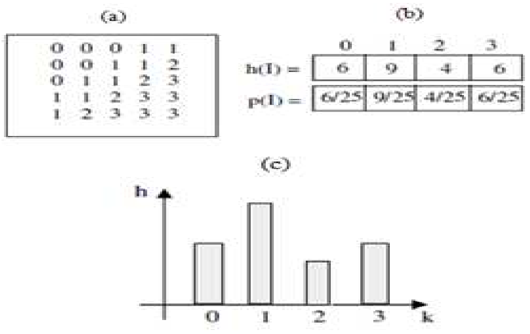


**Figure 2. Histograms related to images from an automobile and a part from a window [Hongming et al. 2005]**

The histograms have a large application in computer vision. Through them is possible as an example, compare the degree of similarity between two images by comparing their histograms. Using histograms is possible have object´s classification, which consists in an identification of an object in a scene. In the Figure 3, is showed histograms related to two images. Being the histograms inherent to the images different each other, is possible to conclude not be the same object.

OpenCV is an open source programming library used for development of computer vision applications. Several researches have been adopting this library to perform works associated to Computer Vision [Jahne and Hausseker 2000] [Chaczko and Braun 2010]. The Intel Corporation developed the OpenCV using the C and C++ languages.

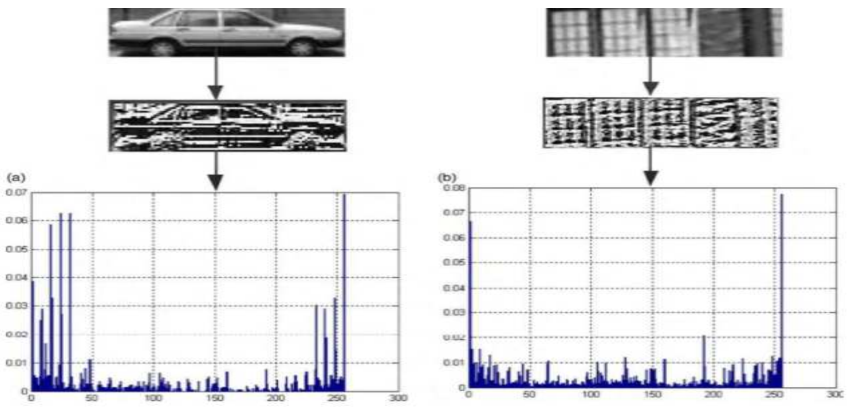


**Figure 3. Door without light reflex and door with light reflex**

The OpenCV library was idealized in order to become computer vision accessible to users and programmers in areas such as real-time-computer-human interface and robotic. The library is available with the source code and binaries optimized for Intel processors. When running, an OpenCV program calls automatically a DLL (Dynamic Linked Library) that detects the model of processor and loads an optimized DLL for the one. With OpenCV package is offered the library IPL (Image

Processing Library), which OpenCV depends on partially, besides the documentation and a group of demo codes [Ching et al. 2009].

The OpenCV has about 500 functions related to several fields of computer vision, such as processing image functions, moving detection and tracking, pattern recognition and camera calibration. They are high-level functions that make easier solving complex problems in computer vision. The Figure 4 shows an overview of OpenCV architecture.

• CXCORE: It has data structures, functions for data transformations, object persisting, memory management, error manipulation, besides text, draw and math functions.

• CV: It has processing image functions, image analysis structure, functions to tracking and moving detection, patterns recognition and camera calibration.

• ML: It has function for statistics classifications and tools for clustering.

• HighGUI: Module responsible for providing interface control functions and input devices. Is important observe that besides the 4 modules mentioned, there is also a module called “CvAux”, which is related to vision algorithms, however, it was in an experimental phase.

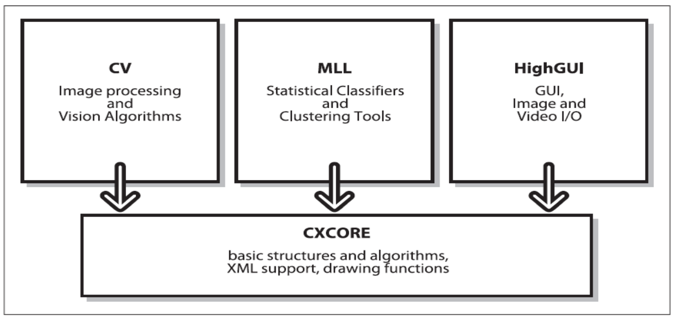

CV
Image processing and Vision Algorithms
MLL
Statistical Classifiers and Clustering Tools
HighGUI
GUI, Image and Video I/O
CXCORE
basic structures and algorithms, XML support, drawing functions

**Figure 4. Basics structure of OpenCV Library [Adept 2011]**

## 3. Mobile Robot

Siegwart and Nourbakhsh (2004) says that the mobile robotics is a field of study relatively recent, which after important decades of evolution, remains as an interesting research subject due to wide applicability in different domains and its relevant aspect in economy and technology.

In a recent past, when it was mentioned about robots, people imagined industrial robots and robotic-mechanical-arm support production and manufacturing. Actually, the attentions are towards to mobile robots, which are able to move in an environment which they are [Chen et al. 2009]. For a mobile robot be considered autonomous, it must have the capacity of locomotion and make decisions without human intervention. Hence, some aspects must be considered, such as capacity of perception (sensors are able to gather information about the environment where the robot is), capacity of act (actuators and motor able to perform actions, for instance a robot displacement in an environment), and intelligence (capacity for deal with different situations, to solve and perform complex tasks) [Mcginn et al. 2015]. Although there is not a final taxonomy for classify mobile robots, it is possible classify those according some characteristics. According Chen, Chen and Chase (2009), the mobile robots can be classified in three big groups: terrestrial, aquatic and aerial.

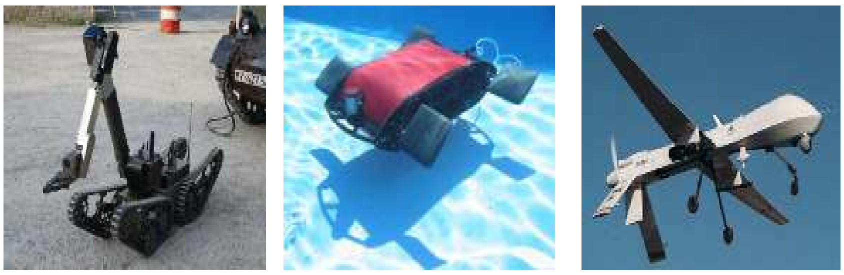


**Figure 5. Terrestrial, aquatic and aerial mobile robots [Chen et al. 2009]**

Robots can also be classified according the autonomy level:

• Tele-operated Robots: The operator, using a control device, sends commands for the robot do tasks.

• Semi-Autonomous robot: The operator sends a macro command to robot, and the one performs tasks being able of making decisions.

• Autonomous Robots: the robot performs tasks without human intervention, making his own decisions based on received information by its sensors.

The autonomous intelligent navigation of robots consists on robot locomotion inside a determined environment, so that it has a capacity of detecting obstacles and deflecting during a course. In order to reach this goal, several real-world properties must be considered [Russel and Norvig 1995]. The planned, reactive [Brooks 1991], hybrid [Tomatis et al. 2001] approaches, and the using of maps [Thrun and Bucken 1996], elucidates way to line up real-world properties with robotics navigation.

The vision-based autonomous navigation systems consist of a robotic vehicle, image-capture devices, and actuators that allows robotic vehicle act under the environment. According to Kundur and Raviv (1998), the building of vision-based autonomous navigation systems must consider the following issues:

• What is the relevant information to be extracted from a sequence of images?

• How to extract that information from bi-dimensional image sequence?

• How to generate control commands to the robotic vehicle based on the extracted visual information?

Extensive research has been done in the field of vision-based autonomous mobile navigation. In the work of Andresen, Jones and Crowley (1997) is proposed an indoor-navigation system, being necessary a previous knowledge about the indoor environment (using previously acquired images). Guzel (2009) presents a proposal of a system navigation where is attached to a robot a PTZ (Pan-Tilt-Zoom) camera that works as a primary sensor. The goal of that work was the robot performs navigation in an indoor environment, without any previous knowledge about that one. In the Valasek et al. (2005) proposal, was used a vision-based navigation to achieve the refueling of unmanned aircraft process. Johns and Yang [Johns and Guang-zhong 2010] presented and approach of scene associations for autonomous robotic navigation, registering landmarks associated to a group of images that formed a scene. Saunders et al. (2015) proposed an algorithm that uses information from a directional camera to find features that refers wall in an indoor environment. The goal was do the robot stop when it was detected a wall near to the robot, by using only a camera as a sensor,

Change and Chuang (2011) presented an approach of robotic navigation and building-map using laser projecting by a mounted projector under the mobile robot, and other camera mounted on the walls of an indoor environment. A 3d-map was built in real-time and used to do the navigation inside the environment. The referred map was built based on gathered coordinates from grouped information given by both camera and laser projection. The Chang and Chung´s work was done based on the technique called SLAM (simultaneous localization and mapping), that was resulted by the work of Chen et al. (2010).

According Moravec (1980) is essential for a vision-based navigation system some knowledge about the environment using the visual information, and there is two different approaches for use that visual information. One based on Open-Loop control, and other based on Closed-Loop control.

- Open-Loop Control: on this approach, the image information extraction and the robot control are two tasks that occur at various times, where the snapshot and the image processing are done firstly, followed by the generating of a control sequence for the robot. That approach is used in this present work.
- Closed-Loop control: Approach whereas the snapshot and image processing as the control sequence generating for the robot are done simultaneously. Based on the camera positioning so that decreasing the error rates related to displacement speed and positioning.

According Desouza and Kak (2002), the vision-based navigation is one of the most important navigation methods, and two techniques reached advances associated to that: vision-based indoor navigation and vision-based outdoor navigation.

Images are pre-stored in the robot memory as templates (models). The robot locates itself and navigates around the environment comparing what is captured by camera with the previously defined template. As using examples of this approach, there is the work of Matsumoto and Ito (1995), where a sequence of images was pre-stored working like a robot´s memory. The robot ran the same route comparing the captured image by camera with the pre-stored images. Jones, Andresen and Crowley [20] created a way for templates be associated to actions that the robot should perform. When an image captured had a matching in a determined level, the robot performed the action.

Ohno, Ohya and Yuta (1996) done practically the same of Jones, however, the robot performed the actions faster.

Ukida, Terema and Ohnishi (2012) used the matching-appearance-based technique to give to a robotic arm (that is equipped with two cameras) the capacity for do object-tracking. Mishra et al. (2013) used the same technique to provide to a robot the capacity of tracking and chasing objects. The vision-based navigation is still an open field when mentioned about research. In an autonomous robot navigation associated to unknown environment, is necessary considering several variables, as relate to computer vision as relate to navigation. The image processing has considerable importance for the success of vision-based navigation.

Currently there are several robotic hardware, used in research groups around the world. Among them, it could be highlighted the hardware Pioneer 3 [Adept 2011], IRobot Create [IROBOT 2011], Lego Mindstorms NXGT [LEGO 2012], RoboDeck [XBOT 2015] and others. At this work, it was used the RoboDeck Robotic Hardware.

The RoboDeck is a Brazilian robot, developed by XBot Company located in São Carlos, state of São Paulo - Brazil. The XBot Company is specialized in robot manufacturing for education, researching and developing. Among the RoboDeck´s features, it has Wi-Fi communication, Bluetooth and ZigBee, camera, infrared and ultrasonic sensors, GPS, compass, accelerometer, among other inputs. RoboDeck can reach speed between 3 and 5 km/h and have an approximated weight of 15kg; where the weight variation is related to the number of batteries [Quigley 2009].

The RoboDeck´s software is composed by firmware written in C/C++ of microcontrollers in low level. In a high level, it has the high-performance module instructions group, which allows interact with autonomous systems or external controllers (applications), besides SDK´s, one of then written in C# and other in Java ME, for the control application development through peripherals.

The essential RoboDeck´s hardware is composed by two microcontroller modules: A ARM9 microcontroller for basic functions as moving and data sensors collecting, and a Jennic microcontroller for communication management with interfaces and ZigBee network. Besides, the robot has a NanoITX board with a high-performance module integrated with the operating system Debian Squeeze. The optional hardware of RoboDeck (also called High-Performance Board) has the goal of providing autonomy and broadband communication for the robot.

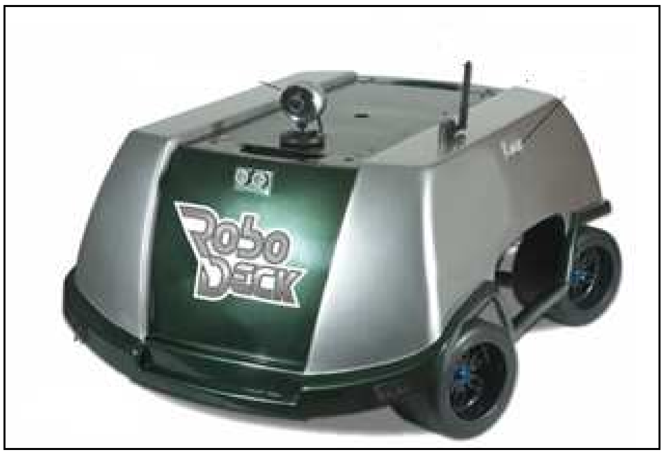


**Figure 6. The RoboDeck Hardware [XBOT 2015]**

**Figure 7. RoboDeck´s High-Performance Board**

That board is endowed of 4 GB-flash-memory (that contains Linux system) and a processor that can reach 500 MHz of clock. The board supports video, keyboard and mouse, allowing the robot being used like a computer. Under this board, also works a software called "Módulo de Alta Performance" (MAP), in English language, "High-Performance Module".

The MAP allows applications controls the robot essential hardware through commands sent directly to MCS - "Módulo de Controle de Sessão" (in English language, Session Control Module). The MCS implements the concept of session to the commands from applications, in order to not allow robot reply for two applications simultaneously. The MCS also is responsible for authentication of applications besides to send directly for MCR (Módulo de Controle Robótico - Robotic Control Module) the robotic commands sent by MAP. The Figure 8 presents an overview of the RoboDeck's architecture, showing its connection with the essential and optional hardware.

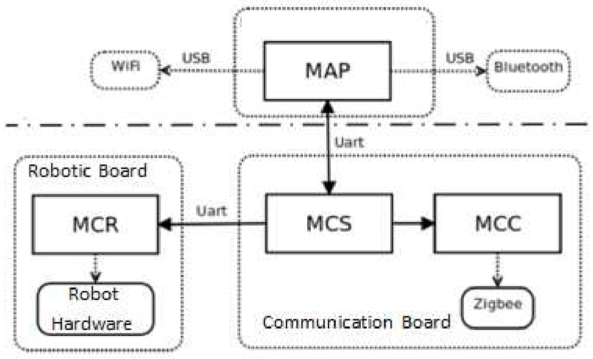


**Figure 8. RoboDeck's architecture overview [Munoz 2011]**

Two factors were fundamental for the choosing of RoboDeck in this work. The first one, is related to RoboDeck be a Brazilian robot (which makes more easy aspects relate to support and maintenance). As a second factor, was considered a better processing capacity when compared with others robotic hardware.

## 4. Robodeck´s API for Communication With Map

For the RoboDeck performs elementary actions as moving forward, back, turn and read sensors and other actions, is necessary to send commands to the robot. Those command sending can be done by MAP. Hence, for an application executes requests related to any robot action, that one must send commands to MAP. Presently, this command sending is done in C language, using low-level commands. This factor can become the application development more complex and prolonged.

Considering the factor mentioned, was thought as viable to develop an API that made the robotic application development more efficient and productive, so that developers do not need to be worried about details relate to communication between the application and the robot. In order to develop this API, was important the reading of RoboDeck´s manual and documents, mainly a document called "Protocolo de Comunicação Externa" (External Communication Protocol). Those documents are available for download at the XBot´s website (*www.xbot.com.br*)

The API was designed with Object-Oriented concepts (using C++ Language), where each class is responsible for a group of functions inherent to RoboDeck. Once Linux environment is offered with the high-performance board, it was designed for that environment. The API design is illustrated in the Figure 9 through a UML class diagram. This figure shows how the classes were organized according to specific functions that are the following:

• RoboDeckInfo – class responsible for returning general information about the robot and commands session;

• RoboDeckMoving – class that contains methods responsible for robot movement;

• RoboDeckSensors – class that contains methods related to use of sensors;

• RoboDeckLocation – class that contains method used for gathering positioning information;

• RoboDeckMap – class that contains method related to information from MAP;

• RoboDeckCam – class that contains method for using camera;

• RoboDeck – class that contains low-level commands responsible for communication with MAP. This class is available by XBot Company;

• Controller – is inherent to the application that will use the API. In this work, the class Controller will be the application of computer vision.

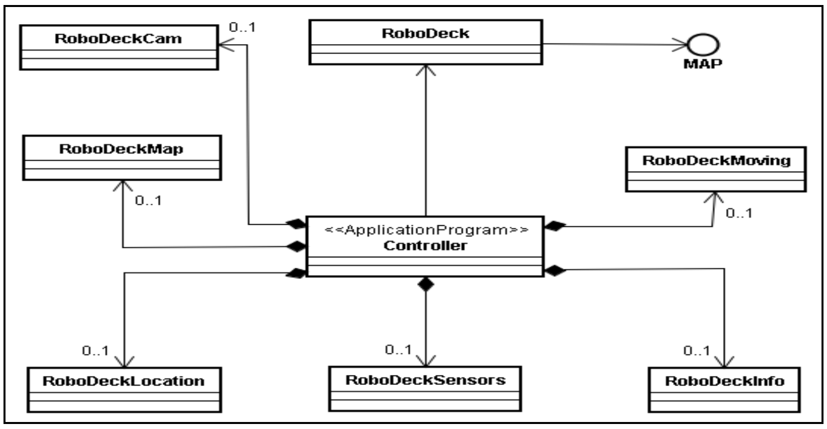
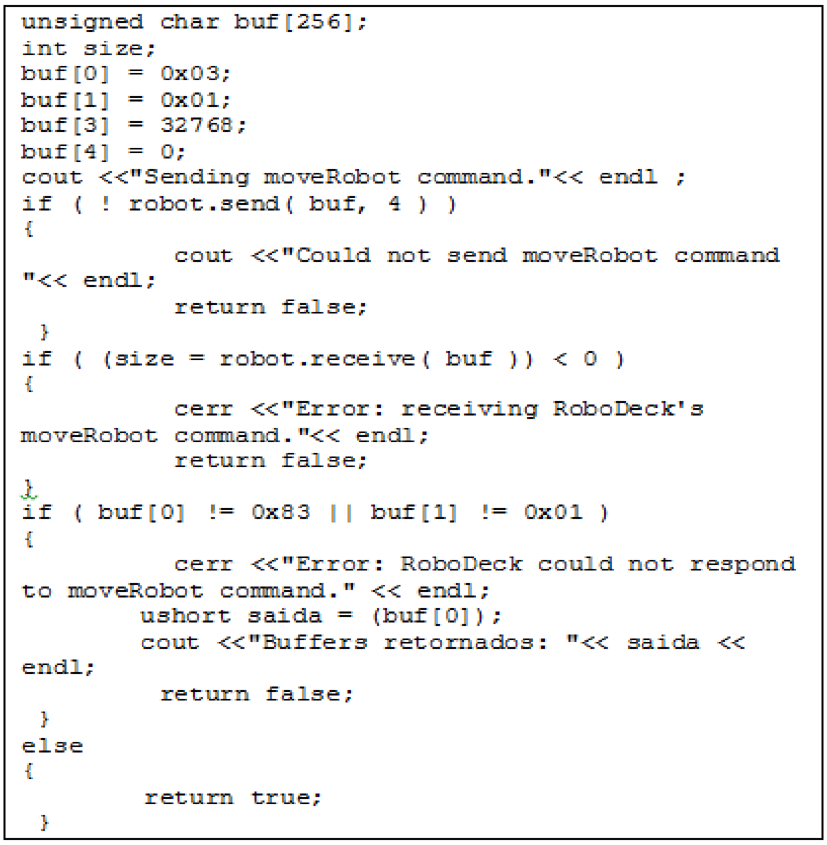


**Figure 9. RoboDeck API class diagram for communication with MAP.**

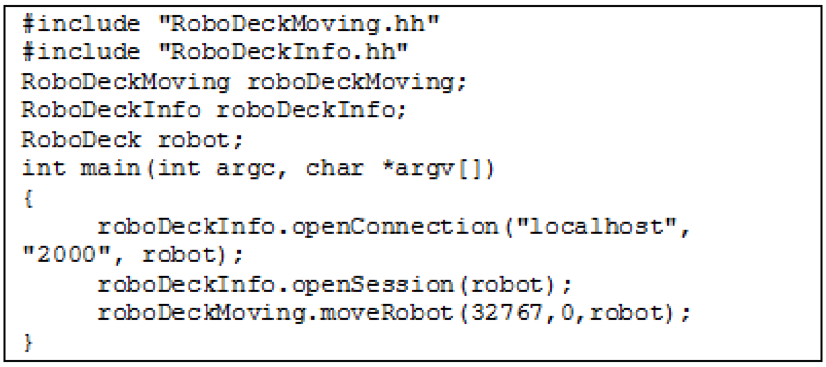
The classes RoboDeckMoving and RoboDeckInfo were implemented in this work. The other classes were designed but not implemented yet. Following, there are the names of the methods contained in the classes RoboDeckMOving and RoboDeckInfo, and explanations about that ones. As mentioned previously, the RoboDeckMoving class has methods responsible for robot movement. To have a better comprehension of the advantages given by that API, will showed two code blocks used for doing the RoboDeck to move forward. The first code showed was used "before" the creation of API. Moreover, the second is the code that uses the API. It can see observed following:

**Code used for RoboDeck to move forward (before the creation of API)**



```
unsigned char buf[256];
int size;
buf[0] = 0x03;
buf[1] = 0x01;
buf[3] = 32768;
buf[4] = 0;
cout <<"Sending moveRobot command."<< endl ;
if ( ! robot.send( buf, 4 ) )
{
          cout <<"Could not send moveRobot command
"<< endl;
          return false;
 }
if ( (size = robot.receive( buf )) < 0 )
{
          cerr <<"Error: receiving RoboDeck's
moveRobot command."<< endl;
          return false;
}
if ( buf[0] != 0x83 || buf[1] != 0x01 )
{
          cerr <<"Error: RoboDeck could not respond
to moveRobot command." << endl;
        ushort saida = (buf[0]);
        cout <<"Buffers retornados: "<< saida <<
endl;
         return false;
 }
else
{
        return true;
 }
```

**Code used for RoboDeck to move forward (after the creation of API)**



```
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"
RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robot;
int main(int argc, char *argv[])
{
     roboDeckInfo.openConnection("localhost",
"2000", robot);
     roboDeckInfo.openSession(robot);
     roboDeckMoving.moveRobot(32767,0,robot);
}
```

## 5. Developing of Vision-Based Navigation Application

A vision-based navigation application was developed in this work having the goal of processing; interpreting the images captured from RoboDeck´s camera in order to proving to it the possibility of make decisions related to environment navigation. The Figure 10 presents a sketch of RoboDeck´s navigation in an indoor environment being the vision-based application running. It was pre-stored in the application, for futures comparison, three images, which can be visualized in the Figure 11. Those three images are mentioned in this work as templates.

The Figure 11(a) presents the signal to turn left. When the application performs the recognition of that signal through the camera and the robot is near by 20 centimeters of the signal, the application will to send commands to the robot (through the API) so that the robot turn left. The same thing will occur related to the Figure. 11(c), however, the action will be turning right. The Figure 11(b) refers to a signal used to stop.

The signals shape had been chosen purposely having the goal of become the recognition more accurate. As the chosen technique for the signal recognition was comparing by histograms (generated from spatial black pixel distribution in an image), the similarity between that signals can decrease the accuracy (factor observed during the performing of the Experiment 1 of this work with the signals showed in the Figure 12). For a better comprehension of this issue, there is in the Figure 12, a possible scenario where there are similarities between different signals.

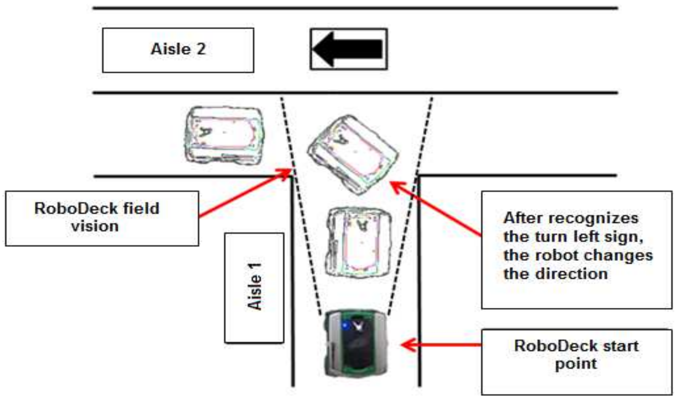

Aisle 2
RoboDeck field vision
Aisle 1
After recognizes the turn left sign, the robot changes the direction
RoboDeck start point

**Figure 10. Sketch of the robot navigation in an environment when running the application**

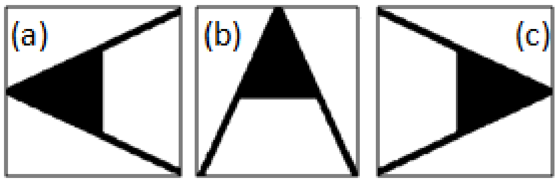


**Figure 11. Signals that can be recognized by the robot**

In the Figure 12(a) and 12(b), there are the chosen arrows for be used in the vision application. In the Figure 12(c) and 12(d), there are the histograms related to the arrows. Following, it will explain the algorithm used for the vision-based navigation application and pieces of relevant codes related to the arrow recognition.

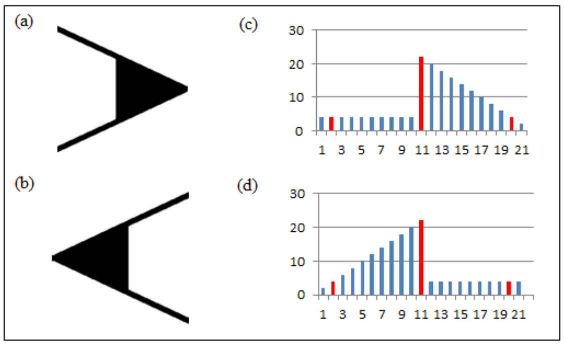


**Figure 12. (a) e (b) Arrows chosen as templates. (c) and (d) Histograms related to the chosen arrows**

After the declarations of used libraries, declarations of variables, and image loading of the signals that are recognized by application (templates), a while loop is started in order to run indefinitely. The algorithm performs the following steps:

1. Inside the loop, the captured image is read frame by frame. One frame is read in each loop of iteration.
2. The frame is converted in a binary image.
3. The binary image is transferred as a parameter to a method called SearchObject(). This method search for some object (edge) in the image. In case of an edge is found, is applied an image threshold, so that to be considered just the region related to the edge.
4. Is created a new image from the found edge.
5. The template is resized to have the same size of the new image.
6. Once resized the template, is generated a histogram from the template (calling the DohProjection() method and passing for the one the resized template.
7. The new image and the histogram of the template are sent to matchObject().
8. The matchObject(), having the new generated image and the histogram of the template, performs the following actions:

   8.1 Generates a histogram from the new image (calling the DohProjection() method and passing for it the new image).

   8.2 Compares the histogram of the new image with the histogram of the template through the cvCompareHist() method.

   8.3 Returns a value between 0.0 and 1.0, where 1.0 means 100% of compatibility between the histograms. In that application based on experimental tests done, was considered 78% compatibility as enough to validate the experiment.

   8.4 Once recognized the signal, is verified the identified image size. If in the captured image by camera the signal has a width greater than 160 pixels, it means the robot is very close to the signal (about 20cm approximately), hence the robotic command related do that signal must be sent to the robot for the robot turn left, right or stop.

Aiming a better explanation of that algorithm, there is a following activity diagram in Figure 13.

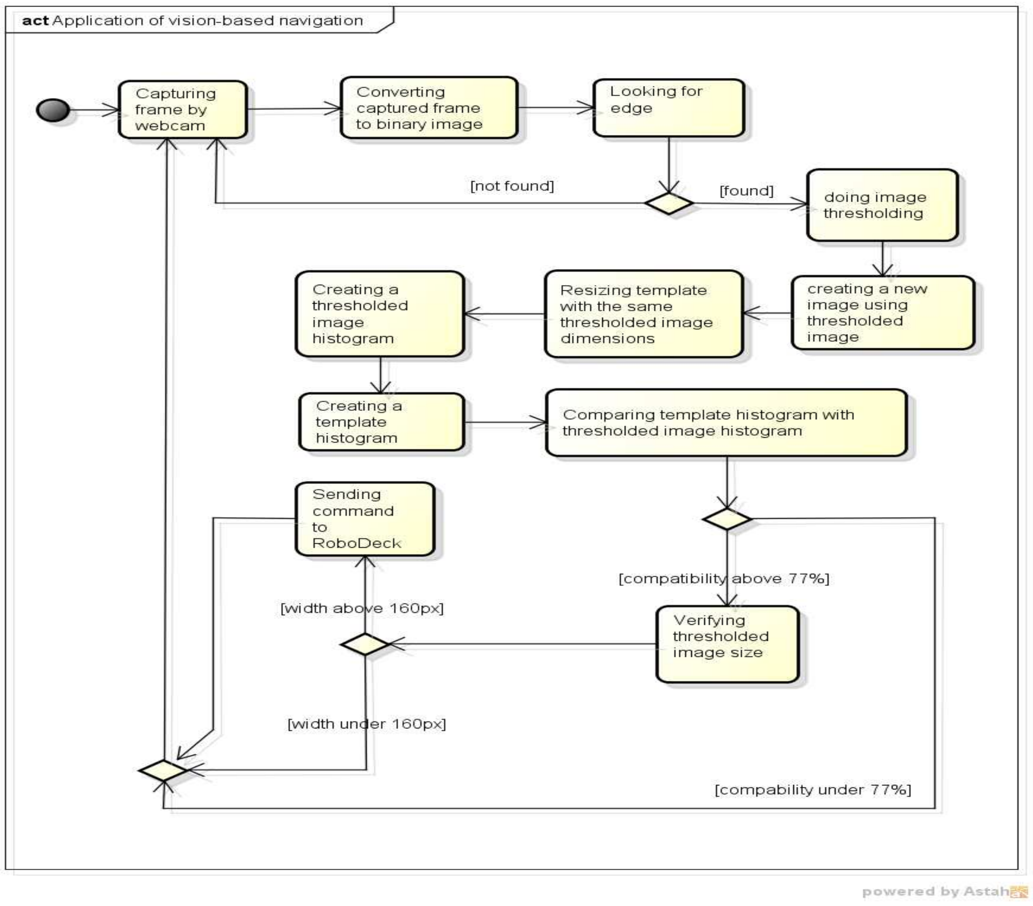

act Application of vision-based navigation
Capturing frame by webcam
Converting captured frame to binary image
Looking for edge
[not found]
[found]
doing image thresholding
creating a new image using thresholded image
Resizing template with the same thresholded image dimensions
Creating a thresholded image histogram
Creating a template histogram
Comparing template histogram with thresholded image histogram
Sending command to RoboDeck
[compatibility above 77%]
[width above 160px]
Verifying thresholded image size
[width under 160px]
[compability under 77%]
powered by Astah

**Figure 13. Activity diagram of vision-based algorithm for navigation**

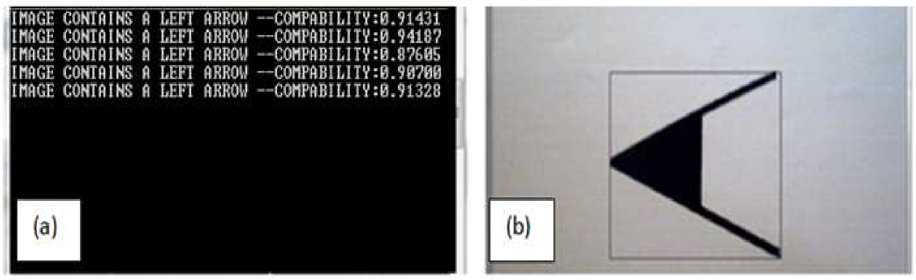
## 6. Performed Experiments

For reach the goal of developing vision-based navigation application, was necessary the developing of other applications, as well related experiments to that ones. Based on knowledge, results and analysis generated from that, was possible to create the vision-based navigation application [Orlandini 2012].

*Experiment 1: Indoor Environment Signal Recognition Application Tests*

The first experiment performed was related to indoor environment signal Recognition application tests. In a first moment, the application only recognized the signals left, right and stop, not sending commands to robot. The results of experiment running can be observed in Figure 14.

That experiment was performed aiming observe the signal recognition accuracy. During the application running, was placed an A4 paper containing arrow signal in front of the notebook webcam. The result was that, even modifying the distance between the paper and the webcam, the recognition was kept satisfactorily. It was performed also with right and stop signals. About the environment, was chose for the experiment a residential room, which had artificial lighting (fluorescent lamp) and windows that allows solar lighting entering. It was observed that a low lighting can change the results related to the signals recognition. It was realized so that the signals must receive a good light incidence to improve the recognition accuracy, however, the excessive incidence of direct light under the signal can impact in a not accurate recognition, it due to image saturation or specular reflex (A reflection in which light rays incident and reflected light rays form exactly the same angle with the surface of the mirror).



```
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.91431
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.94187
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.87605
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.90700
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.91328
```

**Figure 14. (a) terminal with replies given by the application. (b) captured image from camera (left arrow recognized)**

It was also important consider the quality print of signals, once those affects directly the results related to recognition. A performed test with a printed signal with low quality impacted in a low precision recognition. For perform this experiment, was used the following resources: Microcomputer with i7 processor and 8GB RAM; 1.3 Megapixel webcam; Visual Studio 2008 IDE and OpenCV 2.1 Library (both used for the application and development); and A4 Paper for signal printing. Is important to highlight that initially the code development took place in a Windows system using the Visual Studio 2008 IDE, considering the advantages offered by; meantime, for the experiments 3, 4, 5 and 6, after an initial development has been done in Visual Studio IDE, the codes related to the experiments were transferred for a Linux system. A few adjusts were need. Usually, it was associated to library including using the "include" command. Naturally, it was used the g++ compiler in the Linux system.

*<u>Experiment 2: Outdoor Environment Signal Recognition Application Tests</u>*

This experiment is similar to Experiment 1 but it was performed in external environment (outdoor). Observing the Figure 15, is possible see that a left signal was pasted on the wall. The results of the application running can be observed in the Figure 16.

It was realized the signals recognition kept had been done with satisfactory accuracy, such as for left signal as the right and stop signals. The mentioned experiment was performed only during the daylight. For the signals recognition occurs satisfactorily during the night, would be necessary to provide an adequate artificial lighting. This experiment attests that, with some adaptations, there are future possibilities of using this application in external environments. The used resource for running this experiment were the same outlined in the Experiment 1, however, for making the black arrow, was used a black cardboard. This one was cut in an arrow shape.

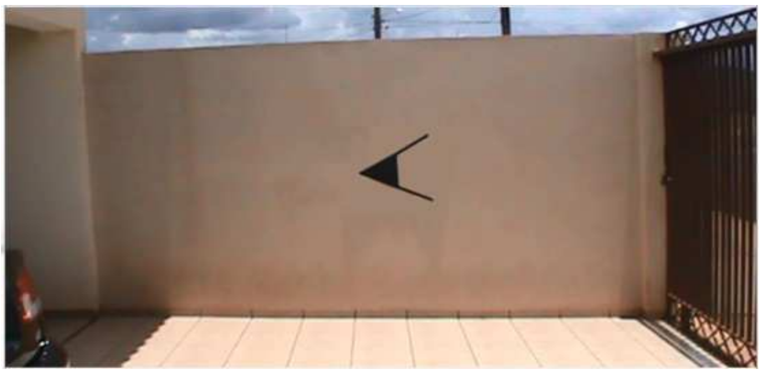
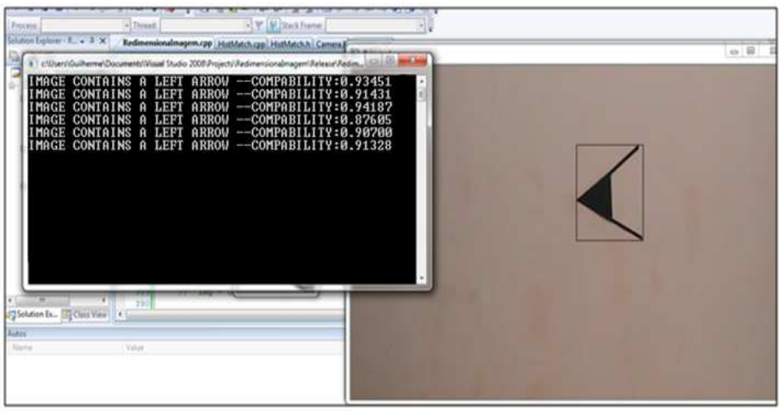


**Figure 15. Environment used in Experiment 2**



```
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.93451
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.91431
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.94187
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.87605
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.90700
IMAGE CONTAINS A LEFT ARROW --COMPABILITY:0.91328
```

**Figure 16. Results of the signal recognition application running**

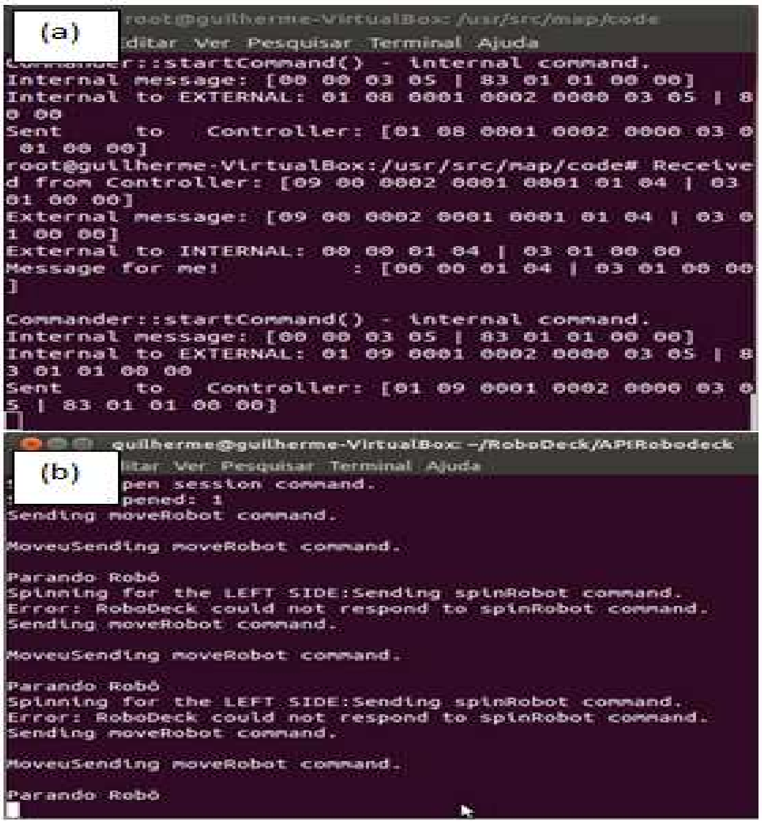
## _Experiment 3: Tests of sending robotic commands application using MAP in simulated mode_

Once developed a recognition signals application, was necessary based on recognized signals, to send commands to the robot. Analyzing RoboDeck´s manuals and documents, was realized that robotic commands although efficient, was done by low-level commands, fact that become the application development hard and prolonged, hence, was justified the creation of the API.

Both API and the sending robotic commands application could be tested and validated initially, even without a direct application in the RoboDeck, once the XBot Company offers the download of the MAP software. This one can be runs in a simulated mode, without integration with the robot. Hence, it was performed a MAP installation in Linux system, and done the tests. For code edition and compilation was used both the tools Nano and g++. During the experiments also was done installation and running tests of MAP in Ubuntu 10.04 and Ubuntu 11.04. The results were satisfactory.

The Figure 17 shows the working of sending robotic commands application and the replies provided by MAP running in simulated mode. In the Figure 17(a), there are replies generated by MAP running in simulated mode. The replies are given by hexadecimal code. The fact of MAP shows a reply in terminal does not guarantee the correct running by the robot but indicates the sent command by application it has been received correctly by MAP.

In the Figure 17(b), there are the replies given by MAP after the application has sent the robotic commands. Is possible to see messages in Portuguese language such as "Parando Robo" (in English language, "stopping robot"), "moveu" (in English language, "moved"), among others. Is also possible see in some lines the message "Error: RoboDeck could not respond to spinRobot command". This means that MAP received the command but cannot perform because it was running in a simulated mode, and then is not possible send commands to robot.


(a)
root@guilherme-VirtualBox: /usr/src/map/code
ditar Ver Pesquisar Terminal Ajuda
(b)
guilherme@guilherme-VirtualBox: ~/RoboDeck/APIRobodeck
litar Ver Pesquisar Terminal Ajuda

```
r::startCommand() - internal command.
Internal message: [00 00 03 05 | 83 01 01 00 00]
Internal to EXTERNAL: 01 08 0001 0002 0000 03 05 | 8
0 00
Sent      to   Controller: [01 08 0001 0002 0000 03 0
 01 00 00]
root@guilherme-VirtualBox:/usr/src/map/code# Receive
d from Controller: [09 00 0002 0001 0001 01 04 | 03
01 00 00]
External message: [09 00 0002 0001 0001 01 04 | 03 0
1 00 00]
External to INTERNAL: 00 00 01 04 | 03 01 00 00
Message for me!          : [00 00 01 04 | 03 01 00 00
]

Commander::startCommand() - internal command.
Internal message: [00 00 03 05 | 83 01 01 00 00]
Internal to EXTERNAL: 01 09 0001 0002 0000 03 05 | 8
3 01 01 00 00
Sent      to   Controller: [01 09 0001 0002 0000 03 0
5 | 83 01 01 00 00]
```

```
pen session command.
pened: 1
Sending moveRobot command.

MoveuSending moveRobot command.

Parando Robô
Spinning for the LEFT SIDE:Sending spinRobot command.
Error: RoboDeck could not respond to spinRobot command.
Sending moveRobot command.

MoveuSending moveRobot command.

Parando Robô
Spinning for the LEFT SIDE:Sending spinRobot command.
Error: RoboDeck could not respond to spinRobot command.
Sending moveRobot command.

MoveuSending moveRobot command.

Parando Robô
```

**Figure 17. (a) MAP running in simulated mode. (b) Running of application that do RoboDeck performs a determined path**

Those experiments also were done using mockup hardware (Figure 18). In these cases, the application was transferred for the board memory flash, and so performed. That board has a very similar hardware of RoboDeck, but without wheels and some sensors.

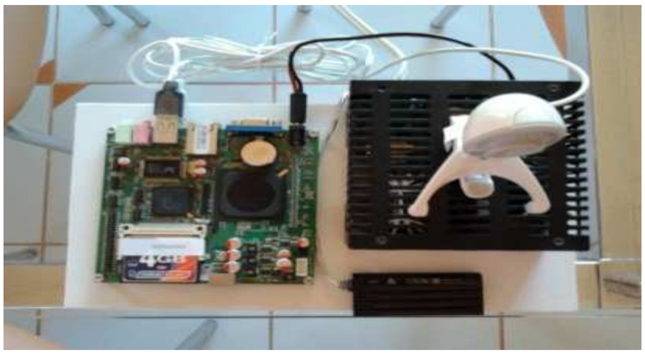


**Figure 18. Mockup hardware used to simulate RoboDeck responses**

<u>*Experiment 4: Tests of Sending robotic commands application to MAP on RoboDeck*</u>

Following the tests using MAP in simulated mode, where performed tests on RoboDeck. For that, it was necessary transfer the sending commands application for the high-level board of RoboDeck. Is important outline that the referred hardware has keyboard and monitor connection, fact that allows developers and researches to programming directly on RoboDeck, like a workstation (Figure 19).

The source-code related to sending robotic application was so transferred to RoboDeck´s high-performance board by USB port. The same could be edited and compiled directly in the robot. As RoboDeck has Linux Debian system offered in the high-performance board, tool to editing and compiling code was available for using.

In the Figure 20, there are pictures related to the sending of robotic commands by application running into RoboDeck. The robot performed the programmed route correctly. The environment where the experiment was done, it was the manufacturing sector of XBot Company. The floor was made by ceramics with some irregularities. Due the floor has a reasonably slick surface, in the moment that the actuation of motors occurred for beginning the movement, the wheels skidded lightly, fact that not affected the achievement of the course.

**Figure 19. Mockup hardware used to simulate RoboDeck responses**

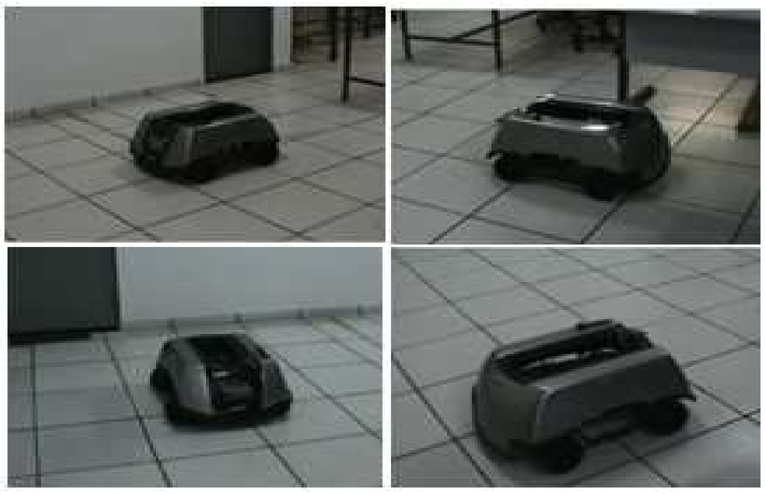


**Figure 20. Running the sending robotic commands application in high-performance board**

*Experiment 5: Recognition color application tests*

As computer vision application generally require a reasonable amount of processing by processor (mainly that one's related to image pattern recognition), so it was considered the possibility of developing a computer vision application that was not so computationally expensive for tolerate tests directly on RoboDeck, and the robot behavior was observed based on application running. Once the computer analysis relate to colors requires a smaller amount of processing when compared to image pattern recognition applications, was decided to develop a color recognition application.

The application was developed in a way where the green color would be an indication for the robot to move forward, and the black color would be an indication to stop. The color recognition application working occurs in a following way:

1. In the main() method, there is a loop. Inside the loop, the captured image by camera is read frame by frame.
2. Every 10 frames read, is done a frame analysis, searching for black color pixels or green color pixels. This analysis is done by VerificaCor() method.
3. The VerificarCor() method receives as parameter a frame, and do a "scanning" pixel to pixel. Each pixel is classified as green or black, based on a verification of RGB (Red-Green-Blue) properties. In this application, was considered a pixel as black when the red, green and blue level of pixels was less than twenty.
4. When the number of green pixels found in the frame reached 20% of the image, was considered that a green object was in the scene, hence, was sent to robot a command for moving forward.

Observation: It can see in the stage 2, the analysis related to colors identification been done each 10 read frames. This is due to trying not to overload the RoboDeck´s processor. The frame-to-frame analysis would become running more slowly. After the colors recognition application source-code compilation, was tried the first running inside the robot. In this first running, errors messages related to OpenCV library file missing were showed. It was realized so that the OpenCV library install would be necessary.

As the robot had a network interface connection, was possible to connect with Internet, fact that became easier the download and installation of packages that are pre-requisites for OpenCV installation. After solved that mentioned pending; was done the color recognition application running. As observed in the Figure 21, was placed a green card paper in front of RoboDeck. By identifying the green color, the robot began moving to forward. By place a black card paper in front of robot, the robot stopped. For having a better idea of used colors card papers, considering the RGB (The RGB pattern reproduces several colors through 3 basic colors: Red, Green and Blue) pattern. In this case: Green: R (19), G(56), B(30); and Black:  R (15), G(18), B(15);

It is also important outline that these values can change due the luminosity environment. In that environment where the experiment was performed, the luminosity was based on fluorescent lamps. The results related to tests done in this experiment can be observed at Table 1.

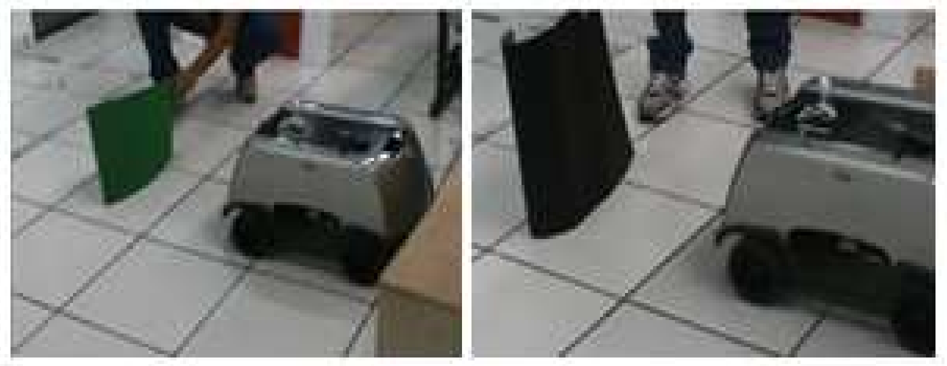


**Figure 21. Color recognition application running in RoboDeck**

**Table 1. Results related to black and green colors recognition**

| Color of Object | Distance between the object and the camera | Amount of Attempts | Amount of recognition done with success | RoboDeck´s action | Average percentage of pixels related to a particular color recognized during attempts |
|---|---|---|---|---|---|
| Green | 50cm | 10 | 9 | Moved | 85% |
| Green | 100cm | 10 | 8 | Moved | 74.4% |
| Black | 50cm | 10 | 10 | Stopped | 87.2% |
| black | 100cm | 10 | 10 | Stopped | 77.3% |

Still mentioning the Table 1, is important to have an explanation about the column "Average percentage of pixels related to a particular color recognized during attempts". In the first test, with a green object in a 50cm distance of RoboDeck, the percentage of green pixels found in the image was 81.3%. In the second attempt, the percentage of green pixels found was 87.4%, and so on until the tenth attempt, which impacted in an average of green pixel recognition by 85%, considering ten attempts. In observance of these results, was designed a chart showed in the Figures 22 and 23.

It was also done analysis related to consume both processor and RAM of robot when running the color recognition application. Was used for this the commands "ps gaux" and "free", this one's native from Linux and that respectively shows both processor and memory consumed. The outputs of this commands run directly under the Debian system of robot which can be observed in the Figure 24.

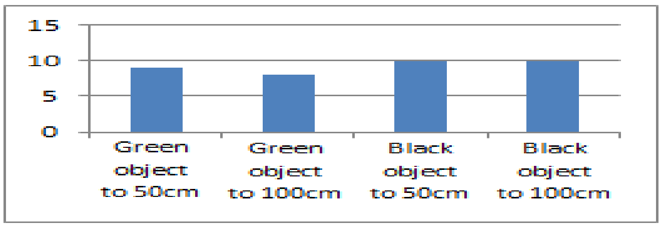


**Figure 22. Amount of times that the color of object was recognized**

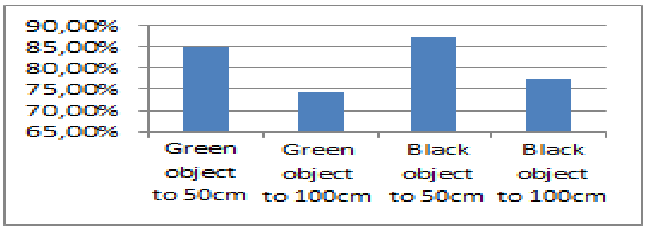
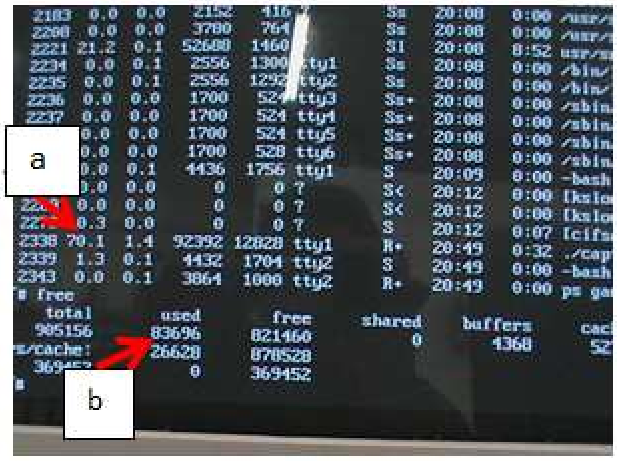


**Figure 23. Average percentage of pixels recognized relate to a determined color during attempts**

As it possible see in the Figure 25(a), during the color recognition application running, there were some peaks of processing consume reaching until 70% of processor capacity. In future experiments, this fact must be considered whether is observed some behaviors as slow or freezing of applications. Related to memory consume, is considered that remained low when considered the robot capacity (1GB). The Figure 25(b) shows the memory consumed after a few minutes of application running. During the monitoring, for few time, the consumer exceeded 84Mb.



```
2183  0.0  0.0   2152   416        Ss   20:08   0:00 /usr/
2208  0.0  0.0   3780   764        Ss   20:08   0:00 /usr/
2221 21.2  0.1  52688  1460        Sl   20:08   8:52 usr/
2234  0.0  0.1   2556  1300 tty1   Ss   20:08   0:00 /bin/
2235  0.0  0.1   2556  1292 tty2   Ss   20:08   0:00 /bin/
2236  0.0  0.0   1700   524 tty3   Ss+  20:08   0:00 /sbin
2237  0.0  0.0   1700   524 tty4   Ss+  20:08   0:00 /sbin
      0.0  0.0   1700   524 tty5   Ss+  20:08   0:00 /sbin
      0.0  0.0   1700   528 tty6   Ss+  20:08   0:00 /sbin
      0.0  0.1   4436  1756 tty1   S    20:09   0:00 -bash
       .0  0.0      0     0 ?      S<   20:12   0:00 [kslo
22    0.0  0.0      0     0 ?      S<   20:12   0:00 [kslo
22    0.3  0.0      0     0 ?      S    20:12   0:07 [cifs
2338 70.1  1.4  92392 12828 tty1   R+   20:49   0:32 ./cap
2339  1.3  0.1   4432  1704 tty2   S    20:49   0:00 -bash
2343  0.0  0.1   3864  1000 tty2   R+   20:49   0:00 ps ga
# free
     total       used       free     shared    buffers     cac
    905156      83696     821460          0       4368      52
s/cache:        26628     878528
    36945           0     369452
#
```

**Figure 24. Average percentage of pixels recognized relate to a determined color during attempts**

*Experiment 6: Vision-Based navigation application tests on RoboDeck*

After the experiments previously mentioned had been performed successfully, the next stage was to do the running experiment that would be the main experiment of this work, where the robot, running a computer vision application, would have autonomy for doing navigation in an environment. During this vision-based navigation application running directly on RoboDeck (in the high-performance board), was realized too much slowness in the response time. The application was demanding too much processing (more than the application described on Experiment 5), and the processor capacity was not being enough for answering that demand.

Hence, it was necessary delegate the task related to image processing and signal identification to an external computer. The Figure 25 shows how that architecture was planned for running this experiment.

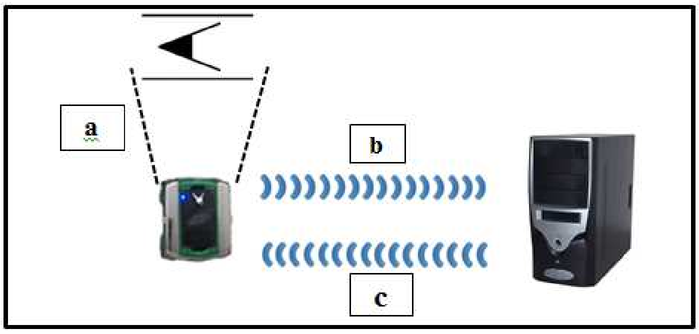


**Figure 25. Configuration planned for running the vision-based application on RoboDeck**

The working of that configuration showed in the Figure 26, occur in a following way:

1. An application running directly inside RoboDeck do the one capture image by webcam (Figure 26-a) and send the image for a microcomputer by wireless network (Figure 26-b). The image sent is based on an image writing in a shared directory by microcomputer, of a file related to the last image captured by RoboDeck.
2. In the microcomputer, the vision-based application kept running, performing readings and analyzing the file related to the captured image by RoboDeck. As mentioned previously, the file is in a shared directory, so the RoboDeck write the image in that shared directory, and the vision application located in microcomputer read the file.
3. When the application running in the microcomputer identifies the arrow signals to turn left, right or stop, the application send to RoboDeck a command to turn left, right or stop (Figure 26-c).
4. RoboDeck runs the command.

It was defined for both RoboDeck and microcomputer respectively the IP numbers 192.168.1.3 and 192.168.1.101. The configured wireless transmission rate between RoboDeck and microcomputer was 54Mbps. And was used the SAMBA software for directory sharing between RoboDeck and microcomputer.

The proposed configuration was set, tested and run the experiment. The image captured an application that was initialized on RoboDeck, and the signals recognition application was initialized on microcomputer. So an arrow sign to turn left was placed in front of the camera, the robot´s wheels turned to left. Once verified that the proposed configuration could work, was performed an environment preparation for doing actually the tests with the vision-based navigation application.

As is possible see in the Figure 26, in the way as the proposed environment for navigation was thought and mounted, the robot should turn left, in sequence turn right, and finally stop. For that experiment was used Styrofoam plates where the arrow signals were placed. To supports the Styrofoam plates, wooden boxes were used.

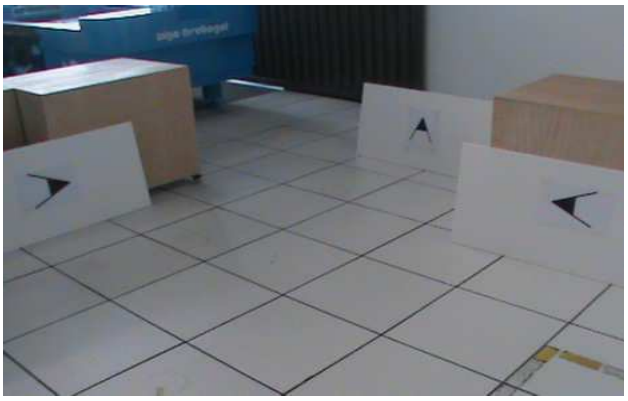


**Figure 26. Prepared environment for robot navigation**

It was done several attempts until the robot performed the course correctly. In the first attempts, sometimes the robot performed conversions to the left or right of way in advance just when sighted the arrow sign, so that the one did not move in towards the next Styrofoam plate. Hence, it was necessary adjustments in the vision-based application, for that robotic commands for direction changing or stopping was just sent when the arrow signal seeing had a width above 160 pixels by application. This did the robot only turn or stop after identifies an arrow signal at an approximate distance of 20cm. It was also necessary some adjusts of Styrofoam plate positioning.

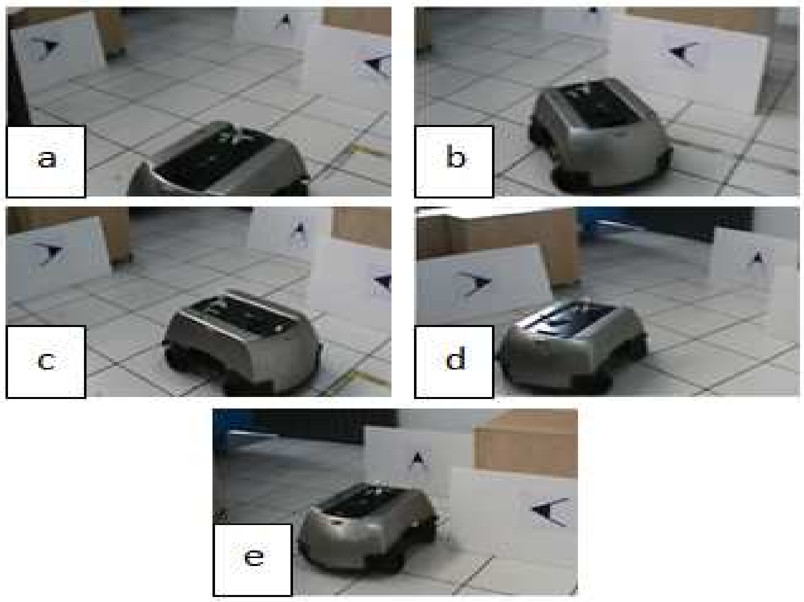


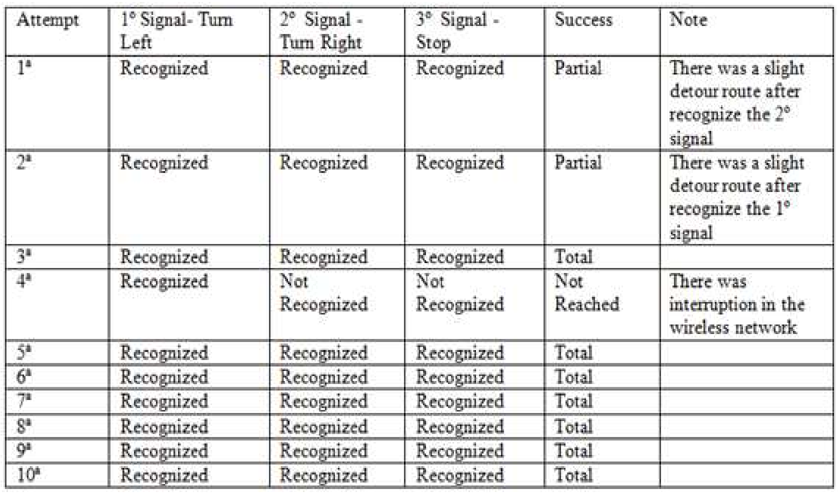
**Figure 27. Configuration planned for running the vision-based application on RoboDeck**

When the application began to run, the robot began moving forward (Figure 27-a). When the one identified the left arrow signal at an approximate distance of 20cm the robot turned left (Figure 27-b) and followed forward (Figure 27-c), approaching to the next signal (right arrow). When identified the right arrow signal, the robot turned right and followed forward (Figure 27-d). Finally, the robot identified the stop signal (Figure 27-e) and stopped. Hence, the robot performed the proposed course with success.

It was performed so more test so that the robot did the course. The results of these ones can be observed in the Table 2. For quantitative and quality results, the achievement of success related to route is given in table as "Not Reached", "Partial" or "Total", where:

- "Not Reached" - The course was not accomplished due to some fail during the one.
- "Partial" - The course was accomplished, but the robot presented a slight route detour in some moment (due to some floor characteristics or not use of robot alignment sensors), fact that did not affected the performing of course.
- "Total" - the course was done normally (successfully).

**Table 2. Tests results related to performing of course by RoboDeck**



| Attempt | 1° Signal- Turn Left | 2° Signal - Turn Right | 3° Signal - Stop | Success | Note |
|---|---|---|---|---|---|
| 1ª | Recognized | Recognized | Recognized | Partial | There was a slight detour route after recognize the 2° signal |
| 2ª | Recognized | Recognized | Recognized | Partial | There was a slight detour route after recognize the 1° signal |
| 3ª | Recognized | Recognized | Recognized | Total | |
| 4ª | Recognized | Not Recognized | Not Recognized | Not Reached | There was interruption in the wireless network |
| 5ª | Recognized | Recognized | Recognized | Total | |
| 6ª | Recognized | Recognized | Recognized | Total | |
| 7ª | Recognized | Recognized | Recognized | Total | |
| 8ª | Recognized | Recognized | Recognized | Total | |
| 9ª | Recognized | Recognized | Recognized | Total | |
| 10ª | Recognized | Recognized | Recognized | Total | |

As registered in the Table 2, ten attempts were done related to performing by RoboDeck about the planned course. Both first and second attempts, it occurred slight route detour during the navigation in some moments (although RoboDeck had done correctly the course). In the first attempt, for instance, after the RoboDeck identified the right signal to turn, the one turned right, and began a rectilinear route in towards the stop signal. Before finish totally the rectilinear route, the robot´s wheels inclined slight (about three degrees) for right, doing the route did not finish totally in straight.

Relate to the fourth attempt registered in the Table 2, the course did not perform successfully due to a momentary interruption of wireless signal between RoboDeck and the microcomputer that it was processing the images (this configuration can be observed in the Figure 26). This factor contributed for a not recognition of the signal by robot.

Still observing the outlined results in the Table 2, was realized that seven of ten attempts occurred with "Total" success. In two of ten attempts was obtained "Partial" success, and in only one of the attempts, the success was "Not Reached". In compliance of these results, was generated a chart contained in the Figure 28, which once analyzed, can show a considerable success rate. In the Figure 29 there is the chart related to the amount of times the recognition of each signal was reached with success and the action was done by RoboDeck. Is presented also the amount of times the course occurred with "Total" success. There, is showed that the left signal was recognized more than the right and stop signals. This difference occurred due the light hitting different angles of the signals. The success was not obtained all the times due some slight detour during the course and interruption in the wireless signal (as mentioned before).

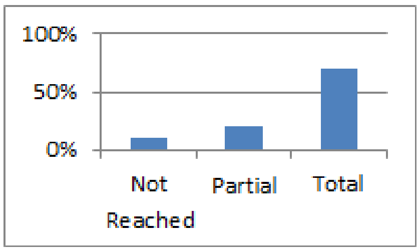


**Figure 28. Percentage related to success of course performing**

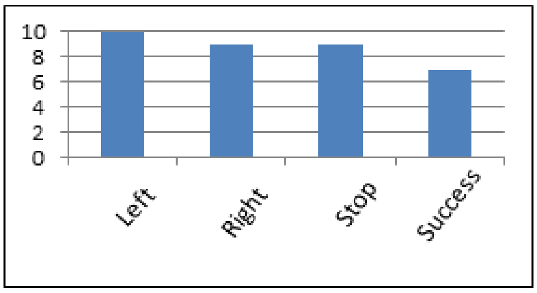


**Figure 29. Hits related to the recognition of each signal and running of the action, and amount of running with "Total" success**

## 7. Conclusion

Based on experienced obtained during the development of application using the OpenCV Library, was realized that that one offers really important advantages for the developing of computer vision applications. Is possible to say that this library has potential for leverage research in the computer vision field, because besides being a rich library with too many functions, it one is free and open source.

The RoboDeck hardware was showed satisfactory related to resource available by this one, showing itself adequate for using OpenCV in development of robotic applications. Those resources allow performing infinity of researches in mobile robot, factor that consequently can increase significantly the amount and quality of Brazilian´s research related to the mentioned subject.

Due the fact of being a relatively new hardware in the market, the RoboDeck still needs of computer vision applications, fact that motivated the developing of this work. It hoped this work can be used as a support material for researchers that have interests in developing computer vision applications for RoboDeck or other robotic hardware, because the developed applications can be easily adjusted for other hardware that support the use of OpenCV.

The aim of developing a based-vision navigation application that provides for the robot a determined degree of autonomy during the navigation in an environment had been accomplished. Naturally, improvements can be performed in future works so that the navigation is done with better accuracy and speed. Several applications can be developed based on algorithms created in this work, for instance, signal recognition applications for traffic and semaphore for implementing outdoor autonomous vehicles, autonomous wheelchair, convoy of autonomous vehicles(for helping people in airports, conventions, museums, etc.), surveillance systems based on autonomous robots (recognition of doors, windows, gates), systems for inspection( from the definition of object for being inspected, for instance, aerial vehicles doing inspections in plantations, transmission lines for electricity, phone towers, etc.).

There are several impediments that render the many approaches to autonomous navigation useless during real time applications. The frequent problem encountered during real-time navigation is those pertaining to anomalies in the environment that deviate them from the typical model of the environment. Obstacle in the environment disrupts the estimates of the robot's location in case of passive navigation techniques such as optical flow or feature tracking. On the other hand, wheel odometer-based navigation systems suffer from wheel-slip situations resulting in location estimates that are far from the robot's true location. The wheel slip situation often happens when the friction in the wheel ground interface in not enough to counter the torque from the wheels rotation. An erroneous measurement of wheel rotation is recorded via the wheel encoders resulting in a faulty robot location estimate.

The movement of persons usually is inevitable in the environment that the robot is present. Errors in the camera system due to digitization process or defects in the camera construction during manufacturing such as lens distortions and improper positioning of the image sensor also prove to be problematic during image acquisition step. These effects trickle down the motion estimation pipeline and eventually corrupt the localization output. On the other hand, the vision-based navigation systems can fall prey to unavailability of proper visual cues in the environment. A key feature based on tracking approach often suffers in environments with uniformly colored surfaces that often have lack in texture. Presence of good amount of texture is essential in feature-tracking-based approaches to navigation. Presence of clutter in the navigating environment such as bushes and other foliage often confuse vision systems in the case of outdoor environments. Outdoor environments also face the problem of lighting variation as the robot moves from one place to another. In addition to this the effect of shadows cannot be ignored as well. Proper localization of the robot within its environment is one of the major challenges in autonomous navigation.

The RoboDeck API for communication with MAP, designed and developed (partially) in this work, can provide facilities for other researchers related to the application development for RoboDeck. As future works, its recommended the following approaches: Optimization of vision algorithms presented in this work so that consume less computational resources and it can be run directly on robot with a satisfactory response time; Adding lighting filters in the vision algorithms; Using of other image patterns recognition techniques, so that improves the recognition accuracy; Integration of the vision-based navigation application with reading commands of ultrasonic sensors and GPS, so that obtains a better accuracy related to robot localization in an environment; Continuity of RoboDeck´s API development for communication with MAP; and Development of collaborative applications, involving two or more robot acting in a jointly way.

## Acknowledgments

Thanks to CNPq for their program RHAE and DT (Productivity in Technological Development and Innovative Extension).

## References

Adept, MobilerRobots (2011) “Specification Manual Pionner 3 – AT”. http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT-RevA.sflb.ashx [21 may 2011].

Andresen, C.; Jones, S. D.; Crowley J. L. (1997) Appearance Based Processes for Visual Navigation. Symposium on Robotic Systems – SIRS 97, 227-236.

Bradisk, Gary; Kaehler, Adrian (2008). Learning OpenCV – Computer Vision With OpenCV Library. 1ªed. O´Reilly Media, pp. 2-4.

Brooks, R.A. (1991). New approaches to robotics. In Science, Vol. 253, 1227-1232.

Budiharto, W. (2014) Modern Robotics with OpenCV. Sci Pub. Group.

Chaczko, Zenon; Braun, Robin (2010) Teaching computer vision for telemedicine systems using OpenCV. In ITHET'10 Proceedings of the 9th international conference on Information technology based higher education and training.

Chang, Wen-Chung; Chuang, Chun-Yi, (2011) "Vision-based robot navigation and map building using active laser projection," System Integration (SII), IEEE/SICE International Symposium, 2011; 24, 29, pp. 20-22.

Chen, XiaoQi; Chen, Y.Q.; Chase, J.G. (2009) Mobile Robots – State of the Art in Land, Sea, Air, and Collaborative Missions, In-Tech.

Chen, Haoyao; Sun, Dong; Yang, Jie; Chen, Jian, (2010) "Localization for Multirobot Formations in Indoor Environment," Mechatronics, IEEE/ASME Transactions, pp. 561-574.

Ching, Yong Kok; Prabuwono, Anton Satria; Sulaiman, Riza. (2009) Visitor Face Tracking System Using OpenCV Library. In Proceedings of IEEE Student Conference on Research and Development.

Choset, H. M., (2005) Principles of robot motion: theory, algorithms, and implementation. MIT press.

Culjak, I. et al. (2012). A brief introduction to OpenCV. In: MIPRO, 2012 Proceedings of the 35th Intern. Convention. IEEE, pp. 1725-1730.

Desouza, Guilherme N.; Kak, Avinash C. (2002) Vision for Mobile Robot Navigation: A Survey. IEEE Trans. Pattern Anal. Mach. Intell. pp. 237-267.

Dawson-howe, K. (2014). A practical introduction to computer vision with OpenCV. John Wiley & Sons.

Fernández-caramés, C. et al. (2014) A real-time door detection system for domestic robotic navigation. Journal of Intelligent & Robotic Systems, v. 76, n. 1, pp. 119-136.

Guzel, M.S. (2009). Mobile robot navigation using a vision based approach. In proceedings Newcastle University Postgraduate Conference.

IROBOT. iRobot (2011) Create Owner's guide. http://www.irobot.com/ filelibrary/pdfs/hrd/create/Create Manual_Final.pdf [23 may 2011].

Jahne, Bernd; Hausseker Horst (2000) Computer Vision and Applications: A Guide for Students and Practitioners. Acad. Press.

Johns, E.; Guang-zhong, Y. (2010) "Scene association for mobile robot navigation," Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference, 4698,4703, 18-22.

Jones, S.D.; Andresen, C.; Crowley, J. L. (1997) Appearance Based Processes for Visual Navigation. In Proc.of IEEE Int'l Conf. on Intelligent Robots and Systems (IROS), pp. 551–557.

Kundur, S.; Raviv, D. A. (1998) Vision-Based Pragmatic Strategy for Autonomous Navigation. Pattern Recognition, Elsevier Science, Vol. 31, N. 9, pp. 1221-1239.

Hongming, Z. et al. (2005) Object detection using spatial histogram features, Image and Vision Computing 24, pp. 327-341.

LEGO (2012) Lego Mindstorms User Guide. http://cache.lego.com/ upload/contentTemplating/Mindstorms2BuildingInstructions/otherfiles/downloadA0 B7A698E231D3E619C43ECCFFE2F27F.pdf [may 2012].

Liang, S. N. et al. (2016) Open source hardware and software platform for robotics and artificial intelligence applications. In: IOP Conference Series: Materials Science and Eng. IOP Publishing, pp. 012142.

Linder, T.; Arras, K. O. (2016) People Detection, Tracking and Visualization Using ROS on a Mobile Service Robot. In: Robot Operating System (ROS). Springer Publishing, pp. 187-213.

Lopes, B. S.; et al. (2010) Fath, Vinicius.; Silva, Marcelo Q.; Assis, Wanderson, O.; Coelho, Alessandra D.; Gomes, Marcelo M., A Robotic Classifier System Including Computer Vision. LARC - Latin American Robotics competition, FEI, Brazil.

Matsumoto, S.; Ito, Y. (1995) Real-time Based Tracking of Submarine Cables for AUV/ROV. In Proc. of IEEE Oceans, pp. 1997–2002.

Mcginn, C. et al. (2015) Towards an embodied system-level architecture for mobile robots. In: Advanced Robotics (ICAR), 2015 International Conference on. IEEE, pp. 536-542.

Mishra, P. et al. (2013) "Robust template matching based obstacle tracking for autonomous rovers," Electronics, Computing and Communication Technologies (CONECCT), 2013 IEEE International Conference, pp. 1-5, 17-19.

Moravec, H. (1980) Obstacle Avoidance and Navigation in the Real World by a seeing robot Rover, PhD thesis, Stanford University.

Muñoz, S M. (2011) Projeto do software do RoboDeck. Versão 1.0. São Carlos.

Nehmzow, Ulrich (2003) Mobile robotics - A practical introduction. 2nd edition, Springer.

Ohno, T.; Ohya, A.; Yuta, S. (1996) Autonomous Navigation for Mobile Robots Referring Pre-Recorded Image Sequence. In Proc. of IEEE Int´l Conf. on Intelligent Robots and Systems (IROS), pp. 672–679.

Orlandini, G. (2012) Desenvolvimento de aplicativos baseados em técnicas de visão computacional para robô móvel autônomo, Mestrado, Universidade Metodista de Piracicaba, p. 95.

Quigley, M. et al. (2009) ROS: an open-source Robot Operating System. In: ICRA workshop on open source software, pp. 5.

Rivera-bautista, J. A. et al. (2012) Using color histograms and range data to track trajectories of moving people from a mobile robot platform. In: Electrical Communications and Computers (CONIELECOMP), 22nd International Conference IEEE, pp. 288-293.

Russ, J. C. (2011) The Image Processing Handbook. Sixth Edition. CRC Press.

Russel, S.; Norvig, P. (1995) Artificial Intelligence: a modern approach. Prentice-Hall, New Jersey.

Saunders, C. et al. (2015) Computer Vision Techniques for Autonomic Collaboration between Mobile Robots.

Siegwart, R.; Nourbakhsh, R. Illah (2004) Introduction to Autonomous Mobile Robots, The Mit Press.

Shapiro, L.; Stockman, G. (2001) Comp. Vision. 1ªed. Prent Hall.

Shrivastava, R. (2013) A hidden Markov model based dynamic hand gesture recognition system using OpenCV. In: Advance Computing Conference (IACC), 2013 IEEE 3rd Inter. IEEE, pp. 947-950.

Suarez, O. D. et al. (2014) OpenCV Essentials. Packt Publishing Ltd.

Szabo, R.; Gontean, A. (2015) Robotic arm detection in space with image recognition made in Linux with the Hough circles method. In: Computer Science and Information Systems (FedCSIS), Federated Conference on. IEEE, pp. 895-900.

Thrun, S.; Bucken, A. (1996) Integrating grid-based and topological maps for mobile robot navigation. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, Portland, pp. 1-7.

Tomatis, N.; et al. (2001) A Hybrid Approach for Robust and Precise Mobile Robot Navigation with Compact Environment Modeling. In Proceedings of IEEE, International Conference on robotics & Automation, Seoul, Korea.

Ukida, H.; et al. (2012) "Object tracking system by adaptive pan-tilt-zoom cameras and arm robot," SICE Annual Conference (SICE), 2012 Proceedings, pp. 1920 -1925.

Valasek, J. et al. (2005) Vision-Based Sensor and Navigation System for Autonomous Air refueling. Journal Guidance, Control and Dynamics.

[48] XBOT Company. Manual do usuário RoboDeck. Versão 1.0. São Carlos, 2011. www.xbot.com.br [10 dec. 2015].

Xie, G.; Lu, W. (2013) Image Edge Detection Based On OpenCV. International Journal of Electronics and Electrical Eng., v. 1, n. 2, pp. 104-6.

Yang, Y. et al. (2010) Realization for Chinese vehicle license plate recognition based on computer vision and fuzzy neural network", Proc. SPIE 7749, 2010 International Conference on Display and Photonics.

Yeoun-Jae, K. et al. (2011) "Vision-based direction determination of a mobile robot in indoor environment," Digital Ecosystems and Technologies Conference (DEST), Proceedings of the 5th IEEE International Conference, pp. 153-157.

Zhao, G., Jia, T. (2009) "Construction of vision-based manipulation system for 3D industrial objects, "IEEE International Conference on Robotics and Biomimetics, 2009; 1051-1056.