Computer Vision applied to autonomous robots using Raspberry Pi

Daniel Jimenez Mendoza
Universitat Autonoma de Barcelona

Index Terms—Computer vision, Robotics, Remote sensing, Motion, Image Processing, Raspberry Pi

Abstract—The aim of this project is to discover new techniques to combine software and hardware components to be applied on an autonomous robot. The appearance of new cheap computational modules like Raspberry Pi and high-resolution cheap cameras makes it possible to combine both, applied to computer vision. This purpose has been achieved by exploring ways of object detection and navigation control using computer vision, low-cost computing devices and sending the path wirelessly via fast wireless network communication. This would favor the adoption of new low-cost autonomous robots in peoples everyday life. The system that I purpose in this paper is based in 3 hardware devices; one actuated robot with a Raspberry Pi, another equipped with another Raspberry Pi and a web camera and one with a smartphone running a user control app. The result is a fully working navigation system. This project is developed under the context of an educational final project and aimed also to reuse the implementation as a practicum material for a future use of the students.

1 Introduction

Throughout the 20th century the use of autonomous robots has been increasing. These devices attempt to analyze the environment and interact with it to achieve a concrete action usually assigned by a human. To modelize and digitalize the environment, they are equipped with different types of sensors, depending on the use. Computer vision, on the other hand, tries to analyze the environment via image processing using only image capturing. The increase of processing power and the appearance of new technologies are making it possible for this technique to be applied to autonomous robots.

These robots are making headway today with the appearance, for example, of autonomous vehicles, drones or security systems which implement advanced computer vision to improve peoples lives. The army and other big companies have already announced their interest in this technology with the intention of using it in their missions and in the field of long-term research as well.

Given the high price of the most common sensors such as radar, IR, laser etc., the aim of this project is to create an autonomous robot using computer vision as the only sensor of the system. The high cost of these sensors and devices currently allows only very few organizations, agencies and institutions, to access them, hindering the investigation and delaying the incorporation of these into society. The appearance of low-cost computational models such Raspberry Pi is perfect for this kind of projects. The combination of hardware and software of this device is designed to be adaptive and really easy to use. Using this device, the production cost would be cheaper, causing them to be manufactured massively. This would end in devices able to improve the human-machine interaction using artificial intelligence with, for example, guiding robots for blind people or managing the traffic of autonomous vehicles in city intersections. The action robot will be sensorless itself and to cover the absence of traditional sensors, there will be an independent vision module located in the scene that will be the responsible one for calculating, coordinating and submitting the movements of the robots.

The Computer Vision Center (CVC) of the Universitat Autonoma de Barcelona is in charge of research projects related to robotics, computer vision and computational field, in general. This project is part of a final degree project, and it is designed to be highly modular, to be reused as a practicum support material for the subject of Robotics, of the engineering school at the UAB. Particularly, this project consist of three hardware modules (a robot, a image capturing device and smartphone) and four software modules.

This is the structure of the document; Firstly I am going to start explaining the system architecture, the work diagram, the methodology of the project I used and the explanation of the different modules involved. Then I will describe the results and conclusions. Lastly I will end with the acknowledgments and bibliography.

2 METHODOLOGY

2.1 System Architecture

As we said before, this autonomous navigation system is able to generate a route between two points (P_{start} and P_{goal}). The figure **??** shows the system scheme.

For the realization of this project three devices will be used; the first one (Pluto from now on) will be equipped with only two wheels (provided by CVC) and a Raspberry Pi. The second one (Zeus from now on) its equipped with a webcam and other Raspberry Pi. The third device will be an Android application to control the entire system.

These hardware devices will work together in this way: Zeus will capture the image of the scene, will trace a route from P_{start} to P_{goal} using computer vision and will be sent to Pluto wirelessly. Ive decided to do it this way because of the scalability of the entire system: a second robot can be added without modifying the entire scheme, for example.

To implement this scheme I purpose the creation of four software modules:

- **Action Module:** This module will be responsable for managing the control actuators of Pluto (its wheels and odometry).
- **Vision and computing module:** This module will be responsable to capture the scence, analyce it by detecting the obstacles using image processing and calculate the path.
- **Communication module:** This module will be responsable to communicate, wirelessly, the 3 devices together. This project will be an offline system. It means that after capturing the image and processing it once, Zeus ceases to have any practical effect in the system.
- **User Control module:** This module will be responsable to perform the entire control of the system. It will be achieved by an Android app connected to Zeus.

Due the modularity of the system, I could work independently over the modules without coupling risk. This made the development faster. There is another final step or module. It is to check every module independently to check their proper operation and then connect them together and ensure the proper operation

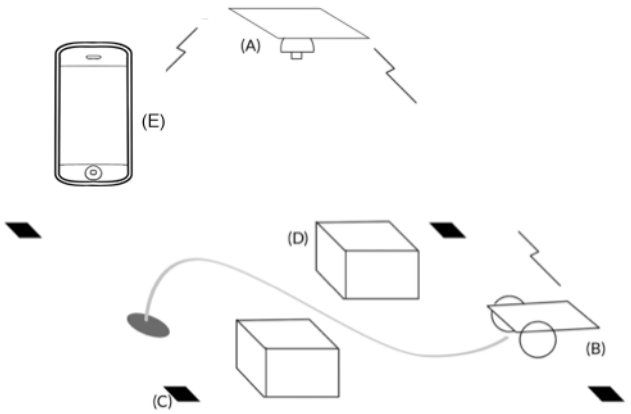

Fig. 1. General system squeme: (A) Zeus, equipped with a camera; (B) Pluto, connected to Zeus wirelessly; (C) Markers to define the workspace; (D) Obstacles to avoid; (E) Android device.

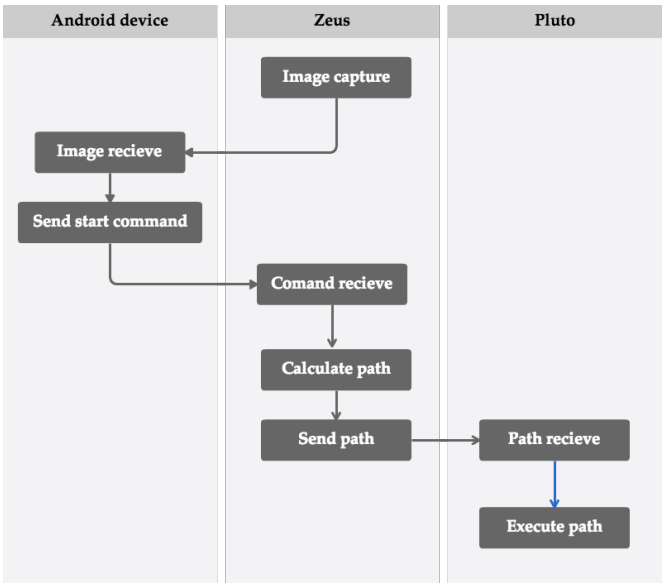

Fig. 2. Flowchart between Android device, Zeus and Pluto

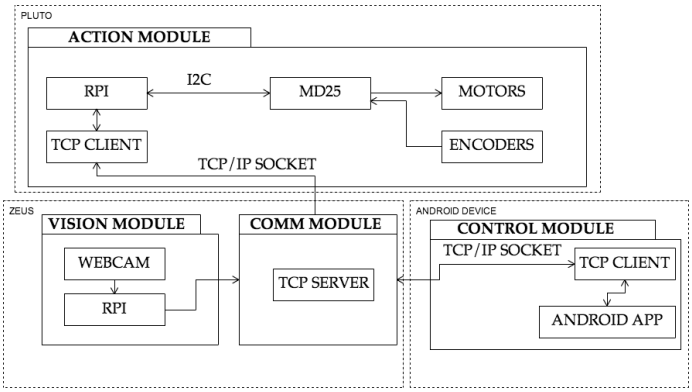

Fig. 3. Main diagram of the system

of the whole system.

2.2 Action Module

This module is the responsible one for giving movement to the autonomous robot. Its an embedded hardware system which corresponds with figure **??**. It comprises an MD25 kit [2], which includes two wheels, encoders and a board to process all the odometry. It is also equipped with a Raspberry Pi to make the wireless communication with Zeus possible. Due to the reduced size of all the components, I installed all of them into a Tupperware (making it a tupperbot), resulting in a lightweight robot. I added a third non-actuated wheel at

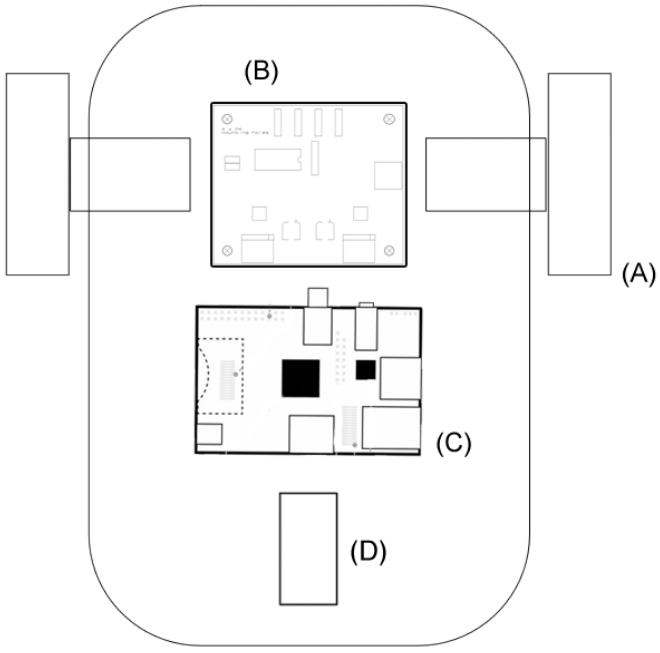

Fig. 4. Pluto diagram: (A) Wheels; (B) MD25 Wheels controller board; (C) Raspberry Pi; (D) non-actuated wheel.

the center to give more stability to the robot. Both wheels can go forwards or backwards independently, and the robot can rotate about its origin. This allows the robot to make a piecewise linear path.

2.2.1 Sending commands to the wheels

The MD25 board is responsible to manage the control of the motors. The board gives the option to select which type of communication with the wheel we can have: I2C or Serial, so I selected the simplest one, I2C. This digital communication is very common in this kind of devices. The advantages of I2C over Serial are that it is faster and it can receive or send data to specific devices if you have connected more than one. In order to communicate the Raspberry Pi with the MD25 control board I need to connect them as shown in the figure **??**.

I started using the Raspberrys GPIO ports (the input and output pins of the Raspberry). I found a library [7] by Adafruit (the same distributors of Raspberry Pi) to work with I2C

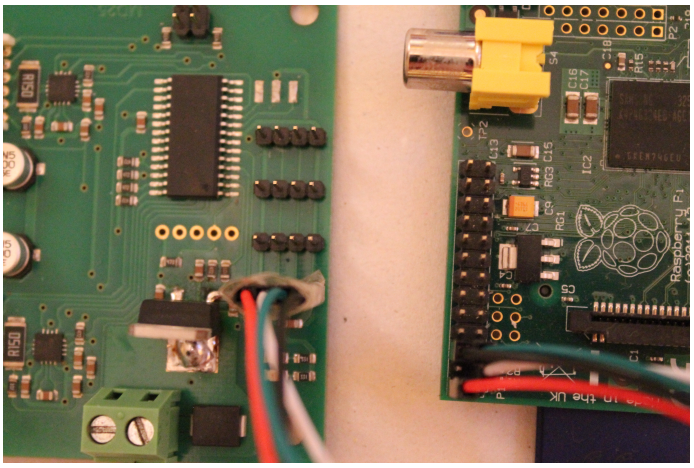

Fig. 5. Connection between Raspberry Pi and MD25 board.

protocol, so it was not difficult to have them working together. I had to calibrate the encoders to make the robot go forward and turn exactly as I wanted. To do that, first I coded the robot to go forward a determinate value. Then I measured how much the robot has moved. This gave me a ratio that now I can apply if I want the robot go forward a specific distance. This calibration depends on the physical features of the robot (distance and diameter between the wheels, mainly) but it can be extrapolated to robots with similar characteristics. For this reason, I have made a Python library (using Adafruits I2C library also) to control MD25 board using a Raspberry Pi.

2.3 Vision Module

This module is completely embedded in Zeus and it consist in a compatible Raspberry Pi webcam and a Raspberry Pi device. Much of the processing of the system lies in this module. It is responsible for interpreting the scene using an image, detecting the obstacles and generating a path from P_{start} to P_{goal}. This module can also be decomposed into a few sub modules:

- **Image Acquisition:** This will capture the image of the scene
- **Object Detection:** Will be the responsable for detect the obstacles in the scene.
- **Markers Detection:** Will be the responsable for detect the black markers on the ground and establish a work area.

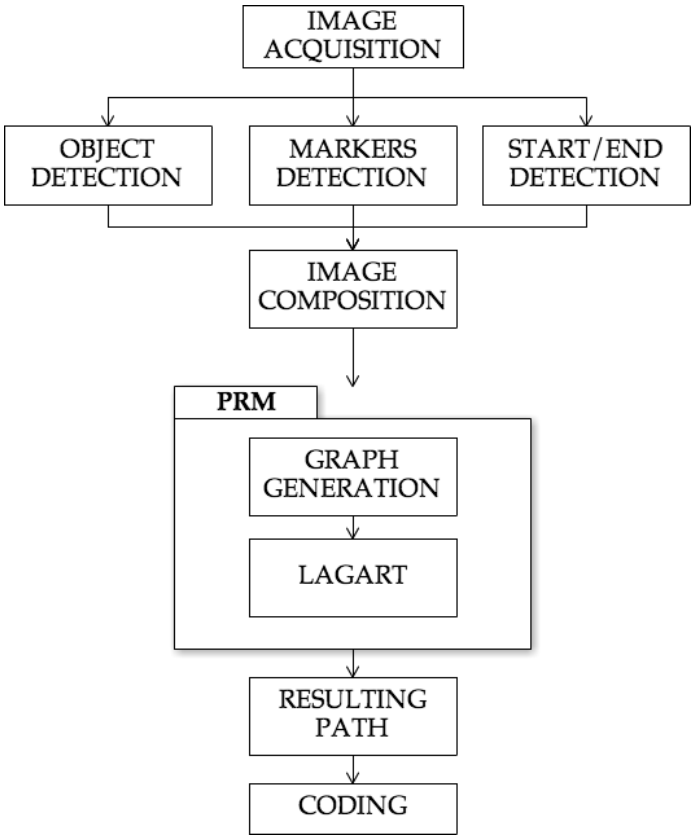

Fig. 6. Detailed diagram of the vision module

- P_{start} **and** P_{goal} **Detection** Will be the responsable for detect the start and ending point of the path in the scnere.
- **Image Composition** After all the detections, this module will combine them all to set up a workspace.
- **Path-Finding** This module will be the responsable for trace a path from point A to point B.
- **Encoding Module** This module will encode the path resulting from PRM to send it to Pluto.

2.3.1 Object Detection

I only consider the gray level image. This reduces the number of layers to analyze without losing a lot of information. Because of this reason, the background is white and the foreground objects are black. We apply a threshold to the image and improve the result. The threshold level was chosen by testing different values and selecting the one which provided

the best result. After a first threshold processing, there were remaining artifacts (elements which werent obstacles but still remain in the image). Mainly there were the wheels of the robot (which were also black) and scene furniture. To delete them I used two image processing techniques called opening and closing [**?**]. The opening is a morphology operation which reduces the noise of the image given a structuring element. It removes small objects from the foreground by placing them in the background. The result of the opening is the same image but without the artifacts, but it makes the objects smaller. Closing is another morphology operation which, in this case, restores the original size of the obstacles. By doing this, I was able to filter only the obstacles to avoid. To count the physical features of the robot and to prevent Pluto from crashing with obstacles when it executes the path, I had to use another image processing technique called dilation. This caused the obstacles detected to seem bigger than the real obstacles. With this operation, I am defining Qfree space (all available configurations of the robot).To define how much the object must be oversized, I must know the physical dimensions of the robot. Measuring it, the minimum radius is 15cm, so, the objects must be oversized 15cm. To represent this 15cm in pixels I used the ratio calculated when detecting the markers (explained in the next subsection). Then, we only need to use this ratio to apply it to the dilation kernel.

2.3.2 Markers Detection

To determine the area of the robot operation, I used two black square (2x2cm) cardboard markers located in opposite corners. To recognize them, I used a technique called template matching [**?**]. Giving a marker template to the function, it returns all the x, y points where it detects the template. Usually this function returns a lot of points, (not only 2) and I had to use a clustering method to clean it up. I chose to use a simple k-means algorithm I found on the internet. After checking it, I had the two correct points. Now we add the diagonal distance between the markers by code and calculate the cm/px ratio.

2.3.3 Points Detection

To determine the starting point and end point of the scene I used black markers with different shapes. The starting point is a black triangle located on the top of the robot and the end marker is a circle located on the floor. I used the same method as explained before (template matching). The only thing I had to change is the template used. I transferred the image of a triangle/circle to the function, and the function returned the starting or ending point.

2.3.4 Image Composition

After detecting all the single parts in the scene I had to combine them into one single image. I set the black areas as working space and the white areas as obstacles. As you can see in the figure [10], only the black area represents where the robot can move.

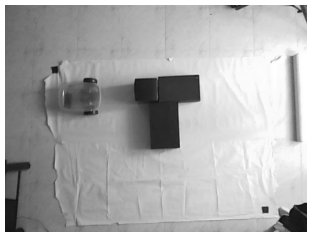

Fig. 7. Raw image

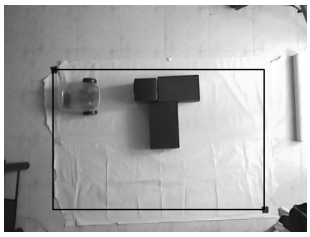

Fig. 8. Limit detection

Fig. 9. Object detection

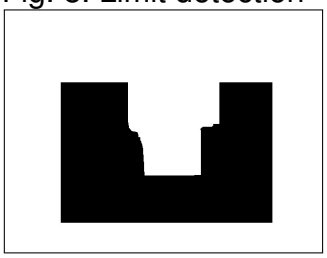

Fig. 10. Final composition

2.3.5 Path-finding

Once I had the working space, I had to calculate the path from point A to point B. One way to achieve it is to scale down the image to a 50x30 pixels image and then apply an Astar algorithm and then rescale the image again to its original size. This ends with a low-resolution path, and there were problems when I tried to accommodate it into physical context due the pixelation of the path. Also, every time we want to calculate a new path, we have to

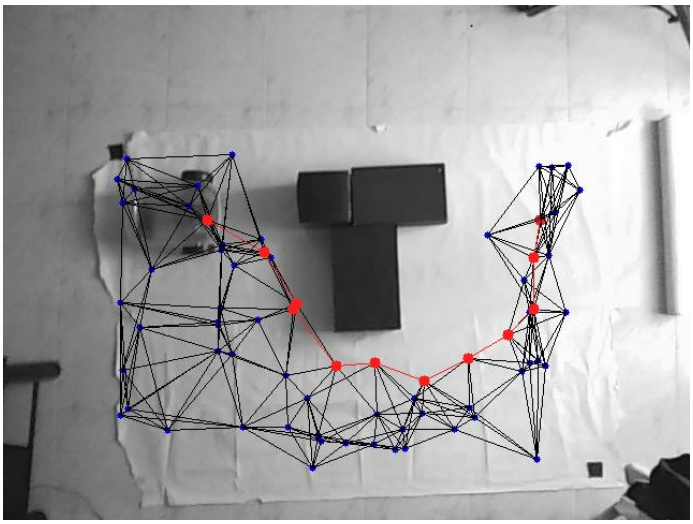

Fig. 11. Path generated by PRM using Lagart algorithm.

execute the entire algorithm. Another way to accomplish that is to use a PRM algorithm. This consists in two phases; the first one (training) is to spread random points and connect them building a graph. The second phase is to apply a path-finding algorithm between two nodes of the graph (from the node which represents the point A and the node which represents the point B). With this method the initial computing cost is higher but once you have the graph, you can recalculate as many paths as you want with a lower cost.

2.3.6 Encoding Module

In this context, we can define a route as a set of actions to go from P_{start} to P_{goal} using the nodes obtained in PRM module. Coding this path to be sent to Pluto is not an easy thing. Mainly because the path calculated by Zeus is in 2D (x, y), while Pluto lives in 3D (x axis, y axis, theta). We can simplify the route by dividing it into a series of segmented combinations of turning and moving. This way, we can represent the Pluto trajectory with an array of pairs the of delta (representing how far it moves) and theta (representing how much it must turn) values. Of course the robot first needs to turn to aim and then go forward. By doing this, when Zeus calculates the route, it breaks down into the number of nodes that the route has and sends them to Pluto as a series of combinations of turning and moving.

2.4 Communication module

The communication module is responsible to manage the communication between the Android device, Zeus and Pluto.

2.4.1 Architecture and protocol

First I had to choose under which communication protocol the system would work. I thought about Bluetooth first, but then I thought that it would be difficult to connect three devices at the same time. Finally I chose TCP-IP protocol because there are already Python-ready approaches [11] that I could use in Raspberry Pi. I needed to give WiFi capability to both devices, so I decided to buy two (one for Pluto and one for Zeus) WiFi USB compatible adapters (or dongles) and plug them into every Raspberry Pi. As those devices are Plug&Play I didnt need to configure anything to start working.
I decided that Zeus would act as a server, offering wireless connection, while Pluto would be the client. I decided that because in future works, more robots like Pluto can be added to the system, so they must connect to the same point, Zeus, and this is the only way that it can be achieved. To do it, I used Python sockets. Zeus would keep listening to all the commands that an Android device would send and then it would establish a parallel connection with Pluto to be able to send it the path. To test this module, first I made a simple chat application between the two Raspberrys.

2.4.2 Android comunication

To be able to control the entire system and to see what Zeus is watching, I decided to build an Android application (taking advantage of the fact that almost all smartphones have WiFi connection). This application also acts as a client (like Pluto) and must be connected to Zeus via WiFi, like an internet access point. The purpose of the application is to connect to the system using TCP-IP sockets (this time in Java) and request what the webcam is seeing. As the image is a matrix of values and we can only send string via sockets, I had to manage

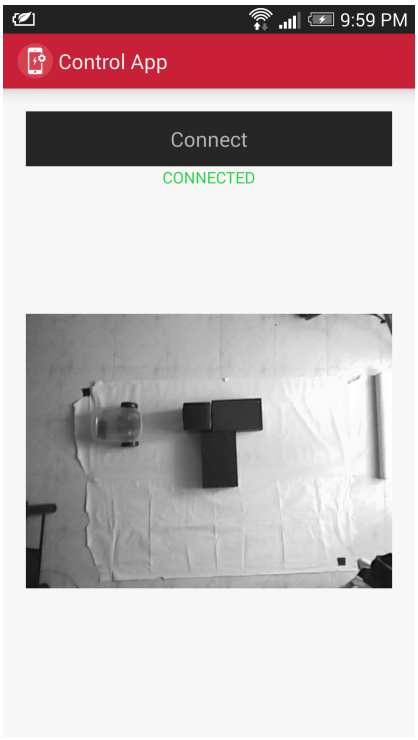

Fig. 12. Android app interface

to convert the image into an array string. I achieved it coding the image in the server using base64 algorithm, sending as an array string and decoding it on the Android app side with base64 again. There are also more methods to encode and decode an image to a string array, but this one is already implemented in Java, so its easier to work with Android. I also put a button in the app to send a START command to Zeus that starts the path-finding algorithm and a capture button to see what Zeus is watching. With this application, the user can interact with the system without having to connect any external devices such as a screen, a keyboard or mouse to have control on both Raspberrys.

3 Experimental Setup

For the realization of this project this is my experimental setup: As seen in the image X I use a white paper for the background, black cardboard boxes to be the obstacles (this makes the detection easier) and I use black paper to identify the P_{start} and P_{goal} points in the ground. For the implementation of the action module I used a Tupperware to put inside all the robot elements (a Raspberry Pi and the MD25 control board) because its made of plastic and its very malleable. For the implementation of the vision module I mounted a webcam in the ceil of the room to be able to catch the entire scene connected to an a Raspberry Pi in a table. For the user control module I used an HTC One smartphone installing the control application on it.

4 Results

For the implementation of the PRM algorithm, I tried to build the graph with several different configurations, but I was interested in having fewer nodes and more connections, rather than having more nodes and fewer connections (The fewer nodes we have, the faster the path-finding algorithm will be). My best configuration parameters were 70 nodes with 8 connections. I tried the 2 most common path-finding algorithms: Astar and Dikjstra. Due to the reduced computing resources in a Raspberry Pi I decided to create my own path-finding algorithm mixing them both. The result is the Lagart algorithm, which is considerably faster than both of them. The Lagarts pseudo-code is attached in the appendix A. For the implementation of the template matching in this case I used the template matching function cv2.matchTemplate of OpenCV library. You need to pass the template you are looking for and the detection method. For the implementation of the opening transfomration I used cv2.morphologyEx(img, cv2.open, kernel), for closing cv2.morphologyEx(img, cv2.close, kernel) and for dilation cv2.dilate(img,kernel,iterations = 1) For all the cases I used a 7x7 pixel square kernel. This size was chosen after trying different values. For this project, the height of the location of Zeus is not relevant. The higher it is located, the lower resolution per cm we will get, so it is interesting to put it as close as we can, but being able to see the markers. A good point to locate it is the ceiling. In this task, I also had to select the resolution of the captured image. Big resolution means more pixels to iterate, which leads to mode processing time, but more resolution per cm. I chose a 640x480

px image resolution.

After analyzing the methodology of all separated modules and combining them to make this project successful, here are the results. Combining all the modules I accomplished the objective of this project: creating an autonomous robot only guided by computer vision. This robot, equipped with 3 wheels, is able to avoid physical predetermined obstacles to go from a point A to a point B without using any sensor, except the encoders of the motors which give information about how much the robot has moved or turned. I have applied the computer vision, and algorithm optimization and artificial intelligence knowledge learned at the Universitat Autonoma de Barcelona about computing module Zeus, making it possible to control all the robots available (one in this case). I've created an algorithm faster (for this case) than the most-used path-finding algorithms to optimize the Raspberry Pi resources, decreasing the execution time of the entire algorithm due to the limited resources of a low-cost computational model such a Raspberry Pi. It is true that maybe this new algorithm reduces the execution time only for this particular case. I created a Python library to help the connection between a Rasp-berry Pi and a MD25 motor controller board using Adafruits I2C protocol library. I created an Android application able to control all the system. In the figure [13] we can see how much less processing time Lagarts algorithm takes for this particular case compared to 2 of the most-used algorithms in path-finding Astar and Dijkstra.

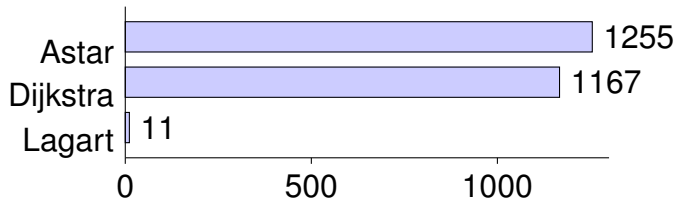

Fig. 13. Comparison between Astar, Dijkstra and Lagart execution time path-finding algorithms (in ms)

5 Conclusions

After working in the field of robotics and on computer vision for months and being able to be in direct contact with this kind of technology, I am able to draw some conclusions: As a first conclusion, from my point of view, I can say that computer vision is totally applicable to the field of autonomous robots. With current technology (rising of computing power, big cameras resolution...) applying computer vision in real time does not mean an obstacle when used on autonomous robots. Another conclusion is that every time you work with computer vision, the project must be done in a controlled environment. It can become an inconvenient when you want to work in real world situations. The light varies its position making shadows behind the obstacles and objects, and they can be very difficult to identify, as you do not know how to distinguish the real object from its shadow. There are a lot of physical uncontrolled conditions that make the probability to fail in the project very high. As this project has been done in controlled conditions (white background, fixed light, black objects, markers...) I can say that the result is successful and the system is functional in all the cases. Thanks to the appearance of new fast wireless network technology and the velocity of WiFi networks (very extended technology), nowadays it is possible to remove the main computation process of the autonomous robots and centralize it in a remote place and communicate them through wireless networks as I have demonstrated doing this project. To conclude, I can say that this technology is getting closer to the costumer every day and the applications where autonomous robot can be applied are increasing in all the fields.

6 Applications

In this section I am going to explain the possible commercial uses of this technology extrapolated to the real world. For example, this project could be implemented to help blind people. One robot could guide the person on the street and a quadcopter could be used to install a webcam to follow and analyze the scene. With the help of this robot, blind people

would not need to use Guide dogs. These are usually really hard to train and there are not enough of them for so many blind people. Another application can be to use this system to manage autonomous vehicles in smart cities intersections. Positioning one (or more, to make the system more robust) the system can communicate via wireless network with the cars and improve the interaction between them in the conflictive intersection. The system could also take into account pedestrian movement to avoid accidents.
The last example I can imagine is applying the system to security surveillance. One or more cameras can be watching the scene and one or more robots could act if needed if a robber is detected. This project has a high educational value as teaching support. As this project is also highly modulable, it will be used as a practicum support for some related subjects at the Universitat Autonoma de Barcelona, such as Robotics or Computer Vision. From this project, a student can get the acknowledgement of robotics, computer vision, image recognition, protocol communication, Android development and hardware development combined into a single project, building up an entire fully working system that one can interact with and that one can see working in real life. One main contribution of this project is the definition of small modules that can be used independently. These are:

- **Action Module:** Uses the python library I made for connecting both Raspberry Pi and MD25 board. It also uses Adafruit I2C library. Acts as a client of the communication module.
- **Vision module:** Runs the PRM and Lagart path-finding algorithm I made to generate the path. Also acts as a server for the communication module.
- **Communication Module:** It's split in 3 devices: Android phone, Pluto and Zeus. It's based on a server-client model via TCP-IP sockets.

You can download the source code of the entire project here: mv.cvc.uab.es/robotrp

7 Future work

To improve the system, a high capacity battery can be added to Pluto. Now it must be plugged in and it cannot move so far away due to the length of the power cable.
Also Zeus can be mounted on a quadcopter or another flying UAV flying over the scene to delete the need to be located in a fixed place (usually on the ceiling). In this case, the quadcopter could follow the robot (this case Pluto) and we would not need to use the delimiter markers and a fixed workspace.
A further future possibility could be to add another action module (another Pluto, for instance) and make them interact amongst themselves to achieve a concrete action that could only be made by cooperating. The system could also be online. In this case, Zeus would only send the first part of the path calculated (from node a to node a+1) and when the robot arrives at the last node, recalculate the path and send him another part. This would improve the precision of the robot and also avoid the obstacles if the condition changed (for example if another obstacle is added to the scene while the robot is moving). The last improvement I would do to the system is to determine the threshold of the object detection module dynamic, deciding the value depending on the conditions of the scene.

Appendix A
Lagart algorithm

Ive developed this path-finding algorithm due to the limited computing resources of the Rasp-berry Pi. I achieved it by mixing the 2 most used path-finding algorithms, Astar and Dijkstra. The idea of the algorithm is this: We have a node list representing the current path. At the beginning this list only contains the node A (where the path starts). Then a do-while loop iterates until the list contains the B node (the last node we want to go). For every iteration, we take the last node we visited we take its neighbors and sort them out. The heuristic I used to sort them is the distance from the neighbor to the last node. By doing this, the algorithm always selects the neighbor which is closest to the end node.

The algorithm only has one path list. This makes it faster than Astar, which has more path lists and sorts them every iteration of the loop. My algorithm does not ensure the fastest path but it is considerably faster. The trade off is accepted for this project. Here is the pseudo-code of Lagart algorithm:

Lagart's pseudo-code

$path \leftarrow [firstnode]$
while lastnode in path == False **do**
  $currentNode \leftarrow path[lastElement]$
  $distances \leftarrow []$
  for $i = 0$ to len(currentNode.neighbours) **do**
    $neighbour \leftarrow currentNode.neighbours[i]$
    $distance \leftarrow euclideanDistance(neighbour, lastNode)$
    distances.append(distance)
  end for
  $sorted \leftarrow sort(distances, asc)$
  $selectedNeighbour \leftarrow selectFirst(sorted)$
  path.append(selectedNeighbour)
end while
return path

ACKNOWLEDGMENTS

I would like to firstly thank Fernando Vilariño, who has been my tutor and who gave me some of the ideas for this project. He has guided me for the project to be successful. I would also like to thank the Computer Vision Center (CVC) for all the resources given. Thanks also to Jordi Pars, who provided me with some of the Pluto materials, to Centre Medic Martorell for providing materials for the set up and to Dani Torrescusa for programming help given. I would like to thank Felipe Lumbreras for helping me to rapair the robot and Metze for correcting this text. Lastly I would like to thank all my family and friends who also have been involved in the project, helping me from the beginning.

REFERENCES

[1] OpenCV dev team. "Introduction to OpenCV" http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html (Accessed June 25, 2014)

[2] MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive Documentation *http://www.robot-electronics.co.uk/htm/md25tech.htm*

[3] F. Mattias *Holonomic versus nonholonomic constraints*. Faculty of Technology and Science, Karlstads universitet, Karlstad, 2012.

[4] Cole,L *Visual Object Recognition using Template Matching*. Australian National University, Australia.

[5] Walker, H. M. *Degrees of freedom.*. Journal of Educational Psychology, Vol 31(4), Apr 1940.

[6] M. Peter *Are we making real progress in computer vision today?*. Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA.

[7] Rafael C, Richard E. *Digital Image Proessing*. Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA.

[8] Adafruit team *Adafruit I2C Python library*. https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code (Accessed June 25, 2014)

[9] Elinux team *Raspberry Pi compatible USB webcams*. http://elinux.org/RPi_USB_Webcams (Accessed June 25, 2014)

[10] Choset,H, et al. *Principles Of Robot Motion. Theory, algorithms and implementation*. MIT Press. 2005.

[11] Python implementation of the K-means clustering algorithm *http://pandoricweb.tumblr.com/post/8646701677/python-implementation-of-the-k-means-clustering*

[12] TCP/IP Client and Server *http://pymotw.com/2/socket/tcp.html* (Accessed June 25, 2014)