Article

A Distributed Vision-Based Navigation System for Khepera IV Mobile Robots

Gonzalo Farias [1,*], Ernesto Fabregas [2], Enrique Torres [1], Gaëtan Bricas [3], Sebastián Dormido-Canto [2], and Sebastián Dormido [2]

[1] Pontificia Universidad Católica de Valparaíso, Av. Brasil 2147, Valparaíso, Chile; gonzalo.farias@pucv.cl
[2] Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia, Juan del Rosal 16, 28040 Madrid, Spain; efabregas@dia.uned.es
[3] The National Institute of Electrical Engineering, Electronics, Computer Science, Fluid Mechanics & Telecommunications and Networks. 2, rue Charles Camichel, 31071 Toulouse Cedex 7, France
* Correspondence: gonzalo.farias@pucv.cl

Abstract: This work presents the development and implementation of a distributed navigation system based on computer vision. The autonomous system consists of a wheeled mobile robot with an integrated colour camera. The robot navigates through a laboratory scenario where the track and several traffic signals must be detected and recognized by using the images acquired with its on-board camera. The images are sent to a computer server that processes them and calculates the corresponding speeds of the robot using a cascade of trained classifiers. These speeds are sent back to the robot, which acts to carry out the corresponding manoeuvre. The classifier cascade should be trained before experimentation with two sets of positive and negative images. The number of images in these sets should be considered to limit the training stage time and avoid overtraining the system.

Keywords: mobile robot; vision-based navigation; cascade classifiers

1. Introduction

The current development of robotics has been influenced by the growth of NICT (New Information and Communication Technologies), which has provided the perfect scenario for the confronting of new challenges. In this context, the autonomous navigation of robots based on vision has grown considerably, showing increased interest for some years now.

This interest began some years ago, when researchers [1] and [2] presented two systems for navigation based on vision with obstacle detection and avoidance in outdoor scenarios in 1988. These works obtained very good results considering the technologies available at the time (cameras, processors, communications, etc.). Another example was published in 2002, in which the authors presented a study on the navigational vision of mobile robots spanning 20 years [3].

In this comprehensive and in-depth survey, interesting breakthroughs can be found in approaches that make a distinction between indoor and outdoor environments [4–7]. In recent years, navigation based on vision has shown renewed interest in the scientific and educational communities [8–11] because it has many practical applications in daily life.

Robots that can navigate autonomously are complex systems involving several components that have a common denominator: communication capabilities and the speed of processing and operation. These capabilities are very important because an autonomous robot must have all the necessary information available to make the right decisions at every sampling time.

Currently, it is common to find robots in the market that integrate cameras with good resolution, high processing performance and communication capabilities such as Bluetooth or WI-FI, which were

not available until recently; examples include the e-puck robot [12], Khepera IV [13], Pioneer 3-DX [14], and TurtleBot3 [15].

These capabilities are important in robot vision-based navigation because robots need to acquire images of their environment, process them, and make decisions to navigate through a scenario [16] and [17]. In the case of robots that are not capable of performing complex tasks such as image processing, these images can be acquired with a camera and sent to a server for processing. The server can answer with the orders for the motors of the robot to navigate, or these orders can be calculated in the robot with the results of the image processing. This introduces a delay of the network in the control loop, which is an important element that must be considered in these systems. In a local WI-FI network, this delay should not affect the operation of the system. The fact that the processing is done in a server allows another type of hard task, such as the implementation of machine learning algorithms [18–20], which may require a training stage before use.

In the literature, you can find some research articles related to vision-based systems using different techniques. For example in [21], the authors present a system for the detection of persons using cascade classifiers. Its method uses the implementation of HOG (Histogram of Oriented Gradients) feature descriptor and AdaBoost algorithm with a 2D laser and a camera. In [22], the authors developed a vision-based control system for the alignment of an autonomous forklift vehicle. They implemented Haar feature-based cascade classifiers for detection and heavy spool pick-and-place operation. In [23], the authors presented two systems for detection and classification of traffic signs in real time. The first method is divided into three stages: color segmentation using a threshold, detection using SVM (Support Vector Machine) and classification tree-like and Random Forest. In [24], the authors propose a detection and recognition system for road warning signs with voice notification for autonomous vehicles. It implements Haar feature-based cascade classifiers. In [25], the authors describe a navigation system of a robot based on paths following. The method uses a combination of the detection of colours, textures and lines with different classifiers.

As can be seen, there is a wide range of approaches to this topic. After an exhaustive analysis of all mentioned works, we realized that none of them is available for laboratory practices with students, which is one of our main purposes with this research.

This article presents a distributed vision-based autonomous navigation system with machine learning in a laboratory environment. The system consists of a Khepera IV robot that acquires the images and sends them to a server. On the server, a trained cascade of classifiers processes the images and calculates the linear and angular velocities of the robot to navigate through the indoor experimental environment built in the laboratory.

The main motivation for this work is that our Engineering School has implemented a platform for educational and research purposes in the field of mobile robotics [26,27]. In which students can experiment with their own controllers in order to incorporate these concepts in the teaching of robot control. In this sense, experiments on position control, obstacle avoidance, among others, have been implemented with very good results, for example [28–31].

Now the purpose is to use the robot's on-board camera to develop even more challenging experiments that attract the students in an interactive environment. These artificial vision experiments can improve the quality of the laboratory practices, which can mean a step up in the teaching-learning process of mobile robot labs.

The main contribution of this work is to propose the use of advanced computer vision algorithms to perform much more sophisticated and engaging experiments with practical mobile robot laboratories for pedagogical purposes. The motivation of this is to provide much more challenging experiments for the students to improve the quality of the teaching-learning process in this field. A summarized list of contributions of this work is the following: 1) the incorporation of new computer vision capabilities to the practical mobile robot laboratories; 2) to provide much more challenging experiments in an interactive and engaging environment; 3) introduce advanced algorithms for image processing and artificial intelligence techniques for teaching control of mobile robots; and 4) the experiments can

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

be tested in simulation firstly, and after that, they can be implemented in a real and easy-to-use environment in a relatively fast and straightforward way.

The remainder of the paper is organized as follows: Section 2 presents the fundamental concepts related to this article; Section 3 describes the implementation of the experiments in the laboratory; Section 4 shows the results of some test experiments developed in simulation and with the platform; and finally, Section 5 presents the main conclusions and future work.

2. Background

The vision-based system consists of the processing of images acquired by the on-board camera. Today, you can find some software libraries to carry out this process, such as Torch3vision, VXL (Vision something Libraries), JavaVIS, LIT-lib, and OpenCV (Open source Computer Vision library) [32]. After some studies and tests, OpenCV was selected to carry out the image processing. It presents the most complete solutions for the problems that we face processing the images acquired by the robot.

OpenCV is a library of functions for real-time computer vision that was developed by Intel. It is free for use under an open-source license. It has interfaces for C++, Python, and Java. It is very versatile because it can be compiled for Windows, Linux, Mac OS, IOS and Android. It has more than 2500 optimized algorithms that include machine learning, face detection, motion detection, object tracking, 3D object model extraction, finding similar images in a database, augmented reality [33], etc. In our case, its main drawback is that it cannot be executed on the robot. That is why we need to send the images from the robot to a server to process them and calculate the orders for the robot.

2.1. Image Segmentation

Different techniques are used to obtain the features of an image. One of the most commonly used techniques is segmentation, which consists of dividing the image into multiple segments (set of pixels) to obtain the information from these segments (also known as super-pixels). The idea is to simplify the image into something that is easier to analyse [34]. This process is commonly used to detect objects and boundaries such as lines, curves, edges, corners, etc. As a result, a set of segments is obtained that is determined by regions with similar characteristics such as intensity, colour or texture. The simplest method in segmentation is thresholding, which can create a binary image (black/white) from a colour image. If the original image is *(x,y)*, the segmented image is *(x′,y′)*, which is determined by the threshold $U(0 < U < 255)$. This operation is defined as follows:

$$(x', y') = 255,\ if\ (x, y) > Threshold \tag{1}$$

$$(x', y') = 0,\ if\ (x, y) <= Threshold \tag{2}$$

The value of the threshold is selected depending on the colours that must be grouped to establish, for example, a difference from the background of the image. Figure 1 shows the traffic signals that will be used for navigation.

Figure 1. Traffic signals that must be detected.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

2.2. Cascade of Classifiers

Cascade classifiers are a concatenation of ensemble learning based on several classifiers. Each basic classifier implements the AdaBoost algorithm in a decision-tree classifier with at least 2 leaves. The information from the output of a given classifier is used as an additional input of the next classifier in the sequence [35]. All the classifiers of the cascade are trained with a set of images of the same size called "positive", which are sample views of the object to be detected and arbitrary "negative" images of the same size.

After the classifier is trained, it can be applied to a search window (of the same size as used during the training) to detect the object in question in the entire frame [36]. This process is repeated until at some stage the analysed segment of the image is rejected or all the stages are passed. The search window can be moved across the image to check every location for the classifier. The classifier outputs a "*T*" if the region is likely to show the object and "F" otherwise. In an image, the extraction of features can be obtained from several methods, including Haar, LBP and HOG. Figure 2 shows this process [37].

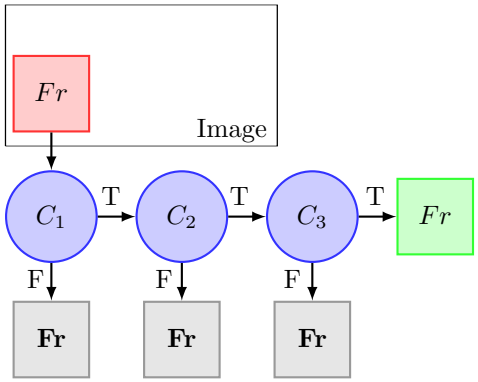

Figure 2. Applying cascade classifiers to the search window on an image.

The magenta circles represent the classifiers (C_1, C_2 and C_3). The red square represents the frame of the image that is being analysed. If the object is detected, the output of all classifiers is T, and the (green square) frame represents this situation. However, if some of the classifier outputs are F, the frame is rejected (grey squares), and the object is not detected in this frame.

3. Implementation in the laboratory

3.1. Platform used

The implementation of the system is based on the platform developed by the authors in a previous work [26–28] with some modifications. Figure 3 shows a diagram of the platform in the laboratory.

Figure 3. Diagram of the platform used for the experiments.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

The system is composed of Khepera IV robots that run software that grants communication and operation. The other component is a server that processes the images acquired by the Khepera IV robot and calculates the velocities to control it. As mentioned before, this software has been developed in Python and uses the OpenCV library to work with the images. The wireless communication between the robot and the system is carried out with a WI-FI router. Furthermore, a Play Station 3 USB camera is connected to the server to obtain the overhead view of the experiments in the server. This video signal is used to watch the development of the experiments by the user.

The setup of the platform for the experiments is the following: The dimensions of the arena are $2.0m \ x \ 1.5m$. The dimensions of the robot Khepera IV are $140mm$ (diameter) and $58mm$ (height). The Dimensions of traffic signals are $26mm \ x \ 26mm$. The Number of traffic signals is 8.

3.2. Objects detection system

Figure 4 shows the block diagram of the system. On the left side, the robot is represented by the green dashed line block.

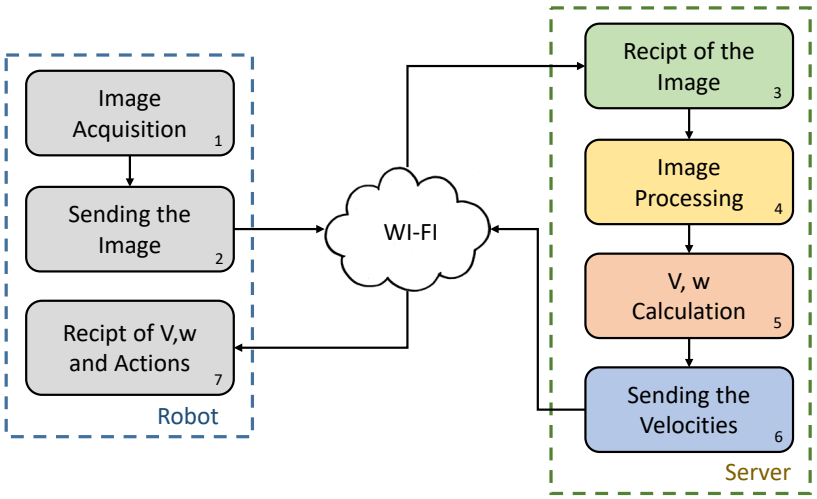

Figure 4. Block diagram of the system.

On the right side, the server is represented by the yellow dashed line. The system works as follows, where the numbers represent the order in which the task is executed:

- Block 1: Robot acquires the image.
- Block 2: The image is sent to the server using the WI-FI connection.
- Block 3: The image is received by the server.
- Block 4: The image is processed in the server.
- Block 5: The linear and angular velocities of the robot are updated using the results of block 4.
- Block 6: The velocities are sent to the robot through the WI-FI connection.
- Block 7: The robot receives the velocities and sends them to the motors.

The software that runs the robot is very simple. It consists of two main functions: 1) to acquire the image from the on-board camera and send it to the server, and 2) to receive the velocities from the server and update them in the motors. On the other hand, the software that runs at the server is more complex because it carries out more difficult tasks, including image processing. Figure 5 shows a simplified block diagram of this application. The tasks are divided into the four blocks of the server side of Figure 4.

The first task is to check if an image has been received. If an image is received, it is processed in the second task (Image Processing) as follows: 1) the image is read, 2) it is converted to grey-scale, 3) its resolution is obtained, 4) traffic signals are detected with the trained cascade of classifiers, and 5) if a signal is detected, its size is obtained. After that, the third task calculates the velocities of the robot

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

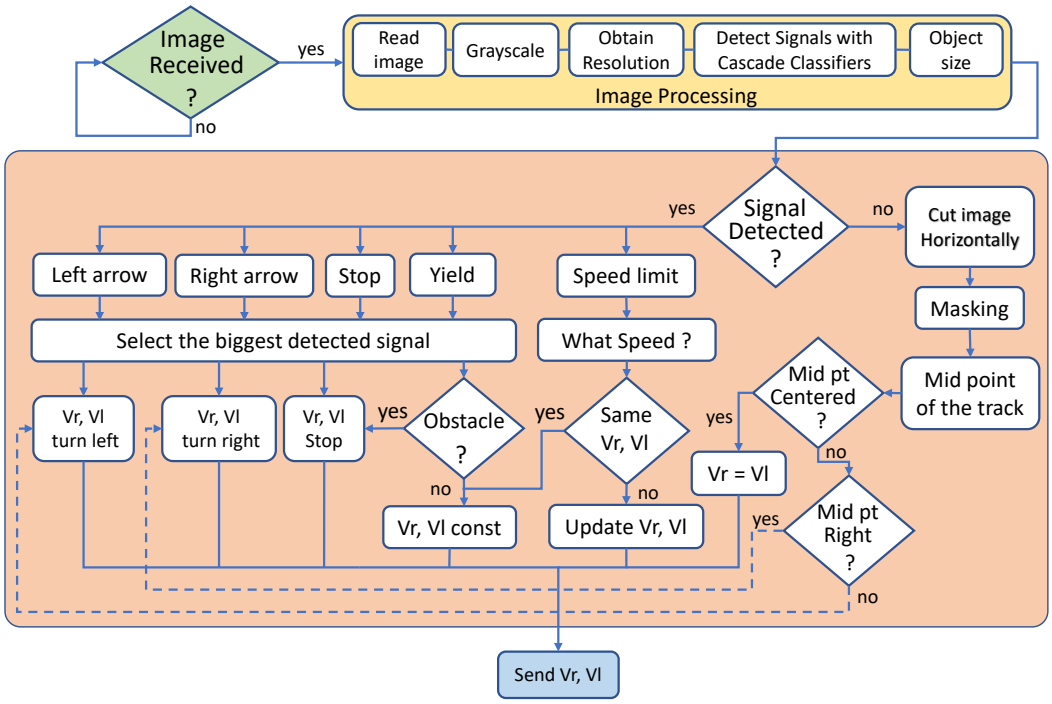

Figure 5. Block diagram of the server side application.

from the object detected. At the beginning of this task, the process forks into two main paths: 1) if a traffic signal is detected and 2) if the track is detected.

If a signal has been detected, depending on its type (left arrow, right arrow, stop, yield or speed limit), the velocities are calculated to make the corresponding action. For example: a left arrow V_L is equal to 0 and V_r is equal to 0.025, which makes the robot turns to the left. If the detected signal is a stop, both velocities are set to 0. If the signal detected is a speed limit, the signal is processed to determine the corresponding speed. On the other hand, if the object detected is a track, the image is processed by thresholding to determine if the track indicates that it is a straight path or a turn to the left or to the right. In both cases, when the velocities are calculated, they are sent to the robot and the application reverts back to the beginning to wait for other new images.

3.3. Implementing cascade classifiers

To train the classifiers, existing databases of "positive" and "negative" images can be used, or customized databases can be created. These image training sets can be created using OpenCV. In our case, the "negative" image set was created using the on-board camera of the robot.

Figure 6. Examples of negative images acquired by the robot.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

Figure 6 shows on the left side, 4 examples of "negative" images obtained by the robot. To provide correct training, these images must not contain the object that wants to be classified as positive. The set of negative images for this experiment was composed of approximately 1400 images. Figure 6 shows on the right side, 4 examples of the positive images acquired by the robot. In this case, the object that wants to be detected is the signal of the airport, which is included in all positive images. In this case, the number of positive images was approximately 1000. With these two sets of images, the OpenCV function "training cascade" was used to carry out the training stage.

The selected classifiers were Haar-like with 5 stages in the cascade. A high number of classifiers in the cascade improves the classification, but the training time can increase exponentially. We selected Haar feature-based algorithm because this classifier showed encouraging performance with high success recognition rates for this experiment.

3.4. Code of the robot

As mentioned before, the programming code of the robot has been developed in Ansi C, which is the programming language that the robot uses. The following code segment shows the algorithm. Note that some lines have been omitted for space reasons.

```
1  #include <khepera/khepera.h>,<signal.h>;
2  #include <stdio.h>,<string.h>,<sys/socket.h>;
3  #include <arpa/inet.h>,<unistd.h>;
4
5  static knet_dev_t$*$dsPic; // Robot PIC microcontroller
6  int main(int argc, char $*$argv[]){
7  ...// Define variables and Server Socket
8  ...// Server Socket is listening (192.168.0.111:8080)
9  ...// Accept connection from an incoming client
10 ...// Initiate libkhepera and robot access
11 ...// Open robot socket, store the handle in its ptr
12 ...// Initialize the motors with tuned parameters
13
14  // Receive a msg from client
15 while((read_size=recv(client_sock,client_msg,4,0))>0){
16   // Execute received velocities
17   write(client_sock,client_msg,strlen(client_msg));
18   vLeft=client_msg[0]-'0';
19   vRight=client_msg[1]-'0';
20   printf("%i %i \n",vLeft,vRight);
21   kh4_SetMode(kh4RegSpeedProfile,dsPic);
22   kh4_set_speed(vLeft*10,vRight*10,dsPic);
23   close(socket_desc);
24  }
25  ... // If an erroneous msg received rise an error msg
26  ... // If some button is pressed program stops
27 }
```

From lines 1 to 10, the variables and some configurations are defined. The communication socket is created and connected to the server. Through this connection, the robot sends the images acquired by the camera of the robot with the MJPG-streamer application [38]. Then, the parameters of the motors of the robots are initialized. From lines 13 to 22, a message received from the computer is processed. The velocities of the robot received in the messages are sent to the motors.

3.5. Code of the server

The following code segment shows the implementation at the server side. This code has been developed with Python. An important detail is that the server reads the images acquired by the robot using the TCP connection created and the application MJPG-streamer that is running at the robot. Note that some lines have been omitted for space reasons.

```
1 import re,os,sys,cv2,numpy as np
2 import threading,argparse,time,urllib,socket
3  ... # Variables and parameters definition
4  ... # Creating client Socket (port 8080)
5  ... # Loading trained cascade classifiers
6 while rval:
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1


```
# Read the image from the robot
# Execute cascade classifiers with the image
stops=stop_csc.detMulSc(scframe,1.05,5,
                        0,(1,1),(500,500))
airport=airport_csc.detMulSc(scframe,1.15,
                        5,0,(1,1),(500,500))
rArrow=rArrow_csc.detMulSc(np.rot90(scframe,2),
                        1.1,5,0,(1,1),(500,500))
lArrow=lArrow_csc.detMulSc(scframe,1.1,5,0,
                        (1,1),(500,500))
Yield=yield_csc.detMulSc(scframe,1.1,5,0,(1,1),
                        (500,500))
signs=CLASSIFIER.detMulSc(scframe,1.2,5,0,(1,1),
                        (500,500))
... # Depending on the detected signal
    # its area is calculated
... # Speed limits are calculated using Flann technique
# Time of action of the signal
if time.time()-time_stop<5: state="STOP"
elif time.time()-time_TurnL<5: state="TURN_LEFT"
elif time.time()-time_TurnR<5: state="TURN_RIGHT"
else: state="GO"
if state == "GO":
 # Calculate the center of the track
 hsvTrac=cv2.cvtColor(imagenTrac,cv2.COLOR_BGR2HSV)
 low_white=np.array([65,85,150],dtype=np.uint8)
 high_white=np.array([79,90,380],dtype=np.uint8)
 # Creating the Mask
 maskTrac=cv2.inRange(hsvTrac,low_white,high_white)
 momentsTrac=cv2.moments(maskTrac)
 areaTrac=momentsTrac['m00']
 if(areaTrac > 200):
  xTrac=int(momentsTrac['m10']/momentsTrac['m00'])
  yTrac=int(momentsTrac['m01']/momentsTrac['m00'])
  cv2.rectangle(imagenTrac,(xTrac,yTrac),
                        (xTrac+2,yTrac+2),(0,0,255),2)
 # Calculate the velocities for each signal
 if abs(xTrac-192/2) < 5:
       v_left=int(int(lastlimit)/14),
       v_right=int(int(lastlimit)/14)
 elif xTrac > 144:
      v_left=int(int(lastlimit)/14),
      v_right=int(int(lastlimit)/56)
 elif xTrac < 192/4:
      v_left=int(int(lastlimit)/56),
      v_right=int(int(lastlimit)/14)
 # Else, turn to the right or to the left
 elif xTrac > 192/2:
      v_left=int(int(lastlimit)/14),
      v_right=int(int(lastlimit)/28)
 elif xTrac < 192/2:
      v_left=int(int(lastlimit)/28),
      v_right=int(int(lastlimit)/14)
 cv2.imshow('Mask', maskTrac)
 elif state == "SLOW":
      v_left=int(int(lastlimit)/28),
      v_right=int(int(lastlimit)/28)
 elif state == "STOP": v_left=0, v_right=0
 elif state == "WAIT": v_left=0, v_right=0
 elif state == "YIELD":
      v_left=int(int(lastlimit)/28),
      v_right=int(int(lastlimit)/28)
 elif state == "TURN_LEFT":
      v_left=0, v_right=int(int(lastlimit)/14)
 elif state == "TURN_RIGHT":
      v_left=int(int(lastlimit)/14), v_right=0
 # Velocities converted and send them to the robot
 # Sending the velocities to the robot
 cv2.putText(origframe,"Current speed limit:"
                        +str(lastlimit)+"km/h.",(5,20),
         cv2.FONT_HERSHEY_SIMPLEX,0.45,(255,255,255),1)
 cv2.imshow("preview",frame)
 k=cv2.waitKey(30) & 0xff
 if k == 27: break
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

From lines 1 to 5, certain tasks are carried out: the variables are defined; the client socket is created and connected; and the trained cascade classifiers are loaded. After that, the main loop starts. From lines 9 to 20, the image from the robot is obtained and conditioned. From lines 24 to 29, the cascade of classifiers is applied to the obtained image. The result of this operation is that only one classifier must be true. If not, the traffic signal is detected with the classifiers, which means that the track must be followed. With the result of the classifiers, the velocities are calculated from lines 30 to 72. Then, the velocities are sent to the robot from lines 73 to 77. Finally, the program waits for a key to finish.

4. Experimental Results

In this Section, all the tests that were carried out for the detection of objects and navigation of the robot are shown. The results are represented from the detection of basic signals (arrows of the same colour) to more complex signals, which are involved in the tests of the designed classifiers (presented in Section 3), as well as traffic signals and new scenarios. These results reflect the efficiency of this new algorithm, both simulated and with the real robot. The Section describes three experiments. The first experiment shows the difficulties of detecting multiple objects by using traditional image processing. The second experiment describes the advantage of the cascade classifier for recognition of several traffic signals, and finally, the third experience provides the results of the implemented vision-based navigation system to perform an advanced experiment in the real environment.

4.1. Experiment 1: Object detection by image processing

To initiate the proposed algorithm, a simulation of a simple scenario was performed using the V-REP simulator and the Khepera IV robot model previously developed by the authors [29] and [30]. The experimental concept is that the robot can navigate through the arena, detecting the arrows.

This work is based on a previous work of the authors presented in [31]. The robot control algorithm must be able to calculate the corresponding speeds that allow the robot to turn in the correct direction depending on the arrow detected. For the first experiment, the arrows are of the same colour to simplify the identification because, using the threshold technique, the arrows can be easily detected. Figure 7 shows the configuration of this experiment on the left side. In the upper right corner of the scenario, the images acquired by the on-board camera of the robot are shown in running time.

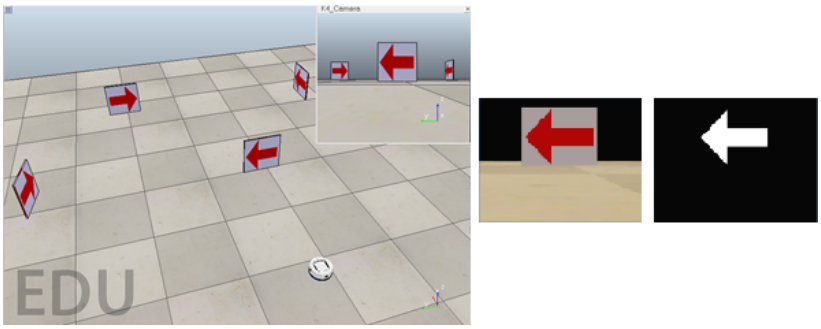

Figure 7. Experimentation detection of arrows of the same color in a simulated environment.

Once the image is detected by the robot in the program, red masking is performed, which only displays the characteristics of the red arrow, as shown on the right side of Figure 7. The image on the left is the image acquired by the robot camera, and the image on the right side is the result from the red masking. The image is segmented in such a way that there is a margin where only the colour red appears. In that image, there may be several arrows, so the robot determines the largest arrow, since it is assumed that this arrow will be the closest to the robot and thus will have more relevance.

OpenCV provides multiple tools to discern the orientation of an object, including cv.minAreaRect, cv.fitEllipse and cv.fitLine. In our case, cv.fitLine was used. This function provides four variables that can be used to obtain two other variables called lefty and righty. In the case of arrow detection, if it is

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

oriented to the left, the value of the lefty variable is positive, and right is negative. However, if the arrow is oriented to the right, these values change their sign.

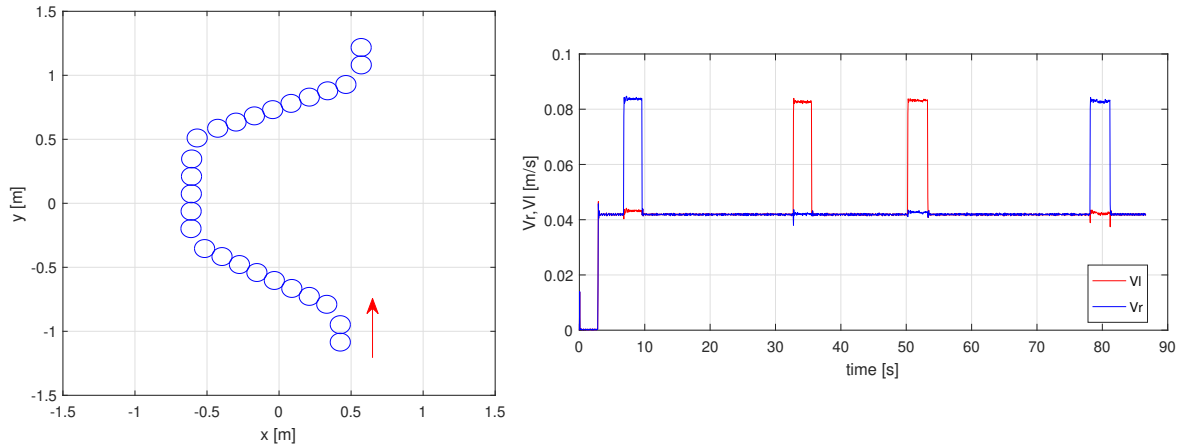

Figure 8. Trajectory of the robot and its corresponding velocities.

Figure 8 shows the path of the robot on the upper side, which indicates that the robot is performing autonomous behaviour. It detects the arrow direction and makes the correct decisions to turn to the right and to the left depending on each arrow. The bottom side of this figure shows the speed behaviour of each motor during the experiment. In the first stage, the speed of the right motor is greater than that of the left, so the robot makes a leftward turn. Then, the speeds are equal again. In the second stage, the speed of the left motor is greater, so the robot turns to the right. In the third stage, the robot turns to the right, and in the last stage, the robot turns to the left. In conclusion, the robot detects four arrows and affirmatively avoids them by producing an algorithm to detect and avoid arrows.

As in the simulated experiment, a test involving the detection of arrows was also carried out with the platform. Figure 9 shows the configuration of this experiment on the left side. The robot is near three red arrows, which are positioned at 80 cm and have an orientation of 90 degrees with respect to each other. As in the previous case, the goal of the robot is to detect these arrows and make decisions to navigate through the scenario.

On the right side of Figure 9, the image acquired by the robot and its corresponding masking of the red colour are shown. In the virtual experience, the cv.fitLine function was used, but in this case, the function did not provide a good result; it did not clearly detect the orientation of the arrow. Therefore, another technique was used according to the following criterion: An arrow has seven vertices, but two of them are located at the top and bottom of the image; therefore, depending on the location of those points on the x-axis, the robot can determine the orientation of the arrow.

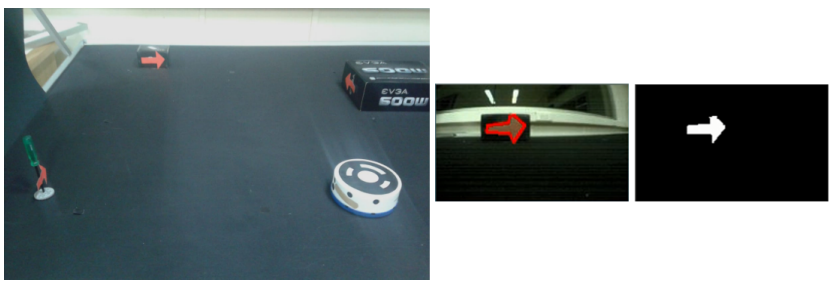

Figure 9. Example of experimental detection of arrows of the same colour in a real environment.

Note that this is a previous stage of the algorithm that we want to implement. This method needs a lot of knowledge (features) about the object to be identified. That is why we have used OpenCV functions and the robot Khepera IV, to show this drawback. On the contrary, we want to implement a

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

system that is capable of detecting known objects from the training stage, and at the same time, a new object can be added to the experiment just re-training the system.

4.2. Experiment 2: Traffic signals detection by cascade classifier in simulation

After testing the system with these simple examples, the detection of more complex objects, such as traffic signals, is implemented both in simulation and with the platform. The simulation test consists of a classic example of a vehicle on a public road. The robot faces different traffic signals during its displacement. In this case, the robot must detect 6 traffic signals to reach its objective, which in this case is the airport signal. Figure 10 shows the configuration of this experiment in the V-REP simulator. On the right side of the figure, the image acquired by the robot shows this signal.

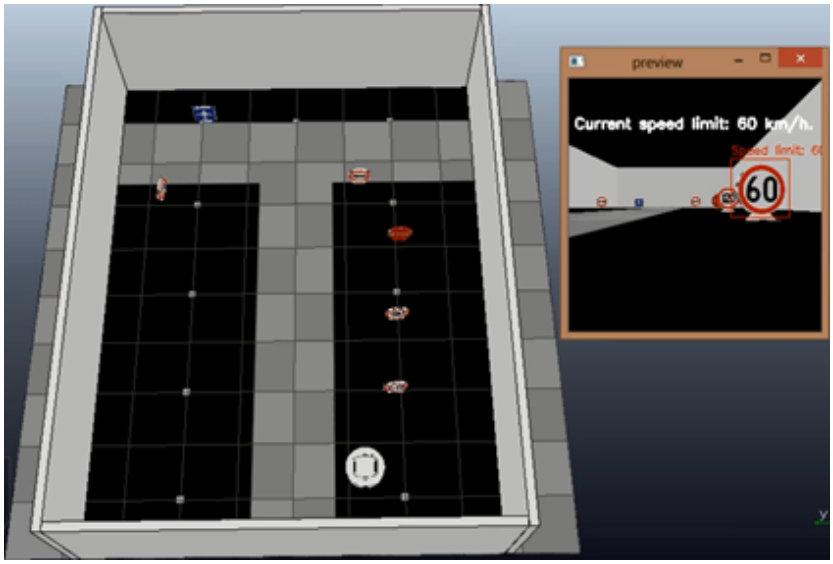

Figure 10. Experimentation detection of traffic signals in simulation.

At the beginning of the experiment, the robot starts to advance linearly with both velocities at 0.0418 [m/s]. When it detects the first signal (60), it decreases both velocities by approximately 0.025 [m/s] over 4.5 [s], which means that it continues advancing but with less speed. Then, it detects the 120 speed limit signal, and it increases both velocities to double the previous value to 0.05 [m/s] over 9.7 [s] and continues with a straight trajectory. After that, the next signal detected is a stop, which makes the robot decrease its speeds until reaching 0 [m/s] over 2.15 [s].

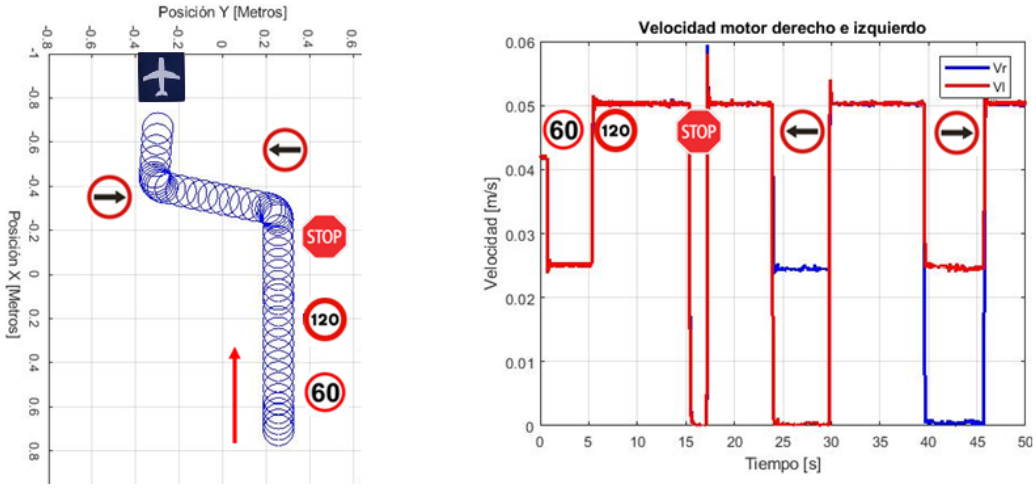

Figure 11. Position of the robot during the experiment and its corresponding velocities.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

The next detected signal is an arrow that indicates that the robot must turn to the left. This makes the left wheel velocity 0 [m/s] and the right 0.025 [m/s] over 5.85 [s]. Then, the next signal is detected by another arrow but indicates turning to the right. This makes the right velocity decrease to 0 [m/s] and the left velocity increase to 0.025 [m/s]. At this point, the last signal (airport) is detected, and the robot stops. Figure 11 shows the position of the robot during this experiment with its corresponding traffic signals in the trajectory on the upper side. On the bottom side, the corresponding velocities of this experiment are shown.

The cascade classifier was trained by following the methodology described in Section 3.3. The training time to build the classifier for the eight traffic signals was about 5 hours.

4.3. Experiment 3: Vision-based navigation in the real laboratory

In this experiment, the vision-based navigation system is implemented in the real environment to detect and classify several traffic signals. Based on the previous results, it was added a white track to the real laboratory in order to aid navigation of the robot. To this end, the images of the track acquired by the on-board camera are also sent to the server and processed by the threshold technique as in [39]. Finally, the robot must move over the track by using the traffic signals. Figure 12 shows the track added to the arena.

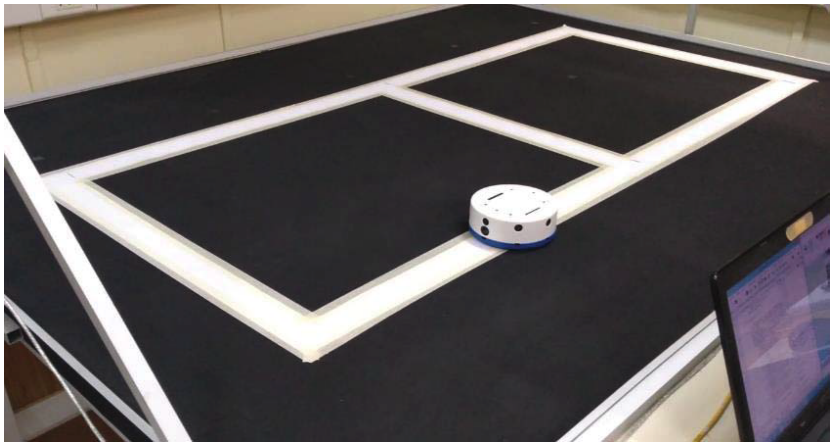

Figure 12. Track added to the platform to help the navigation.

Figure 13 shows three subfigures; in each, the left side shows the image acquired by the on-board camera of the robot, and the right side shows the result of the threshold technique applied to this image. The result of the first image is that the robot must turn to the left. The result of the second image is that the robot must turn to the right. The result of the last image is that the robot must go straight. In this way, the robot will take actions based on both the traffic signals and the track.

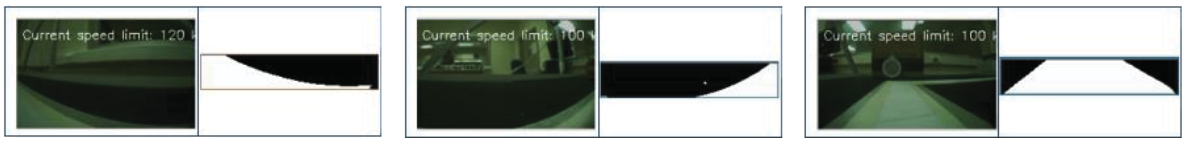

Figure 13. Experimental detection of tracks in a real environment.

After these modifications and tests, a more complex circuit was implemented with 13 traffic signals, including the logo of our university, a McDonald's symbol and the airport symbol. Figure 14 shows the configuration of this experiment in the laboratory with the platform.

The experiment begins with the robot located above the white track to guide its path. The first manoeuvre of the robot is to turn to the right. This order is calculated based on the image of the track obtained by the robot. After that, the next signal detected is a stop, which is applied to reduce the speed of the robot. The next step is a straight advance based on the track until a turn to the right is

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2020 doi:10.20944/preprints202007.0326.v1

Figure 14. Experimental detection of traffic signals in a real environment.

executed, also based on track information. After that, the robot advances straight along the track until a right turn arrow appears, and it continues to the target (McDonald symbol), by following the track and traffic signals.

The main metric for object detection corresponds to a fraction of the frames with a successful recognition of the traffic signals. In the experiment shown in Figure 14, the detection rates for each signal are the following: 100% for right and left arrows, 72% for the stop signal and 88% for the airport signal, 62% for the 60-speed limit and 85% for the 100-speed limit signals.

The upper side of Figure 15 shows the trajectory followed by the robot in the circuit; the bottom side shows the corresponding velocities during the experiment. In the first 195 seconds, the speeds of the motors range between 0 and 0.02 [m/s] since their maximum speed is equivalent to 100 [km/h]. Depending on the speed limit signal, the speeds of the motors increase or decrease. Thus, when the speed limit signal is 60, the maximum motor speed decreases to 0.014 [m/s], and when the speed limit signal is 120, the maximum velocity increases to 0.027 [m/s]. The total duration of the trajectory is 300 [s].

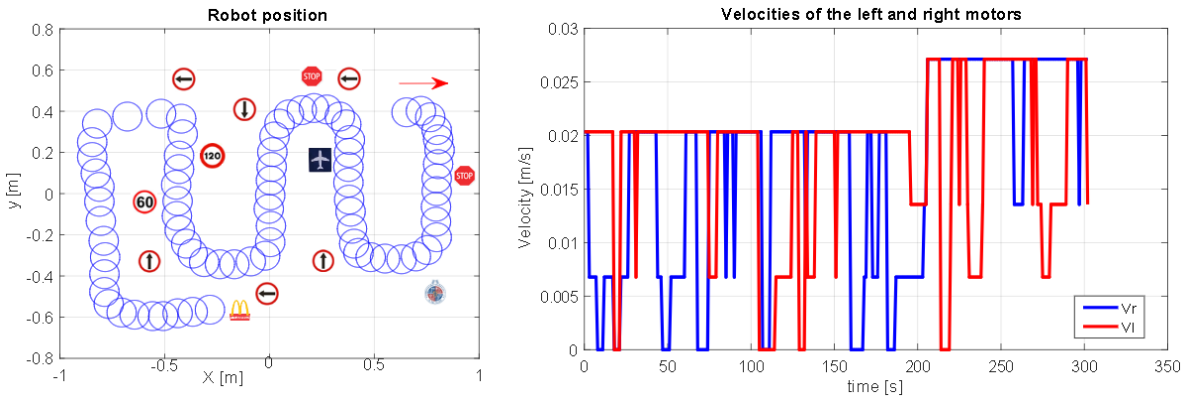

Figure 15. Position and velocities of the robot during the experiment.

Finally note that there are three main processing times in the real environment: Acquisition and sent of the image captured by the robot's camera to the server (approximately between 100ms and 150ms), the classification time of the cascade algorithm (in average about 30ms), and the time required to send the command signals to the Khepera robot from the server (around 10ms). Thus, the total

processing time for the vision-based navigation system is about 200ms, which is equivalent to 4mm at the maximum speed of the robot (27mm/s).

5. Conclusions

This work presented a vision-based system in a laboratory environment. The robot used for the implementation is a Khepera IV, which presents high performance for this kind of application. However, the robot cannot carry out image processing on-board to navigate through the scenario by analysing complex information such as traffic signals.

After some simulations and tests, a server was added to the system to implement image processing. In this way, the system was transformed into a distributed system, where the image acquired by the robot is sent to the server using a WI-FI network configured for the experiments. In the server, the image is processed using different image algorithms. To perform a more complex analysis of the scenario, a cascade of classifiers has been developed to interpret several traffic signals (speed limits, yield signal, stop signal, airport, etc.). The analysis of the traffic signals allows the robot to compute the corresponding motor speeds. The calculated velocities are then sent to the robot to make the corresponding manoeuvre at each sample time. To complement the scenario, multiple tracks are added where the robot can move. The image of a track is also processed by the server.

The performance of the system is highly encouraging. The communication process is fast enough to acquire the image, send it to the server and move the robot according to the velocities received. The hardest task is training the cascade of classifiers. The time can increase exponentially depending on the number of images used in the training. This aspect must be considered when building a similar system that includes many more signals. Future work will include the implementation of different classifiers with other image processing algorithms in the server and the addition of more than one robot to perform exploring and collaboration mapping.

Author Contributions: E. Torres, G. Bricas and E. Fabregas designed and developed the experiments in simulation and in the real environment. E. Torres and E. Fabregas collaborated in the development of the platform and wrote the manuscript. G. Farias, S. Dormido-Canto and S. Dormido performed the theoretical conception, formal analysis, resources, validation, founding and planning of the system.

Funding: This work has been funded by the Chilean Ministry of Education under the Project FONDECYT Regular 1191188, the Spanish Ministry of Economy and Competitiveness under Projects ENE2015-64914-C3-2-R and RTI2018-094665-B-I00, and the Spanish Ministry of Science and Innovation under the Project PID2019-108377RB-C32.

References

1. C. Thorpe, M.H. Hebert, T. Kanade and S.A. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362-373, 1988.
2. M. A. Turk, D.G. Morgenthaler, K.D. Gremban and M. Marra, "VITS-a vision system for autonomous land vehicle navigation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 342-361, 1988.
3. G.N. Desouza and A.C. Kak, "Vision for mobile robot navigation: a survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237-267, 2002.
4. N.J. Nilsson, Shakey the Robot, Technical Report 323, SRI AI International Menlo Park, 1984.
5. D.A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network", Technical Report, Carnegie Mellon Univ., 1989.
6. P. Pan, D.J. Pack, A. Kosaka, A.C. Kak, "FUZZY-NAV: a vision-based robot navigation architecture using fuzzy inference for uncertainty-reasoning", World Congress on Neural Networks, pp. 602-607, 1995.
7. D. Kim, R. Nevatia, "Recognition and localization of generic objects for indoor navigation using functionality", *Image and Vision Computing*, no.16, vol. 11, pp. 729-43, 1998.
8. J. Gaspar, N. Winters and J. Santos-Victor, "Vision-based navigation and environmental representations with an omnidirectional camera", *IEEE Trans on Robotics and Automation*, 16, pp. 890-898, 2000.

9. C. Lee and D. Kim, "Visual Homing Navigation With Haar-Like Features in the Snapshot", *IEEE Access*, vol. 6, pp. 33666-33681, 2018.
10. U. Witkowski, P. Bolte and J. Sitte, "Learning Vision Based Navigation with a Smartphone Mobile Robot", IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Auckland, pp. 1216-1221, 2018.
11. E. Garcia-Fidalgo, A. Ortiz, "Vision-based topological mapping and localization methods: A survey", *Robotics and Autonomous Systems*, vol. 64, pp. 1-20, 2015.
12. A. G. Millard, R. A. Joyce, J. A. Hilder, C. Fleseriu, L. Newbrook, W. Li, McDaid, and D.M. Halliday, "The pi-puck extension board: a raspberry pi interface for the e-puck robot platform", In IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 741-748, 2017.
13. KTeam. Khepera IV User Manual, 2018. Available online: http://ftp.k-team.com/KheperaIV/UserManual/
14. Adept Technology, Inc. Pioneer 3-DX User Manual, 2018.
15. Robotis, Inc. Turtlebot3 User Maunal, 2019. Available online: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
16. F.A.X. Da Mota, M.X. Rocha, J.J.P.C. Rodrigues, V.H.C. De Albuquerque and A.R. De Alexandria, "Localization and Navigation for Autonomous Mobile Robots Using Petri Nets in Indoor Environments", *IEEE Access*, vol. 6, pp. 31665-31676, 2018.
17. X. Gao et al., "Review of Wheeled Mobile Robots' Navigation Problems and Application Prospects in Agriculture," in IEEE Access, vol. 6, pp. 49248-49268, 2018.
18. S. Hsu, S. Chan, P. Wu, K. Xiao and L. Fu, "Distributed Deep Reinforcement Learning based Indoor Visual Navigation", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, pp. 2532-2537, 2018.
19. A. Giusti et al., "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots", *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661-667, 2016.
20. A. Mosavi, A. R. Varkonyi-Koczy, "Integration of machine learning and optimization for robot learning", *Recent Global Research and Education: Technological Challenges*, pp. 349-355, 2017.
21. M. Lakrouf, L. Stanislas, D. Michel, A. Nouara, "Moving obstacles detection and camera pointing for mobile robot applications," In Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering, pp. 57-62, 2017.
22. A. Irawan, M. Yaacob, F. Azman, M. Daud, A. Razali y S. Ali, "Vision-based Alignment Control for Mini Forklift System in Confine Area Operation," International Symposium on Agent, Multi-Agent Systems and Robotics, pp. 1-6, 2018.
23. F. Zaklouta y B. Stanciulescu, "Real-time traffic sign recognition in three stages," *Robotics and autonomous systems*, vol. 62, no. 1, pp. 16-24, 2014.
24. G. Archana, S. Thejas, S. Ramani, S. Iyengar, "Image Processing Approaches for Autonomous Navigation of Terrestrial Vehicles in Low Illumination," 2nd International Conference On Emerging Computation and Information Technologies, pp. 16, 2017.
25. H. Zhang, D. Hernandez, Z. Su y B. Su, "A Low Cost Vision-Based Road-Following System for Mobile Robots," *Applied Sciences*, vol. 8, no. 9, p. 1635, 2018.
26. G. Farias, E. Fabregas, E. Peralta, H. Vargas, S. Dormido-Canto and S. Dormido, "Development of an Easy-to-Use Multi-Agent Platform for Teaching Mobile Robotics", *IEEE Access*, vol. 7, pp. 55885-55897, 2109.
27. E. Fabregas, G. Farias, E. Peralta, J. Sanchez, S. Dormido, "Two Mobile Robots Platforms for Experimentation: Comparison and Synthesis", International Conference on Informatics in Control, Madrid, Spain, 2017.
28. G. Farias, E. Fabregas, E. Peralta, H. Vargas, G. Hermosilla, G. Garcia, S. Dormido, "A neural network approach for building an obstacle detection model by fusion of proximity sensors data", *Sensors*, vol. 18, no. 3, pp. 1-18, 2018.
29. E. Peralta, E. Fabregas, G. Farias, H. Vargas, S. Dormido, "Development of a Khepera IV Library for the V-REP Simulator", *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 81-86, 2016.
30. G. Farias, E. Fabregas, E. Peralta, E. Torres, S. Dormido, "A Khepera IV library for robotic control education using V-REP", *IFAC-PapersOnLine*, no. 50, vol. 1, pp. 9150-9155, 2017.
31. G. Farias, E. Torres, E. Fabregas, H. Vargas, S. Dormido-Canto and S. Dormido, "Navigation control of the Khepera IV model with OpenCV in V-REP simulator", IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA), Concepción, Chile, pp. 1-6, 2018.

32. Pulli, Kari, et al. "Real-time computer vision with OpenCV", *Communications of the ACM*, no. 55, vol.6, pp. 61-69, 2012.
33. Team OpenCV Developers. OpenCV, 2018. [online]. Available: https://opencv.org/
34. H. Kim et al., "Vision-Based Real-Time Obstacle Segmentation Algorithm for Autonomous Surface Vehicle", *IEEE Access*, vol. 7, pp. 179420-179428, 2019.
35. L. Huitao, "Optimization design of cascade classifiers", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, pp. 480-485, vol. 1, 2005.
36. S.B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques", *Emerging artificial intelligence applications in computer engineering*, no. 160, pp. 3-24, 2007.
37. P. Viola, and M. Jones, "Rapid object detection using a boosted cascade of simple features", Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai, HI, USA, 2001.
38. S.Y. Chen, Y.H. Huang, "Research and application of open source video server MJPG-streamer", *Electronic Design Engineering*, no. 5, pp. 172-176, 2012.
39. K. Lu, J. Li, X. An, H. He, "Vision sensor-based road detection for field robot navigation", *Sensors*, vol. 15, no. 11, pp. 29594-29617, 2015.

Sample Availability: Samples of the compounds ..... are available from the authors.