A *Khepera IV* library for robotic control education using *V-REP*

G. Farias * **E. Fabregas** ** **E. Peralta** * **E. Torres** * **S. Dormido** **

* *Pontificia Universidad Católica de Valparaíso, Av. Brasil, 2147, Valparaíso, Chile (e-mails: gonzalo.farias@pucv.cl, emmanuel.peraltah@gmail.com, enrique_torres1994@hotmail.com)*
** *Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia, Juan del Rosal, 16, 28040, Madrid, Spain (emails: efabregas@bec.uned.es, sdormido@dia.uned.es)*

Abstract: This paper describes a new module to create advanced simulations with the *Khepera IV* mobile robot in *V-REP* simulator. The library, called *KH4VREP*, allows users to add a *Khepera IV* model to a new or an existing *V-REP* simulation. The *KH4VREP* library depicts a graphical representation of the *Khepera IV* model, and also provides several methods to programmatically read and manipulate the sensors and actuators of the robot. The visual design of the model has been developed using Autodesk Inventor. The library provides functionality to test the mobile robot under different control problems such as: position control, trajectory tracking, path following, obstacle avoidance, and multi-robot experiments with formation control.

Keywords: Khepera IV, Control engineering simulations, V-REP simulator.

1. INTRODUCTION

During the last years, robotics has been introduced in education at all levels. Robotics is a multidisciplinary subject. It expands to several engineering and scientific disciplines such as electrical engineering, computer science, control engineering, and mechanical among others. That is why robotics is very interesting and versatile from a pedagogical point of view Farias et al. (2015).

Robotics is also a vast area, it has a large number of branches. Mobile robotics is the branch that deals with robots that are not physically fixed during their operation. In the literature a mobile robot is defined as: "an automatic machine that is capable of locomotion" Peralta et al. (2016). The students can see this kind of robots as just attractive toys, but with mobile robots, students can analyze, test and understand fundamental concepts that are difficult to explain from a theoretical point of view Fabregas et al. (2011); Neamtu et al. (2011); Merdan et al. (2016).

In most research fields, simulators are very important to test theories, ideas, and designs before the real implementation. But in the field of robotics, these simulators are even much more important since advanced robots are still very expensive. Currently, there are many simulators for different robotic branches with very sophisticated features. Among others, it can be found the following popular simulators: *Webots* (Michel (2004)), *Gazebo* (Koenig and Howard (2004)), *RFCSIM* (Fabregas et al. (2014)) and *V-REP* (Coppelia Robotics GmbH (2015)). These simulators have several specific characteristics, but in general all of them are well made and have a good performance. Some of these platforms have licenses that can be used free of charge for educational purposes. Most of these simulators include models of the best known and commercialized robots. In some cases the developments can be exported to the real robot for testing after using the simulator.

The *V-REP* simulator deserves a special mention for being one of the most widely used for pedagogical purposes today. This simulator is a versatile and scalable framework for creating 3D simulations in a relatively short period of time (Rohmer et al. (2013)).

V-REP has an integrated development environment (IDE) that is based on a distributed and script-driven architecture: each scene object can have an embedded script attached, all operating at the same time, in a threaded or non-threaded fashion. *V-REP* comes with a large number of examples, models of robots, sensors and actuators to create a virtual world and to interact with it in running time. New models can be also designed and added to *V-REP* to implement customized simulation experiments.

This paper is a continuation of the previous work of the authors Peralta et al. (2016). The paper describes the library *KH4VREP*, which allows users to add a *Khepera IV* robot model to a new or existing *V-REP* simulation. The model has been encapsulated with many built-in functions to manipulate and control the robot in very easy way. This new version includes several examples, such as: a new algorithm for obstacles avoidance (*VFH*, Ulrich and Borenstein (2000)); and some multi-robot experiments (with several kheperas or other existing models in *V-REP*) to perform formation control.

The remainder of the paper is organized as follows: Section 2, presents some characteristics of the *V-REP* simulator and the *Khepera IV* robot; Section 3 describes the library *KH4VREP*; Section 4 shows the implementation and the results of some test experiments developed with the library; and Section 5 presents the main conclusions and future works.

2. V-REP AND KHEPERA IV ROBOT

The Virtual Robot Experimentation Platform (*V-REP*) is a versatile and scalable framework to develop 3D simulations in a relatively short period of time. This simulator was created in 2010 and has been growing during the last years. Nowadays, this is one of the most used simulators for education in the robotics field (with a free licence for educational purposes). As its authors say: "*V-REP is the Swiss army knife among robot simulators.*"

V-REP has an integrated development environment (IDE) that is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plug-in, a Robot Operating System (ROS) node, a remote API client, or a custom solution. The controllers can be written in different languages: *C/C++*, *Python*, *Java*, *Lua*, *Matlab*, *Octave* or *Urbi* (Coppelia Robotics GmbH (2015)). *V-REP* comes with a large number of examples, models of robots, sensors and actuators to create a virtual world and to interact with it in running time.

2.1 Khepera IV robot

The *Khepera IV* robot is the most recent development of the Swiss company K-Team (2015). This version is newest generation of Khepera family. From its first version, these robots have been designed with research and educational purposes in mind.

It offers wifi and bluetooth communications, color camera, USB host, a large number of advanced features such as: 8 infra-reds sensors, 5 ultrasonics sensors, accelerometer, gyroscope, microphone, a large duration battery (5 hours), loudspeaker, 3 top RGB LED, improved odometry and precision. The CPU is a 800MHz ARM Cortex-A8 Processor (with linux core running) and additional microcontroller for peripherals management.

The robot can be programmed from a Linux-based operating system and C language. *Khepera IV* has a modular configuration, with a cylindrical shape to minimize the damage and negative effects in its parts under collision conditions. *Khepera* is a differential wheeled robot whose movement is based on two separately driven wheels placed on each side of its body. The device can be used for different experiments such as: line following, obstacle detection and avoidance in a dynamic environment, advanced signal processing, motion control, wireless communication, image processing, formation control, and many other advanced topics. Undoubtedly, this robot is a powerful tool for teaching robotics.

3. KH4VREP LIBRARY

To use the library you need to include it in *V-REP* simulator. To do it you need to download the file *KheperaIV.ttm* from: `https://github.com/EAPH/K4_Model_VREP` and copy it in the following path: `..\V-REP3\V-REP_PRO_EDU\models\robots\mobile`. Once the file is copied, it will appear in the list of models of *V-REP*, as it can be seen on the left side of Figure 1.

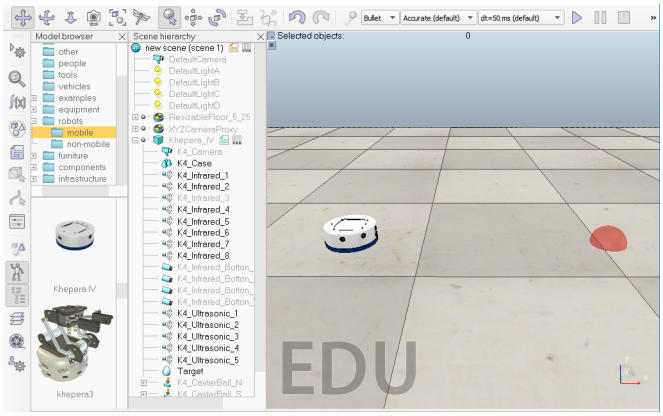

Fig. 1. Integration of the model in *V-REP*

To add the model to the simulation you need to drag and drop the robot to the workspace. The result is shown on the right side of the Figure 1.

The library is titled *KH4VREP* and it is divided into two parts: 1) the graphical model of the *Khepera IV* robot (visual components); and 2) a set of software functions to carry out some specific tasks.

3.1 Visual components

The *Khepera IV* is a rigid body with a differential behavior, due to it is equipped with two wheels that are driven by DC motors with encoder and gearbox. The minimum controlled speed of the robot is 3 mm/s and the maximum speed is about 813 mm/s. In the case of the caster wheels, both are considered as fixed spheres in the model. Note that the gyroscope and accelerometer sensors were not included in the model since that information is given directly by the *V-REP* simulator for each included model.

To design and to implement the model, all visual parts of the robot (case, base, wheels, etc...) were modeled using Autodesk Inventor and assembled in *V-REP*. The obtained prototype was imported to *V-REP* as *.stl* format. *V-REP* allows to add physical properties and dynamics to this kind of prototypes. The model also has 5 ultrasonic sensors, 8 infrared sensors (eight around the case and four in the bottom of the robot to perform line following), one color camera, two wheel motors, and two caster wheels. All these sensors and actuators were added and configured using existing elements of *V-REP*. Figure 2 shows the assembling process of the robot in *V-REP*.

3.2 Software components

The software components of the library are in a script associated to the model. This script is a main thread with an infinite loop that contains calls to functions that carry out some specific tasks during the simulation. The following code segment shows the structure of the main function for the example of position control.

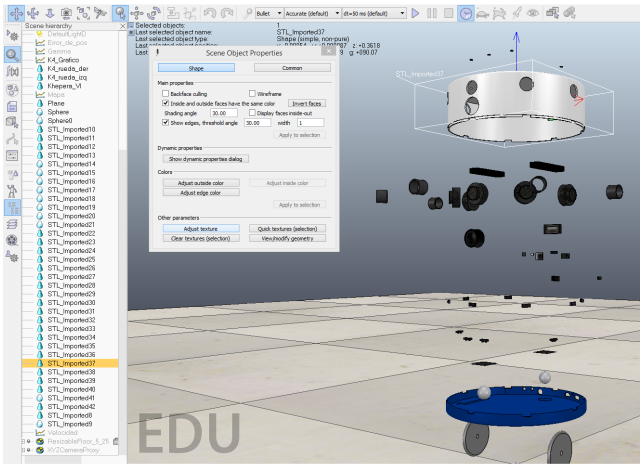

Fig. 2. Assembling the model of the *Khepera IV*

```
1 setRobot(vmax,wmax,L,xi,yi,thetai)
2 threadFunction=function()
3  while (loop==1) do
4     --Get position and orientation of the robot
5     xc,yc,theta=getRobotPosition()
6     --Get target position
7     xp,yp=getTargetPosition()
8     --Apply position control
9     v,w=positionControl(xp,yp,xc,yc,theta)
10    --Update robot velocities
11    updateVelocities(v,w)
12 end
13 end
```

The code starts by setting the initial conditions, for example: function *setRobot* to establish the initialization parameters of the robot (line 1). Line 2 is the definition of the thread main function (*threadFunction*). In line 3 starts the infinite while loop. Line 5 obtains the current position and orientation of the robot with the function *getPosition*. Line 7 obtains the coordinates of the target point with the function *getTarget*. Line 9 executes the position control of the robot with function *positionControl*. Line 11 updates the angular and linear velocities of the robot model.

3.3 Position control experiment

This experiment is to drive the wheeled mobile robot from its current position to a predefined target point. This problem has been widely studied during the last years, due to the kinematic model of these robots may seem deceptively simple, but nonholonomic constraints introduce a challenging problems in the designing of the control law (Fabregas et al. (2016)). Figure 3 shows the blocks diagram of this experiment.

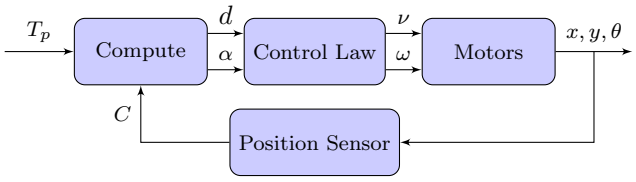

Fig. 3. Block diagram of the position control problem

Figure 4 shows the variables involved in this experiment. The robot tries to minimize its orientation error, $\theta_e = \alpha - \theta$, where α is the current angle to the target point and θ is the current orientation of the robot. At the same time, the robot tries to reduce the distance to the target point ($d \approx 0$).

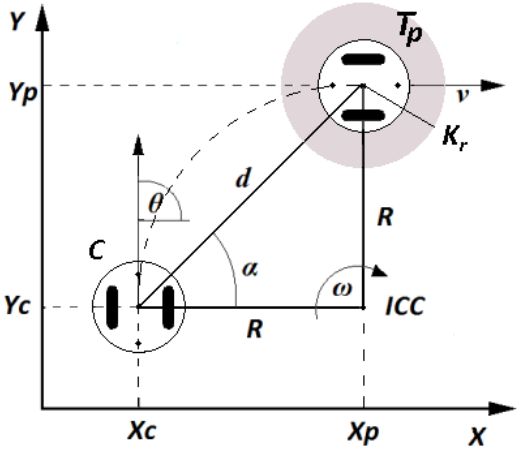

Fig. 4. Position control experiment

Equations (1) and (2) are implemented in the block *Compute*; by using as reference the target point (T_p) and the current position of the robot (C).

$$d = \sqrt{(y_p - y_r)^2 + (x_p - x_r)^2} \tag{1}$$

$$\alpha = \tan^{-1}\left(\frac{y_p - y_r}{x_p - x_r}\right) \tag{2}$$

With the outputs of the block *Compute* the block *Control Law* calculates the linear velocity (ν) and the angular velocity (ω) of the robot as it is shown in equations (3) and (4) (Villela et al. (2004)).

$$\nu = \begin{cases} \nu_{max} & if\ |d| > k_r \\ d\left(\dfrac{\nu_{max}}{k_r}\right) & if\ |d| \leq k_r \end{cases} \tag{3}$$

$$\omega = \omega_{max} \sin(\alpha - \theta) \tag{4}$$

where ν_{max} is the maximum linear velocity, k_r is the radius of a docking area (around the target point) and ω_{max} is the maximum angular velocity of the robot.

The following code segment represents the implementation of the function *positionControl* in the *KH4VREP* library. Note that this function is called in line 9 of the main function in the previous example.

```
1 positionControl=function(xp,yp,xc,yc,theta)
2     d = math.sqrt(((xp-xc)^2)+((yp-yc)^2))
3     alpha = math.atan2(yp-yc,xp-xc) --Target angle
4     Oc = alpha-theta                --Error angle
5     if(d>=0.05) then
6        w=Wmax*math.sin(Oc)    --Angular velocity
7        v=(vmax/kr)*d          --Linear velocity
8        if(v>vmax) then --Linear velocity saturation
9          v=vmax
10       end
11    else v=0, w=0
12    end
13    return v,w  --Values returned
14 end
```

Line 1 is the definition of the *positionControl* function which receives as parameters the coordinates of the current target point ($xp; yp$) and the current position ($xc; yc$) and

orientation (*theta*) of the robot. Lines 2 and 3 calculate equations (1) and (2). Line 4 obtains the orientation angle error. Lines 6 calculates equation (4) and line 7 calculates equation (3). Lines 8 to 10 are the saturation of the calculated linear velocity (ν). Lines 12 and 13 set the velocities to zero if the robot is very close to the destination point (less than 0.05 m). The function returns the values of the velocities in line 15.

3.4 Obstacles avoidance experiment

The obstacles avoidance problem has been widely studied using different techniques. Note that this problem is a next step of the position control. The robot must be able to avoid the obstacles that appear on its way to the destination. To do that, the block *Avoidance* is added after the *Control Law* block. This block calculates new velocities (ν' and ω') if the sensors detect obstacles. In the absence of obstacles ν'=ν and ω'=ω. Figure 5 shows the block diagram of this experiment.

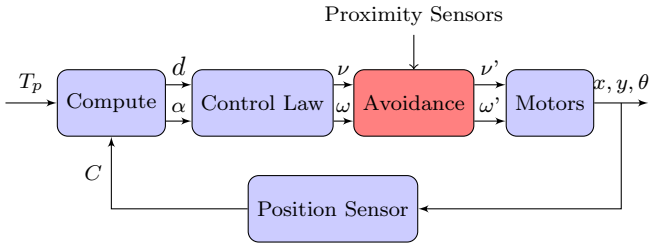

Fig. 5. Control block diagram with obstacle avoidance

The block *Avoidance* can be implemented with different algorithms depending of the obstacles sensors and its location on the robot. In this case, to test the library two algorithms have been implemented: Braitenberg (Yang et al. (2006)) and (*VFH*) Vector Field Histogram (Ulrich and Borenstein (2000)).

Braitenberg algorithm. This algorithm creates a weighted matrix that converts the sensor inputs into motors speeds. This matrix is a two-dimensional array with the number of columns corresponding to the eight sensors, and the rows corresponding to the two motors. Equation (5) represents such matrix, where W_{LS_1} is the weight of sensor S_1 in the speed of the left motor. Equation (6) represents the values of the sensors at each time.

$$W = \begin{pmatrix} W_{LS_1} & W_{LS_2} & \dots & W_{LS_8} \\ W_{RS_1} & W_{RS_2} & \dots & W_{RS_8} \end{pmatrix} \tag{5}$$

$$S = \begin{pmatrix} S_1 & S_2 & \dots & S_8 \end{pmatrix}^T \tag{6}$$

With these matrices, the velocities for each motor are calculated as shown in equation (7), where (S_{max}) is the maximum value of the proximity sensors if no obstacle is detected.

$$\nu_{L,R} = W * (1 - (S/S_{max})) \tag{7}$$

Velocities ν' and ω' are calculated from ν_R and ν_L using the differential model equations. The following code segment shows the implementation of this algorithm.

```
1 braitenbergControl=function()
2  if(S[1]+S[2]+S[3]+S[4]+S[5].. --Obstacle detected
3       ..+S[6]+S[7]+S[8]<=Smax*8) then
4    for i=1,8,1 do
5      Vl=Vl+WLS[i]*(1-(S[i]/Smax))
6      Vr=Vr+WRS[i]*(1-(S[i]/Smax))
7    end
8  v=(Vr+Vl)/2
9  w=(Vr-Vl)/L
10 end
11 return v,w
12 end
```

The function *Braitenberg* is called after the function *positionControl* in the main *threadFunction*.Line 1 is the definition of this function. Line 2 asks for the obstacle detection. If one of the 8 sensors does not has the maximum value, is because an obstacle has been detected. Lines 5 and 6 implement equations (5), (6) and (7). Lines 8 and 9 calculate ν and ω from the differential model. Line 11 returns the values of the calculated velocities.

VFH algorithm. This method allows a fast and smooth motion of the robot avoiding obstacles in a dynamical environment. The speed is the maximum in the absence of obstacles. The robot tries to maintain this speed during the trajectory unless being forced to slow down by the *VFH* algorithm to the instantaneous speed ν and to change the angular velocity according to equations (8) and (9).

$$\nu = \nu' \left(1 - \frac{\omega}{\omega_{max}}\right) \tag{8}$$

$$\omega = \omega_{max} \sin(\theta_e) \tag{9}$$

where ω_{max} is the maximal allowable angular velocity for the robot that will be achieved when the orientation error, $\theta_e = \pm 90°$, and ν' is defined in equations (10 and 11).

$$\nu' = \nu_{max} \left(1 - \frac{h_c''}{h_m}\right) \tag{10}$$

$$h_c'' = min(h_c', h_m) \tag{11}$$

where ν_{max} is the maximum linear velocity, h_m is a constant empirically determined to cause a reduction in the speed in presence of obstacles, and h_c' is the value of the histogram of the distance between the robot and the obstacles. If $h_c' > 0$ then an obstacle lies ahead of the robot. Large values of h_c' means a bigger obstacle lies ahead or an obstacle is very close to the robot. In both cases a change of the direction is required. This means that a reduction in the linear velocity is needed to turn to the new direction.

3.5 Multi-robot experiment

This experiment is a formation maneuver of a group of robots. Where one of them is the leader and the rest are the followers. The position of the leader is controlled using the aforementioned algorithm. The followers also control their positions, but taking the position of the leader as reference to make a formation around him. For example a circle with the leader located at the center (Lawton et al. (2003)).

The followers have three parameters in the function *targetPosition* to modify their destination points: 1) the coordinates of the leader robot; 2) the distance to the leader in the formation; and 3) the angle with respect to the leader's position. This configuration allows different formation around the leader robot. The coordinates of the

target point (x_p, y_p) of the followers in the formation are obtained in equation (12).

$$\begin{cases} x_p = x_L + rcos(\beta + \theta) \\ y_p = y_L + rsin(\beta + \theta) \end{cases} \quad (12)$$

where the position of the leader robot is (x_L,y_L). r is the distance of the follower to the leader (for example: radius of the circle). β is the angle to the leader and θ is the orientation of the leader robot.

The following code segment shows an example of the implementation of this function for a follower robot located at 0.5 meters and 180° with respect to the leader robot.

```
1 xp,yp=getTargetPosition('Khepera_IV0', 0.5, 180)
2 ...
3 targetPosition=function(obj,objDist,objAng)
4  distanceMaster=objDist
5  angleMaster=objAng*math.pi/180
6  target=simGetObjectHandle(obj)
7  simSetObjectParent(target,-1,true)
8  tar=simGetObjectPosition(target,-1)
9  oriTar=simGetObjectOrientation(target,-1)
10 xp=tar[1]+distMaster*math.cos(angMaster+oriTar[3])
11 yp=tar[2]+distMaster*math.sin(angMaster+oriTar[3])
12 return xp,yp
13 end
14 ...
15 v,w=positionControl(xp,yp,xc,yc,theta)
```

Line 1 is the invocation of this function from the main thread. Line 3 is the definition of the function with three parameters: 1) leader robot name; 2) distance to the leader robot in the formation; and 3) angle with respect to the leader robot in the formation. Lines 4 to 9 obtain the parameters of the leader robot (handle, position and orientation). Lines 10 and 11 implement equation (12). Line 12 returns the target coordinates (x_p, y_p). After that, in the main thread the position control is executed (line 15).

Table 1 shows a summary of implemented functions and their different tasks.

Table 1. KH4VREP Library methods

Methods	Characteristics
setRobot	Sets the parameters of the robot.
getPosition	Obtain the position of the robot.
plot	Creates a 2D line plot of the data.
getTarget	Obtain the target point.
positionControl	Controls the position of the robot.
braitenbergControl	Executes the Braitenberg algorithm.
updateVelocities	Updates the velocities of the robot.

4. RESULTS

In this section the implemented control problems are presented to test the library: obstacles avoidance and multi-robots. Note that these experiments implement the position control of the robot.

4.1 Collision Avoidance (Braitenberg vs VFH Algorithms)

Figure 6 shows a sequence of the comparison of these two algorithms. On the left side the robot implements the position control experiment with the Braitenberg algorithm for obstacles avoidance and on the right side, with the *VFH* algorithm. In both cases, the robot has to reach its destination point (red semi-sphere). The configuration in both cases is similar: a wall (in green and black colors) and an obstacle in (yellow and black colors) on the way to the target point.

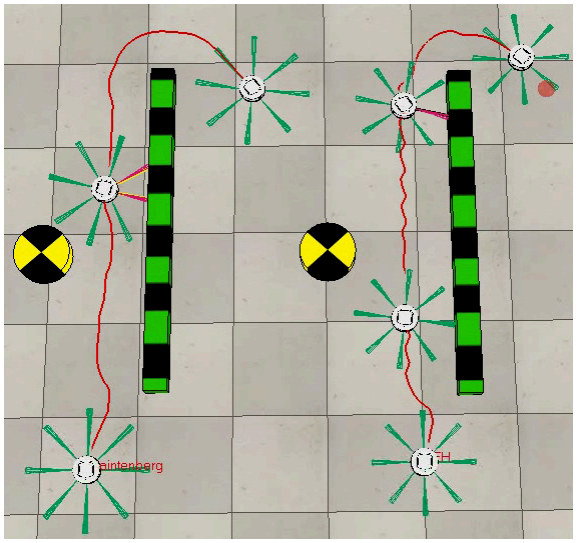

Fig. 6. Braitenberg vs VFH

As can be seen with the Braitenberg algorithm the robot reaches the destination point in a much more direct and smoothed way. That is why the robot reaches the destination point in less time. In the case of *VFH*, the robot moves with a zigzag path because the built histogram has "blind zones" between the sensors due to the localization of the sensors around the robot. For this reason the histogram is not continuous and the algorithm does not work properly as expected.

4.2 Multi-robots experiment

Figure 7 shows an example of the multi-robot experiment including obstacles avoidance (Braitenberg). In this case, five robots have been used (one leader and four followers). The configuration shows obstacles (in light gray color) with different shapes and sizes.

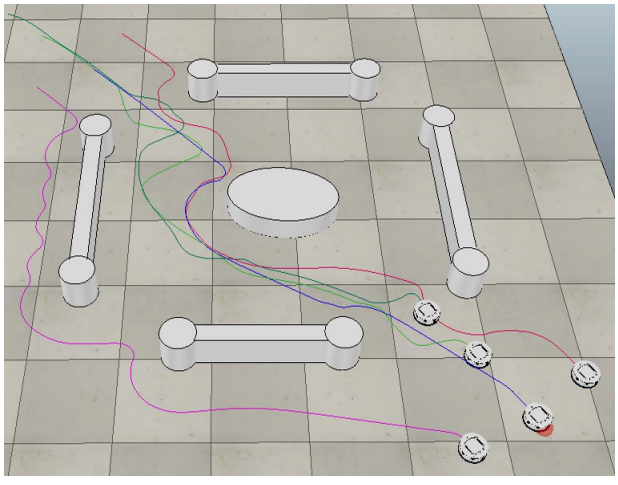

Fig. 7. Multi-robots performing formation control

The goal is to make a formation with cross form with the leader robot in the center as reference. All robots start stopped occupying their respective positions in the formation. Then the leader tries to reach its destination point avoiding the obstacles that appear on its way. The traces of the positions of the robots show the trajectories followed to reach the destinations. As can be seen, formation is lost during the displacement due to two aspects: 1) the targets points of the followers are relative to the position of the guide and it is moving; 2) follower avoid the obstacles that appear in their path.

Another example of multi-robots experiment have been implemented to test the integration of the library with the existing models of *V-REP*. The experience consists in a *Khepera IV* that performs a position control experiment and a quadcopter that must follows the *Khepera IV* using its position as reference (x, y) maintaining constant height (z) relative to the ground. Figure 8 shows an image of this experiment: the on-board camera of the *Khepera* (bottom left side) and the on-board camera of the quadcopter (bottom right side). As can be seen the quadcopter follows the *Khepera IV* robot in acceptable way.

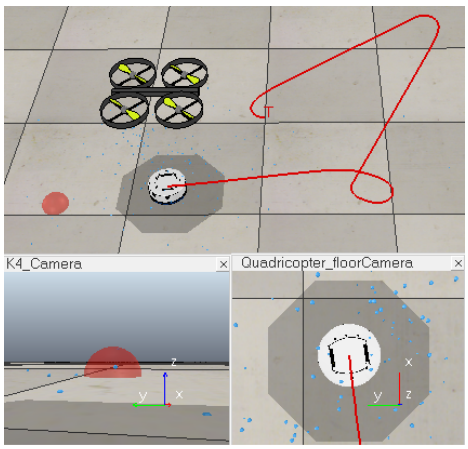

Fig. 8. A quadcopter follows a Khepera IV that performs position control

5. CONCLUSION

In this paper the library *KH4VREP* for *V-REP* simulator has been presented. The library is an encapsulation of the model of the *Khepera IV* and includes built-in methods to manipulate the robot, and also a set of simulations to show many control problems with mobile robots. The result of this development is a ready to use tool that allows the implementation of experiments with these robots in an interactive and attractive environment. The library allows to test and understand theoretical concepts in a dynamic way in order to teach engineering control.

In the near future, the library will be included in a pilot platform that has been developed in our laboratories. The main objective of the future work will be to design advanced control algorithms using the library and the simulation environment. Among other, multi-agents experiments with consensus control problem and event-based communications between the robots could be included in the library.

ACKNOWLEDGEMENTS

This work has been funded by the Chilean Ministry of Education under the Project FONDECYT 1161584, the Projects DPI2012-31303 and DPI2014-55932-C2-1-R of the Spanish Ministry of Economy and Competitiveness.

REFERENCES

Coppelia Robotics GmbH (2015). V-REP simulator.

Fabregas, E., Farias, G., Dormido-Canto, S., and Dormido, S. (2014). RFCSIM simulador interactivo de robótica móvil para control de formación con evitación de obstáculos. In *XVI Congreso Latinoamericano de Control Automático, Cancún, Quintana Roo, México*.

Fabregas, E., Farias, G., Dormido-Canto, S., Dormido, S., and Esquembre, F. (2011). Developing a remote laboratory for engineering education. *Computers & Education*, 57(2), 1686–1697.

Fabregas, E., Farias, G., Dormido-Canto, S., Guinaldo, M., Sánchez, J., and Dormido Bencomo, S. (2016). Platform for teaching mobile robotics. *Journal of Intelligent & Robotic Systems*, 81(1), 131–143.

Farias, G., Muñoz, D., Gómez-Estern, F., De la Torre, L., Sánchez, C., and Dormido, S. (2015). Adding automatic evaluation to interactive virtual labs. *Interactive Learning Environments*, 1–21.

K-Team (2015). K team mobile robotics.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2149–2154. Sendai, Japan.

Lawton, J., Beard, R., and Young, B. (2003). A decentralized approach to formation maneuvers. *IEEE Transactions on Robotics and Automation*, 19(6).

Merdan, M., Lepuschitz, W., Koppensteiner, G., and Balogh, R. (2016). *Robotics in Education*. Springer.

Michel, O. (2004). WebotsTM: Professional mobile robot simulation. *arXiv preprint cs/0412052*.

Neamtu, D., Fabregas, E., Wyns, B., De Keyser, R., Dormido, S., and Ionescu, C. (2011). A remote laboratory for mobile robot applications. In *18th IFAC World Congress*, volume 44, 7280–7285.

Peralta, E., Fabregas, E., Farias, G., Vargas, H., and Dormido, S. (2016). Development of a Khepera IV Library for the V-REP Simulator. In *11th IFAC Symposium on Advances in Control Education ACE 2016*, volume 49, 81–86. Bratislava, Slovakia.

Rohmer, E., Singh, S., and Freese, M. (2013). V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1321–1326.

Ulrich, I. and Borenstein, J. (2000). VFH*: Local obstacle avoidance with look-ahead verification. In *ICRA*, 2505–2511.

Villela, V.J.G., Parkin, R., Parra, M.L., González, J.M.D., Liho, M.J.G., and Way, H. (2004). A wheeled mobile robot with obstacle avoidance capability. *Tecnología Y Desarrollo*, 1(5), 159–166.

Yang, X., Patel, R., and Moallem, M. (2006). A fuzzy braitenberg navigation strategy for differential drive mobile robots. *Journal of Intelligent and Robotic Systems*, 47(2), 101–124.