2016 IEEE Conference on Control Applications (CCA)
Part of 2016 IEEE Multi-Conference on Systems and Control
September 19-22, 2016. Buenos Aires, Argentina

Teaching control in mobile robotics with *V-REP* and a *Khepera IV* library

Ernesto Fabregas*, Gonzalo Farias†, Emmanuel Peralta†, Héctor Vargas†, Sebastián Dormido*
*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
†Pontificia Universidad Católica de Valparaíso
Av. Brasil, 2147, Valparaíso, Chile.
Emails: gonzalo.farias@pucv.cl, emmanuel.peraltah@gmail.com, hector.vargas@pucv.cl

Abstract—**The advances of information and communication technologies have impacted in control education. Virtual laboratories are increasingly been used to enhance the way that students interact with simulations. High degree of visualization and interaction offered by modern computers open the opportunity to teach theory fundamentals with a more natural approach. This work describes the use of the robot simulator V-REP and the Khepera IV library to teach interesting control problems with mobile robots. The Khepera IV library supports the new functionalities of the most recent version of the wheeled robot Khepera, which allows creating advanced 3D simulations in the V-REP environment. The article also explains how instructors can use this development to teach classical control problems in mobile robotics. In particular, the theory and practice of the problems for position control, trajectory tracking, path following and the obstacle avoidance are given in detail.**

Keywords*—*Control education, Computer-based, teaching environments

I. Introduction

During the last years, the universities have modified their curricula to adapt them to the new information and communication technologies. The main idea have been to incorporate modern techniques to introduce important concepts in an innovative way from the academic and didactic points of view. In this context, the control engineering education has introduced the robotics in this novel teaching-learning process. Because robots can be seen as attractive toys from a recreational and playful point of view. But they can also be seen as an excellent example of interesting control problems.

Robotics is a multidisciplinary field that includes a combination of computer science, electronics, mechanics, artificial intelligence, control, among other topics. This makes it very versatile from the pedagogical point of view. Because it can be introduced into education to study a wide range of concepts in different levels, depending of the complexity of the problems to study.

Mobile robotics is the branch concerned with robots that have the capability to move around their environment and are not fixed to one physical location. This capability allows to introduce challenging experiments (position and formation control, trajectory tracking, path following, navigation, exploration, obstacles avoidance, etc.). All of them have in common that robots use the environment information (obtained with sensors) to make decisions. These decisions will depend on the experiment as is shown later.

Nowadays, the advanced robots are still too expensive and they are exposed to be damage during the experimentation in the labs. That is why the simulators are very important in this field [1]. The development of virtual laboratories using these simulators offers significant benefits for robotics education. Using such virtual labs, students can test and understand, at any time and pace, and from any place, concepts that are not easy to follow in the classroom.

Currently, there are many simulators for different areas of robotics. For example: *ARGoS* [2], *Webots* [3] and *V-REP* [4], to mention only the most used. Some of these platforms have licenses that can be used free of charge for educational purposes.

The *V-REP* simulator deserves a special attention 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 [5]. *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 designed and added to *V-REP* to implement customized simulation experiments.

This paper describes the implementation of some interesting control problem with wheeled mobile robots. The robot simulator *V-REP* and the *Khepera IV* library ([6]) have been used. The remainder of the paper is organized as follows: Section II, presents the control problems used in the virtual environment; Section III presents the implementation and the use of this problems in *V-REP* simulator; Section IV presents the main conclusions and the future works of this research.

II. Control experiments with mobile robotics

Figure 1 shows the block diagram of a typical feedback control loop of a *System*. The main idea is simple: the output of the system (y) must follow the desired reference (r). To do that, a *Controller* must be designed. The *Controller* has to take as input the error (e), which is the difference between the reference (r) and the variable (y_m) measured by the *Sensor*, in order to calculate the control signal (u) that commands the *System*. With this control signal, the *System* should be able to follow the reference and at the same time to reject the possible *Disturbances*.

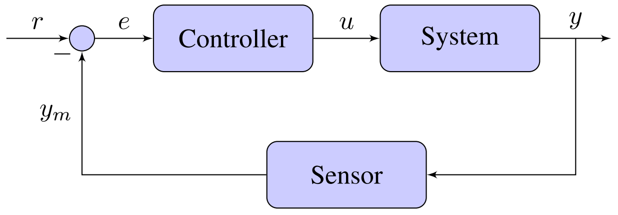

Fig. 1. Feedback control loop block diagram

In this section four control problems with mobile robots, based on the feedback control loop scheme, are shown in detail: *i*) position control, *ii*) trajectory tracking, *iii*) path following and *iv*) obstacles avoidance.

A. Position Control or "Point Stabilization"

This problem is titled position control or point stabilization of a differential wheeled mobile robot. Its objective is to drive the robot from the current position $C(x_c, y_c)$ and orientation (θ) to the target point $T_p(x_p, y_p)$, as is shown in Figure 2. This problem has been widely studied mainly due to the designing of the control law, under nonholonomic constraints, introduces challenging control problems from the academic point of view [7].

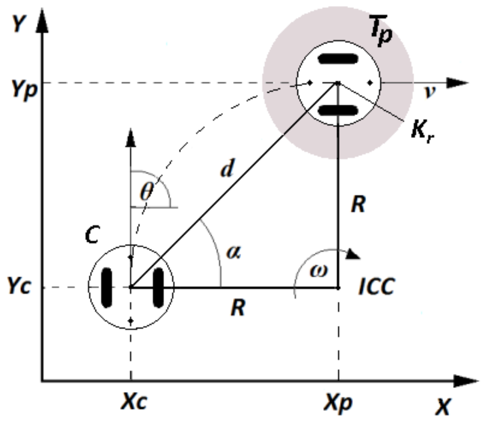

Fig. 2. Position control problem

In order to achieve the control objective, the distance (d) and the angle (α) between the points C and T_p are calculated with Eq. 1 and 2.

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

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

Figure 3 shows the control blocks diagram of this problem. The robot tries to minimize the orientation error, $\theta_e = \alpha - \theta$, and at the same time, to reduce the distance to the target point ($d = 0$). Eq. 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). These two values and the orientation θ are used by the *Control Law* block to obtain the control signals (linear velocity (ν) and the angular velocity of the robot (ω)).

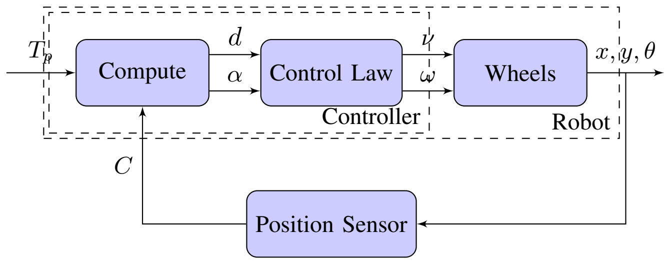

Fig. 3. Diagram of the position control problem

Equations 3 and 4 represent the implementation of the *Control Law* block. 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. Finally, the velocities for the right wheel (V_r) and left wheel (V_l) are computed by Eq.5 ([8]), where L is the distance between the wheels.

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

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

$$V_r = \frac{2\nu + \omega L}{2}, V_l = \frac{2\nu - \omega L}{2} \tag{5}$$

B. Trajectory tracking and path following

In the trajectory tracking problem the robot has to follow a collection of points of a geometric trajectory ([9]). This case can be seen as a loop, where on each iteration, the robot executes the position control problem (it goes from the "current position" to the "current target"). In this problem, the time to reach the trajectory points is not taken into account. An important issue in this case is the arrival angle to the "current target". Because depending on this angle, the way to reach the next point will be more or less efficient to follow the complete trajectory. The angles to the current target (α_{refn}) and the next target (α_{refn+1}) are calculated in Eq. 6.

$$\begin{cases} xd_{n+1} = x_{ref(n+1)} - K_x(x_{refn} - x_c) \\ yd_{n+1} = y_{ref(n+1)} - K_y(y_{refn} - y_c) \\ \theta d_{n+1} = \theta_{ref(n+1)} - K_\theta(\theta_{refn} - \theta_c) \end{cases} \tag{6}$$

The differences between the current position of the robot (x_c, y_c, θ_c) and the second point in the trajectory (xd_{n+1} ,yd_{n+1}, θd_{n+1}) are calculated using Eq. 7.

$$\begin{cases} \triangle x = xd_{n+1} - x_c) \\ \triangle y = yd_{n+1} - y_c) \\ \triangle\theta = \theta d_{n+1} - \theta_c) \end{cases} \tag{7}$$

With these values the velocities ν and ω of the robot are calculated taking into account the next two points, as is shown in Eq. 8. For this control problem the term $T_0 = 1$, which will be explained in the next case.

$$\begin{cases} \nu = \left(\frac{\triangle x}{T_0}\right) cos(\theta_d) + \left(\frac{\triangle y}{T_0}\right) sin(\theta_d) \\ \omega = \frac{\triangle\theta}{T_0} \end{cases} \tag{8}$$

Figure 4 shows an example of this problem. The trajectory is divided in 4 points (P_1, P_2, P_3 and P_4). The robot starts at point P_1 and tries to reach the point P_2. In this iteration, if the controller takes into account the point P_3, the trajectory followed is represented by the continuous line, and the robot reaches P_2 with a suitable orientation to P_3. But, if the controller does not consider P_3, the robot reaches P_2 with a bad orientation to P_3. Note that, in the next iteration, the same consideration has to be taken into account to reach the point P_4.

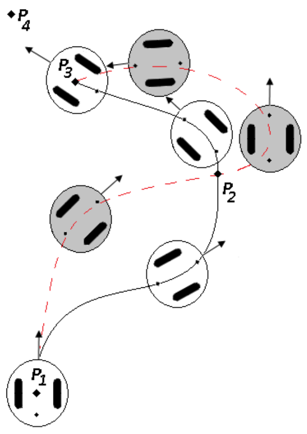

Fig. 4. Trajectory tracking example taking into account next targets (white robot, continuous line) and not (dark robot, dashed line).

The path following problem is similar to the trajectory tracking. The robot has to follow the trajectory, but also taking into account the time to reach each point. The time at which the target point should be reached is KT_0, where K is the index of the target point in the trajectory and T_0 is the period of time. The problem formulation for the path following can be also described by Eq. 8 ([10]).

C. Obstacles avoidance

Once the position of the robot is controlled, it is interesting to add obstacles in the trajectory of the robot and try to avoid them. In the literature there are several ways to address this problem. Most of these algorithms are based on obstacles sensors of the robot, which has a limited vision margin around itself.

In previous works of the authors of this paper, the following methods have been used with encourage results: VFH, VFH^+ and VFH^*([11]). These algorithms build a histogram around the robot using the information of the sensors. The histogram is divided in circular sectors around the robot and marked as free or busy by an obstacle. Then, the control law calculates the velocities (ν, ω) considering the free sectors to avoid the obstacles.

The distribution of the obstacles sensors of the *Khepera IV* robot (each 45°) does not allow to use the mentioned algorithms. That is why the Braitenberg ([12]) algorithm has been implemented here. This algorithm is based on the Braitenberg vehicles ([13]); where the robot's sensors are tied directly to the motors controls and the motors speeds respond to the sensor input directly.

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 number of obstacles sensors (8) and the number of rows corresponding to the number of motors (2). The weights of the matrix are determined empirically depending on the location of the sensors in the robot.The 8 sensors of the Khepera IV robot were numbered clockwise beginning with the front sensor. Equation 9 represent the mentioned matrix, where for example the term W_{LS_1} represents the weight of the sensor S_1 in the speed of the left motor. Eq. 10 represents the values of the sensors at each time.

$$W = \begin{pmatrix} W_{LS_1} & W_{LS_2} & . & . & . & W_{LS_8} \\ W_{RS_1} & W_{RS_2} & . & . & . & W_{RS_8} \end{pmatrix} \tag{9}$$

$$S = (S_1 \quad S_2 \quad . \quad . \quad . \quad S_8)^T \tag{10}$$

With these matrices, the velocities for each motor are calculated as is shown in Eq. 11. Where (S_{max}) represents the maximum value of the sensor.

$$\nu_{L,R} = W * (1 - (S/S_{max})) \tag{11}$$

Figure 5 shows the feedback control loop diagram considering the obstacles avoidance algorithm. The *Obstacles* block represents the Braitenberg algorithm that calculates the new velocities (ν', ω') in presence of obstacles. If none obstacle is detected, the output velocities of the block *Obstacles* are the same that it receives as inputs (ν, ω).

III. Virtual Lab with Mobile Robots Simulations

In this section, the implementations and results of the experiments presented before are described. Besides, a short guide to use these simulations to teach control is provided.

The simulation environment selected is the well known *V-REP* simulator. Which 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

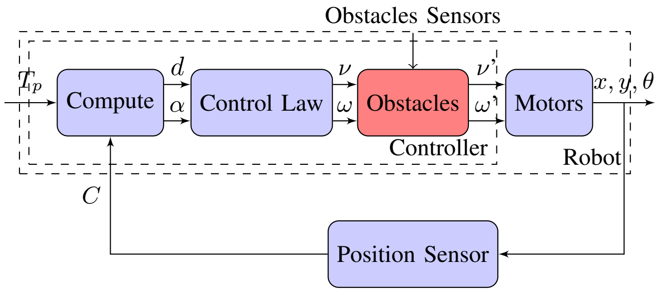

Fig. 5. Control block diagram with obstacle avoidance

models can be designed and added to *V-REP* to implement customized simulation experiments. In a previous work the authors of this paper, developed a library ([6]) to incorporate the *Khepera IV* robot to *V-REP*.

The experiments use the library to implement in *V-REP* the control problems presented before with the *Khepera IV* robot. All the experiments have the following architecture from the source code point of view: a main *while* loop that will be always running during the experiment. In this loop all the actions are taking place by calling functions that make specific tasks. Each iteration of this loop ends when the velocities of the robot are updated. The following fragment of source code shows the main *while* loop.

```
1  threadFunction=function()
2    while (loop==1) do
3      //Get position and orientation of the robot
4      xc,yc,theta=PosOriRobot()
5      //Get target position
6      xp,yp=PosTarget()
7      //Get distances from infrared sensors
8      Flag,Vr,Vl=PosControl(xp,yp,xc,yc,theta)
9      //Update robot velocities
10     UpdateVelocities(Vr,Vl)
11   end
12 end
```

The function of line 4 obtains the position and orientation of the robot. The function of line 6 obtains the coordinates of the target point. The function of line 8 obtains the velocities of the position control algorithm. The function of line 10 updates the wheel velocities in the robot model.

A. Simulation of position control

This experiment is widely used in education to teach control. The following fragment of source code shows the implementation of the function *posControl* and represents the *Controller* block of the Fig. 3.

```
1  posControl=function(xp,yp,xc,yc,th)
2      //Distance to the target point
3      d = math.sqrt(((xp-xc)^2)+((yp-yc)^2))
4      //Angle to the target point
5      alp = math.atan2(yp-yc,xp-xc)
6      //Angle error
7      Oc = alp-th
8      if (d>=0.005 or Oc>=0.087) then
9          //Equation 3
10         V=(Kp*d)
11         if V>Vmax then
12             V=Vmax
13         end
14         //Equation 4
15         w=(Wmax*math.sin(Oc))
16         //Differential Velocities
17         Vr=(2*V+w*L)/2
18         Vl=(2*V-w*L)/2
19         Flag=0
20     else  //Robot must to stop
21         Vl=0
22         Vr=0
23         Flag=1
24     end
25     return Flag,Vr,Vl
26 end
```

The function *posControl* receives as parameters the position and orientation of the robot, and the target point. In the lines 3 and 5, the block *Compute* of Fig. 3 is implemented. While, lines 5 to 12 calculate the *Control Law* block of the diagram (Eq. 3 and Eq. 4).

Figure 6 shows an overhead view of the scene of this simulation. The red circle is the target point (T_p), so the robot has to reach this point. At the beginning, the target is situated in the back of the robot. The red line represents the trajectory that the robot performs to reach the target point in the simulation.

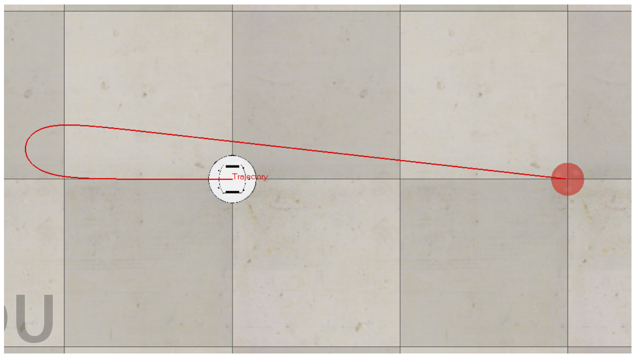

Fig. 6. Position control simulation

Note that in this control law, the linear velocity (ν) is proportional to the distance to the target. Note also that, in a differential robot, ν and ω are related by the turning radius. Thus, when the robot is far from the target (at the beginning of the simulation), the linear velocity (ν) is greater than the angular velocity (ω), and the Khepera can go fast, but it can not turn. After that, when the linear velocity is saturated, the robot starts to turn and move toward the target point.

Figure 7 shows the plot of the position of the robot for this simulation (where the time is in seconds and the distance is in meters). As can be seen, at the beginning the distance to the target is $1.2m$. When robot stars to moves it goes away from the target until $1.6m$. In this point the distance to the target starts to be reduced until the robots reach the target and the distance is equal to 0.

In order to use this virtual lab in the classroom, first the instructor should explain the theory of the control problem to the students. After that, the students should load the simulation in *V-REP*. This file should have everything needed to run the simulation, excepting the section of the code where the control law (*posControl*) is implemented. Thus, students should complete that part in the simulation in order to execute the position control experiment successfully. Note

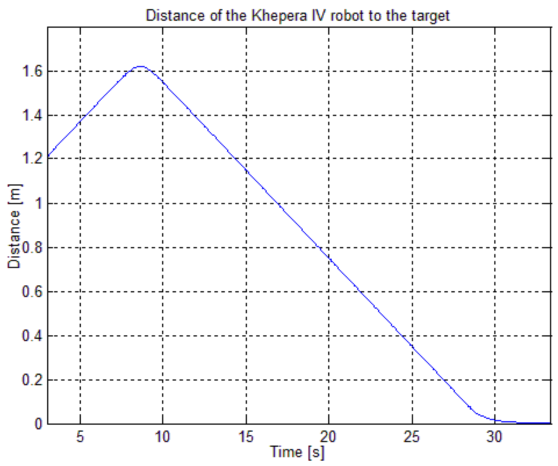

Fig. 7. Position control experiment data

that *posControl* should use the current position $C(x_c, y_c)$ and orientation (θ) of the robot, and the target point $T_p(x_p, y_p)$, in order to compute the linear (ν) and angular (ω) velocities, and to return the differential velocities of each wheel.

B. Simulation of trajectory tracking

This experiment also implements the main *while* loop explained before which is titled *threadFunction*. In this case the first line calls a function that calculates a *Lissajous* curve that will be used as reference. The next line obtains the position of the robot with the function *PosOriRobot*. Then, the function *TrajTracCont* is called to control the position of the robot in order to follow the trajectory. This function takes the current position of the robot and the next two target points of the trajectory in order to calculate the wheel velocities. The following fragment of code shows this function.

```
1  TrajTracCont=function(Xr,Yr,Xr1,Yr1,xc,yc,Odp,Od1p)
2   //Distance to the target
3   d=math.sqrt(((Xr-xc)^2)+((Yr-yc)^2))
4   //Angle to the target
5   Od=math.atan2(Yr-yc,Xr-xc)
6   //Angle to the next target
7   Od1=math.atan2(Yr1-yc,Xr1-xc)
8   //Correction of theta
9   Odp,Od1p,Odco,Od1co,p1,p2=CoTh(Odp,Od1p,Od,Od1)
10  //Calculating the differences
11  Xdn1=Xr-kx*(Xr-xc)    //Equations 5
12  Ydn1=Yr1-ky*(Yr-yc)
13  Odn1=Od1_corr-ktheta*(Od_corr-thc)
14  Dx=Xdn1-xc          //Equations 6
15  Dy=Ydn1-yc
16  Do=Odn1-thc
17  //Calculating the velocities (Equations 7)
18  V=Dx*math.cos(Odc)+Dy*math.sin(Odc)
19  w=Do
20  if (V>Vmax) then //Linear velocity saturation
21      V=Vmax
22  end
23  if (w>Wmax) then //Angular velocity saturation
24      w=Wmax
25  end
26  if (w<-Wmax) then
27      w=-Wmax
28  end
29  //Differential velocities
30  Vr=(2*V+w*L)/2
31  Vl=(2*V-w*L)/2
32  return Vr,Vl,Odp,Od1p,p1,p2,d,w
33 end
```

Figure 8 shows an example of this simulation where the robot is tracking a trajectory based on a *Lissajous* curve. The blue line is the trajectory and the red line is the trace performed by the robot. As can be seen, at the beginning the robot is in the center of the scene, and some seconds later, the robot reaches the trajectory to follow. Note that the time is not taken into account for this experiment.

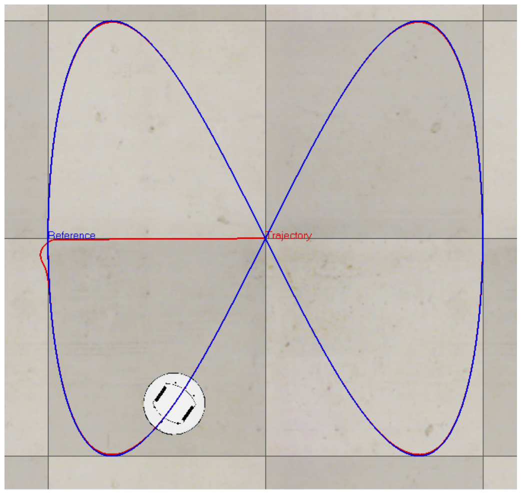

Fig. 8. Example of trajectory tracking

C. Simulation of path following

As was mentioned before, this experiment is similar to the trajectory tracking problem. But in this case the robot must follow the trajectory taking into account the time to arrive to each point. The implementation of this simulation is very similar to the trajectory tracking experiment. The function that calculates the velocities is modified (lines 18 and 19) to receive an extra parameter ($To = 0.5$) which is a constant that represents the time to arrive to each point. Another modification needed is related to the calculation of the velocities as was explained in Eq. 8.

```
//Calculating the velocities (Equations 7)
V=(Dx/To)*math.cos(Odc)+(Dy/To)*math.sin(Odc)
w=Do/To
```

Figure 9 shows the results of the path following simulation for the same *Lissajous* curve of the trajectory tracking case. Since now the robot takes into account the time to reach each point, there is a change at the beginning of the experiment. In this case, the points that were not reached at time by the robot are irrelevant from the control algorithm perspective, and therefore the trace of the robot here is different in comparison to the trajectory tracking performance.

Both simulations of trajectory tracking and path following, can be used in the classroom in a similar way as position control experiment. Instructors can also ask students to test several trajectories and to evaluate them using control performance indexes for both experiments.

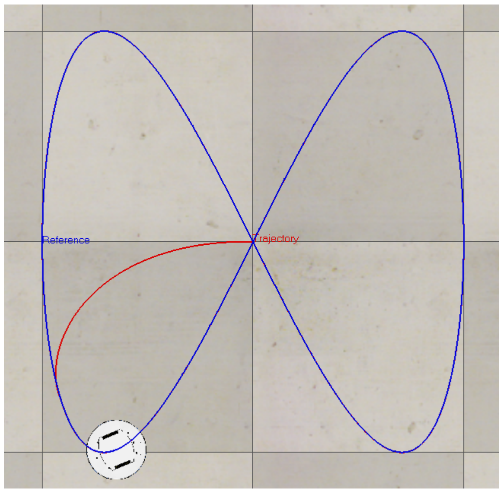

Fig. 9. Example of path following

D. Position control with obstacles Avoidance

This simulation implements the Braitenberg algorithm to avoid obstacles using the distance sensors of Khepera IV. The sensors are distributed around the robot each $45°$. This implementation executes the main *while* loop of the position control experiment. The function *Braitenberg* is added to avoid the obstacles. The following code fragment shows a part of this function. The *S* array is the measurement of the sensors and the *Wl* and *Wr* arrays are the weighted matrix determined empirically.

```
Braintenberg=function(Vmax,Wmax,Vl,Vr)
//If some obstacle is detected
if(S[1]+S[2]+S[3]+S[4]+S[5]+S[6]+S[7]+S[8]!=0)then
    for i=1,8,1 do
        // Velocities are calculated
        Vl = Vl + Vmax*Wl[i]*(1-(S[i]/Smax))
        Vr = Vr + Vmax*Wr[i]*(1-(S[i]/Smax))
    end
```

Figure 10 shows an experiment where the robot tries to reach the target point (red sphere) while avoiding obstacles in its path. The performance of the algorithm depends on the empirical weighted matrix. Note that Instructors can ask students to propose and evaluate several weighted matrices for different obstacle distributions.

IV. Conclusion

In this paper a virtual laboratory with some interesting experiments with mobile robots has been presented and implemented. For the implementations the library of the *Khepera IV* robot for *V-REP* has been used. The result of this development is a ready to use set of experiments that allows to test and understand theoretical concepts in a dynamic and attractive environment. In the near future, new experiments will be implemented. More robots can be added to simulate multi-robots experiments with the models included in *V-REP*.

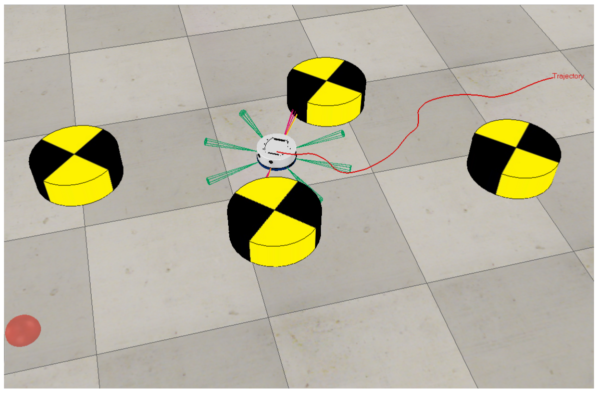

Fig. 10. Braitenberg strategy with the *Khepera IV* robot

Acknowledgment

The authors would like to thank to the Spanish Ministry of Economy and Competitiveness under the Project DPI2012-31303, and the Chilean Ministry of Education under the Project FONDECYT 1161584.

References

[1] E. Fabregas, G. Farias, S. Dormido-Canto, and S. Dormido, "RFC-SIM simulador de robótica móvil para control de formación con evitación de obstáculos," in *XVI Congreso Latinoamericano de Control Automático*, Cancún, México, 2014.

[2] C. Pinciroli, V. Trianni, and R. O'Grady, "ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics," in *International Conference on Intelligent Robots and Systems (IROS)*, Sept 2011, pp. 5027–5034.

[3] L. Guyot, N. Heiniger, O. Michel, and F. Rohrer, "Teaching robotics with an open curriculum based on the e-puck robot, simulations and competitions," in *Proceedings of the 2^{nd} International Conference on Robotics in Education. Vienna, Austria*, 2011.

[4] Coppelia Robotics GmbH, "V-REP simulator," 2016. [Online]. Available: http://www.coppeliarobotics.com

[5] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[6] E. Peralta, E. Fabregas, G. Farias, H. Vargas, and S. Dormido, "Development of a khepera iv library for the V-REP simulator," in 11^{th} *IFAC Symposium on Advances in Control Education*, Bratislava, Slovakia, June 2016.

[7] F. Kühne, W. F. Lages, and J. M. G. da Silva Jr, "Point stabilization of mobile robots with nonlinear model predictive control," in *Mechatronics and Automation, 2005 IEEE International Conference*, vol. 3, 2005.

[8] V. J. G. Villela, R. Parkin, M. L. Parra, J. M. D. González, M. J. G. Liho, and H. Way, "A wheeled mobile robot with obstacle avoidance capability," *Tecnología Y Desarrollo*, vol. 1, no. 5, pp. 159–166, 2004.

[9] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, May 1990, pp. 384–389 vol.1.

[10] P. Encarnacao and A. Pascoal, "3D path following for autonomous underwater vehicle," in *Proc. 39^{th} IEEE Conference on Decision and Control*, 2000.

[11] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," in *ICRA*, 2000, pp. 2505–2511.

[12] X. Yang, R. V. Patel, and M. Moallem, "A fuzzy braitenberg navigation strategy for differential drive mobile robots," *Journal of Intelligent and Robotic Systems*, vol. 47, no. 2, pp. 101–124, 2006.

[13] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.