Integration of PSO algorithm and differential robot kinematics using smooth and continuous controllers

Aldo Aguilar Nadalini
Mechatronics Engineering Department
Universidad del Valle de Guatemala
Guatemala City, Guatemala
agu15170@uvg.edu.gt

Dr. Luis Alberto Rivera
Mechatronics Engineering Department
Universidad del Valle de Guatemala
Guatemala City, Guatemala
larivera@uvg.edu.gt

MSc. Miguel Zea Arenales
Mechatronics Engineering Department
Universidad del Valle de Guatemala
Guatemala City, Guatemala
mezea@uvg.edu.gt

***Abstract*—A multi-agent differential robot system requires a definite algorithm to behave as a swarm with goal searching capabilities. The use of the Particle Swarm Optimization (PSO) algorithm is proposed to enable the agents to collectively find the optimal path to a goal in a fitness function. The classic PSO algorithm is designed for particles with no mass or physical dimensions unlike differential robots. Therefore, a Modified Particle Swarm Optimization (MPSO) algorithm is designed to take into account the restrictions derived from the kinematic equations of the robotic agents and the finite dimensions of the search space. The MPSO includes multiple known PSO parameters such as inertia weight, a constriction parameter, two scaling factors and two uniformity factors (for the cognitive and social components of the PSO) optimized for differential robots. It also includes the necessary controllers to map particle velocities into smooth and continuous differential robot velocities. Four different controllers were tested including: Transformed Unicycle (TUC), Transformed Unicycle with LQR (TUC-LQR), Transformed Unicycle with PID (TUC-PID), and a Lyapunov-stable Pose Controller (LSPC). The TUC-LQR was the controller that performed better in terms of achieving smooth trajectories and generating smooth continuous differential robot velocities.**

***Index Terms*—Differential robots; swarm intelligence; control; LQR; PID; PSO.**

I. Introduction

The Particle Swarm Optimization (PSO) algorithm is a popular tool in the computational intelligence field when it comes to finding the optimal solution of a determined fitness function. The main objective of this research is to find a way of coupling the searching capabilities of this algorithm to a swarm of differential robots such that it can be used in goal searching applications. The investigation focuses on the design of an algorithm that enables the differential robots to work as a whole and collectively find the most efficient and optimal path towards a specific target. To achieve this, the use of a Modified Particle Swarm Optimization (MPSO) algorithm is proposed to make the robots search for a path that minimizes the cost of their own position (X, Y) in a bi-dimensional plane evaluated in a fitness function. In this case, the goal or target to look for would be the absolute minimum of the fitness function.

In robotic swarm applications, agents with reduced physical dimensions are used. For example, in [1] the Kilobot agents are implemented. Other variations of differential robots are widely used as well. The EPFL E-Puck agents are differential robots with two wheel actuators and reduced dimensions, which makes them perfect for multi-agent swarm structuring. For that reason, this robot model is used for the research presented in this paper. More information regarding the specifications of the E-Puck robots can be found in [2]. The main challenge of coupling the PSO algorithm to differential robots, such as the E-Pucks, is the correct implementation of swarm algorithm considering the kinematic restrictions of these agents along with the velocity saturation limits of its wheels.

The classic PSO algorithm is designed for swarms of particles with no mass or physical dimensions unlike differential robots. To tackle this system inconsistency, the new MPSO algorithm is structured with PSO parameters optimized for the robotic agents and a set of controllers tasked with the calculation of the differential robot velocities using particle velocities as inputs. To find the optimal controller that ensures smooth and continuous control of the robots, this research compares the use of a velocity mapping technique using transformed unicycle kinematics [3], the use of pose controllers with Lyapunov stability and the use of a PID controller for the angular velocity of the E-Puck robots. The goal is to determine which controller performs better when it comes to driving the entire robotic swarm towards a specific goal while at the same time reducing the saturation and hard stops in the velocities of the robots' actuators. In section II, the background of this investigation is presented. In section III, the problem statement is presented. In section IV, the selection of optimal PSO parameter for the differential robots is presented. In section V, the structuring of the Webots simulation using MPSO is presented. In the last section (VI), the performance results of each controller during the Webots simulations are presented.

II. Background

A. Classic PSO

The classic PSO algorithm was first proposed in "Particle Swarm Optimization" by J. Kennedy and R. Eberhart (1995) [4]. It consists in a set of possible solutions, referred as particles, located throughout the problem space. Each particle has a fitness value based on its current position and a velocity vector

that indicates the direction to which the particle's position will change during the current iteration. The PSO algorithm initializes all particles' position randomly and then it is able to locate optima in a fitness function by updating generations of particle positions. Each particle moves throughout the space by following its new velocity vector calculated by adding the current velocity vector of the particle V_i, the cognitive factor which is the distance between the particle's current position p_i and the best position found locally by the particle p_l, and the social factor which is the distance between the particle's current position p_i and the best position found globally by the entire particle swarm p_g. Therefore, the two vectorial equations used by the PSO algorithm to search for optima are:

$$V_{i+1} = \varphi \cdot (\omega V_i + C_1\rho_1(p_l - p_i) + C_2\rho_2(p_g - p_i)) \quad (1)$$

$$p_{i+1} = p_i + V_{i+1} \cdot \Delta t \quad (2)$$

Equation (1) is the update rule for the particle's velocity. The variable φ is a constriction parameter multiplying all velocity factors to limit the maximum step that a particle can take when updating its position inside the problem space [5]. The variable ω is an inertia parameter first introduced in [6] that weighs the contribution of the current velocity in the calculation of the particle's new velocity. Its value can be computed in several ways including the calculations shown in [7]. C_1 and C_2 are scaling parameters for the cognitive and social factors in (1). The effects of different values for the scaling parameters in the particle swarm's behavior are explored in [8]. Variables ρ_1 and ρ_2 are uniformity factors which have a real random value in the interval (0,1]. These last factors add a key random behavior to the swarm so it can appropriately move in different directions while searching for optima. The update rule for the particle's position is shown in (2), where the new position vector p_{i+1} is the sum of the particle's current position p_i vector and the new velocity vector V_{i+1} multiplied by a time scaling factor Δt matching the sampling time used in computer simulations of the PSO.

The PSO is ideal for swarm intelligence optimization applications since it is cheap in terms of computational complexity. Particles of the swarm do not need to know any information from other particles other than their found local best positions p_l to compute the swarm's global best position p_g. Each particle's high individuality using the PSO algorithm makes it easy to implement in applications where there is no central processor calculating the results of each agent (e.g. mobile robots). Another advantage of the PSO when used in robotic applications is that the new position update of each agent depends on its current position. This means that the algorithm computes continuous paths for each agent as they search for optima.

B. Kinematics of differential robots

Differential robots are non-holonomic systems having three DOF and only two wheel actuators. Therefore, velocity mapping is required to transform the angular velocities of the robot's wheels (Φ_R, Φ_L) into velocities calculated in a bi-dimensional reference frame ($\dot{x}$, $\dot{y}$, $\dot{\theta}$); θ being the orientation angle in the XY coordinate system. The mapping of angular wheel velocities into linear wheel velocities is done by using the well-known unicycle robot model [3]. Algebraic manipulation of the unicycle model mapping equations and the robot dimensions result in the differential robot model equations:

$$\Phi_R = \frac{v + \omega l}{r} \quad (3)$$

$$\Phi_L = \frac{v - \omega l}{r} \quad (4)$$

where *r* is the wheel radius of the robot, *l* is the base radius of the robot, *v* and ω are the linear and angular velocity of the robot, respectively. These last velocities can be mapped to rectangular coordinates using simple trigonometric transformations [3].

C. Robot velocity controllers

There are many ways to achieve the movement of a robot from point to point. Several control laws have been developed in different investigations and have resulted in the correct translation of a robot. Control rules can be optimized for velocity smoothness, trajectory smoothness or both.

1) Point to point planar velocities: Planar velocity inputs can be calculated in different ways by using the error between the goal position and the actual position of the controlled object. In the case of a system based on a bi-dimensional plane XY, the planar velocities can be computed as follows:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} I_x \cdot tanh(k_x \cdot e_x) \\ I_y \cdot tanh(k_y \cdot e_y) \end{bmatrix} \quad (5)$$

where e_x and e_y are the position errors between the goal's coordinates and the robot's position [9]. $I_{x,y}$ are saturation constants for the hyperbolic tangent limiting the output and $k_{x,y}$ are proportional scaling factors. The velocities can also be computed using LQR control as follows:

$$\mathbf{u} = -\mathbf{K} \cdot (\mathbf{x} - \mathbf{x}_g) + \mathbf{u}_g \quad (6)$$

using the negative state feedback structured by multiplying the resulting LQR matrix **K** and the error between the system state **x** (robot coordinates) and the operational point ($\mathbf{x}_g$, $\mathbf{u}_g$).

2) Transformation of unicycle kinematics: This type of control transformation is based on the assumption that we can directly control the planar velocities u_1 and u_2 of a single point in a differential robot [9]. Therefore, the kinematic equations are written as follows:

$$v = u_1 \cdot cos(\theta) + u_2 \cdot sin(\theta) \quad (7)$$

$$\omega = \frac{-u_1}{l} \cdot sin(\theta) + \frac{u_2}{l} \cdot cos(\theta) \quad (8)$$

where all variables are the same as used in Subsection 2c.

3) Angular PID Controller: For increased stability in the robot's velocities, a PID controller for angular velocity can be coupled with linear velocity controllers such as (7). The PID would return the robot's angular velocity ω after summing the proportional, integral and derivative angular error *e(t)* between the robot and the goal.

4) Pose Controllers: A useful control technique for goal reaching with smooth trajectories is the pose control. This type of controller tries to minimize the error in three different parameters measured in polar coordinates. Firstly, parameter ρ is the magnitude of the vector between the center of the robot and the goal point, α is the angle between the ρ vector and the horizontal axis of the robot's reference frame, and β is the desired angle of orientation when the robot reaches the goal [10]. The control laws are structured as follows:

$$v = k_\rho \cdot \rho \tag{9}$$

$$w = k_\alpha \cdot \alpha + k_\beta \cdot \beta \tag{10}$$

Some variants of this last pose controller have been developed. In [11], pose control laws were developed using the direct Lyapunov stability criterion and the Barbalat's Lemma. This controller only minimizes the ρ and α parameters. The control laws are structured as follows:

$$v = K_\rho \rho \cdot cos(\alpha); \quad K_\rho > 0 \tag{11}$$

$$\omega = K_\rho sin(\alpha) cos(\alpha) + K_\alpha \alpha; \quad K_\alpha > 0 \tag{12}$$

III. Problem Statement

The classic PSO algorithm is structured to work with particle velocities. Particles do not have kinematic restrictions since they have neither physical dimensions nor mass, unlike a differential robot such as the E-Puck model. The movements that they can execute are limited, unlike particles which can move in any direction at any velocity. These robotic agents also have wheel actuators that have saturation velocities and physical restrictions such as the speed in which they can change their velocities. A swarm of differential robots is also limited to the number of agents implemented. In this investigation, a swarm of 10 E-Puck robots is used in the simulations. The amount of agents to use was determined according to the dimension constrains of the simulated search space (2x2m) to avoid overcrowding. The first objective of this research is to determine the PSO parameters that cause the desired swarm behavior. To account for the reduced number of agents used in the robotic swarm, ample range of exploration of the problem space is required to avoid incorrect convergence to local minima. Goal exploitation is also desired since the robotic swarm needs to converge to the goal in a finite amount of time.

The second objective is to determine which velocity controller is the best to transform the discontinuous PSO vectorial velocities into smooth and continuous differential robot velocities. The selected controller needs to be able to guide all robots to the goal in a finite amount of time, needs to regulate velocities in such ways that it can reduce saturation and hard stops in the robots' actuators, and it needs to generate robot trajectories that are dynamic enough for goal searching applications. The E-Puck swarm simulations will be done with the Webots software [12] and the results will be shown in Section V.

IV. PSO Parameter Selection with MATLAB Simulations

To determine which parameters were adequate to use in the PSO equations (1) and (2), MATLAB simulations were performed using swarms of 200 particles in a 10x10m space and a sampling period (Δt) of 0.1s. Several known benchmark functions were used as fitness functions to test the algorithm's performance including the Rosenbrock, Himmelblau, Booth and Sphere functions [13]. Goal convergence time and initial swarm dispersion (measured at the first second of the simulation to avoid measuring initial positions dispersion) were measured to analyze the PSO performance.

During simulations, the analyzed parameter was the only modified value between interations of the program and the remaining parameters were fixed at default values. The default values for the scaling parameters C_1 and C_2 were set at 1 and 5, respectively, instead of the popular combination of 2 and 2 proposed in [4]. This 1:5 ratio between cognitive and social factor was arbitrarily set to prioritize goal convergence speed over space exploration to avoid swarm fragmentation. The default value for the constriction factor φ was fixed at 1.0 to eliminate its effect over the PSO velocity calculations.

The Inertia Parameter ω was the first factor analyzed to determine its ideal default value for the next simulations to tune the other parameters. Five different inertia calculations were tested: chaotic (CH), random (RD), constant (CT), linear decreasing (LD), and natural exponential inertia (NE) [7]. The NE inertia showed the most initial swarm dispersion indicating a broader exploration range on the problem space and the swarm was able to converge at the goal in a finite amount of time (5.50s). Therefore, the NE Inertia was selected as the appropriate default value.

To tune the scaling parameters, three arbitrary combinations were tested to analyze the swarm's behavior in the most important cases: cognitive and social factors weigh the same, cognitive factor weighs more than the social factor, and social factor weighs more than the cognitive factor. For the first case, the popular combination C_1 = 2, C_2 = 2 proposed in [4] was used. For the second case, an arbitrary ratio of 5:1 in favor of cognitive factor was used. Lastly, for the third case, the default ratio of 1:5 in favor of social factor was used. The last case resulted in the most ample exploration range of the search space without swarm fragmentation and the swarm converged to the goal in a finite time. It was observed that the scaling parameters could be doubled in value while keeping the same ratio 1:5 to further extend the exploration range.

To tune the the constriction factor φ, three main cases were evaluated: constriction factor shortens the particle's steps (factor is less than one), constriction factor elongates the particle's steps (factor is greater than one), constriction factor has no effect (factor is equal to one). For the first case, φ was set to 0.5. For the second case, φ was set to 1.3. And for the third case φ = 1.0. Simulations showed that values of $\varphi > 1.2$ caused swarm divergence and values of $\varphi < 0.6$ reduced the space exploration to nearly zero. For that reason, φ was set to 1.0 to eliminate this parameter's effect over the PSO velocity.

Therefore, the selected PSO parameters are: ω = NE inertia parameter [7], $C_1 = 2$, $C_2 = 10$, and $\varphi = 1.0$. These selected parameters ensure an adequate exploration range (around 70%) of the problem space and adequate convergence time, as shown in Fig.1.

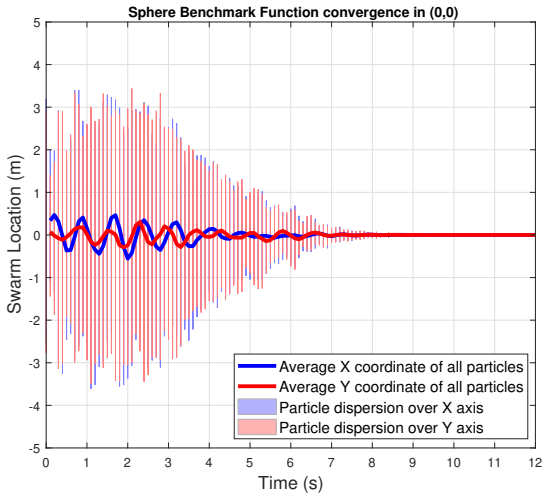

Fig. 1. PSO simulation in MATLAB with selected parameters

V. MPSO Algorithm Structuring in Webots

After selecting the adequate PSO parameters to achieve the desired swarm behavior, the Webots simulated world was structured to proceed with the analysis of the PSO implementation on differential robots using 4 different kinematic controllers for velocity mapping. Webots simulations were performed using swarms of 10 E-Puck robots in a 2x2m space and a sampling period (T_s) of 32ms. The Sphere benchmark function with absolute minima at (0,0) was used as fitness function $f(\cdot)$ to evaluate the algorithm's performance. The *GPS* and *Compass* nodes in Webots were used to gather the position and orientation information of each E-Puck agent. More information on how to read data from these nodes can be found in [14]. The *Emitter* and *Receiver* nodes were used for transmission and reception of local best positions p_l of every E-Puck to poll for the global best position p_g inside the swarm. [a]

[a]A detailed description of the MPSO algorithm code containing the integration of the selected PSO parameters with the kinematics controllers can be found in [14], along with extended research results.

During the algorithm's execution, the new agent velocity was calculated using (1). The PSO parameters used in this equation were the ones chosen in the previous section. Next, the mapping of PSO velocities to differential robot velocities was done through kinematic controllers. In this research, four kinematic controllers were structured and tested to determine which gave the robotic swarm the best performance in terms of convergence velocity, trajectory smoothness and smooth velocities (less saturation and hard stops of the robot actuators). The four tested kinematic controllers were the following:

A. Transformed Unicycle controller (TUC)

It was structured by coupling the planar velocities calculations shown in (5) with (7) and (8) to get the robot's linear and angular velocities. In this case, parameters used in (5) were set to: $\Phi_{max} = 6.28$, $I_{x,y} = 0.5 \cdot \Phi_{max}$. The proportional scaling factors k_x and k_y used in (5) were calculated as follows:

$$k = \frac{1 - exp(-2 \cdot \sqrt{e_x^2 + e_y^2})}{2 \cdot \sqrt{e_x^2 + e_y^2}} \tag{13}$$

B. Transformed Unicycle with LQR controller (TUC-LQR)

It was structured by coupling the planar velocities calculations shown in (6) with (7) and (8). In this case, the 2x2 LQR matrix **K** was computed with the MATLAB command *lqr(A,B,Q,R)* where matrix A is zero, B and R were identity, and Q was identity scaled by 0.01. An additional scaling factor ζ directly multiplying the reference $\mathbf{x}_g$ was set at 0.01 to directly tune tracking smoothness. Vectors $\mathbf{x}_g$ and $\mathbf{x}$ in (6) are equivalent to p_{i+1} and p_i from (2), respectively.

C. Transformed Unicycle PID controller (TUC-PID)

It was structured by coupling the planar velocities calculations shown in (5) with (7) to get the robot's linear velocity and using a digital PID controller to get the robot's angular velocity according to the orientation error e_o. In this case parameters were set to: $K_p = 0.5$, $K_i = 0.1$, $K_d = 0.001$, $\Phi_{max} = 6.28$, $I_{x,y} = 0.5 \cdot \Phi_{max}$. The proportional scaling factors k_x and k_y used in (5) were calculated as in (13). The orientation error e_o for the PID was calculated as follows:

$$\theta_g = atan2(y_g - y,\ x_g - x) \tag{14}$$

$$e_o = atan2(sin(\theta_g - \theta),\ cos(\theta_g - \theta)) \tag{15}$$

D. Lyapunov-stable Pose Controller (LSPC)

This controller used the control laws presented in (11) and (12) to calculate the linear and angular velocity of the E-Puck robots. In this case, position and orientation errors between the goal and the robot's location were mapped to polar coordinates as follows:

$$\rho = \sqrt{e_x^2 + e_y^2} \tag{16}$$

$$\alpha = -\theta + atan2(e_y, e_x) \tag{17}$$

where α belongs to the interval [-π,π]. The controller's parameters in (11) and (12) were set to: $k_\rho = 0.01$, $k_\alpha = 0.5$. If α pointed at the left quadrants of the XY plane, the sign of the linear velocity was inverted.

VI. Webots Simulation Results

Four simulations using the MPSO algorithm were performed in Webots. Each simulation used a different kinematic controller to compare the performance results of each one. The E-Pucks were located at different distances and orientations from the goal to vary initial conditions for each robot. PSO parameters were not modified between simulations. Compared to the chosen PSO parameters in section IV, the time scaling factor Δt was changed to 1s rather than 0.032s and φ was changed to 0.8 because it stabilized particle velocities enough to be implemented to the robots. Trajectories of the 10 E-Puck robots reaching the goal were plotted to evaluate smoothness of the paths. Figures 2-5 show the trajectory results for all four kinematic controllers. The 2x2m grid represents the problem space, the blue plots are individual E-Puck trajectories, the red dots are initial positions of each E-Puck robot, and the black dot at (0,0) is the absolute minima of the Sphere fitness function.

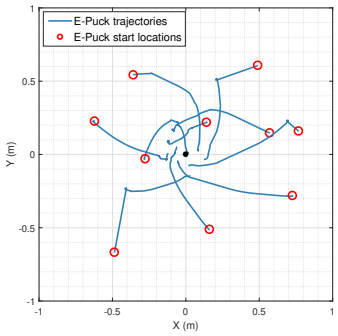

Fig. 2. Simulation with TUC

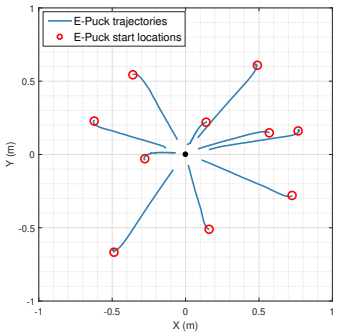

Fig. 3. Simulation with TUC-LQR

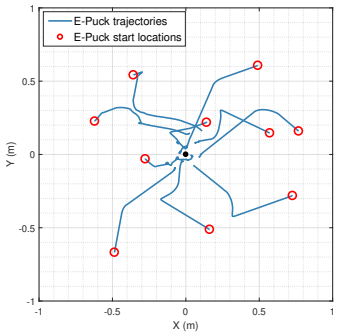

Fig. 4. Simulation with TUC-PID

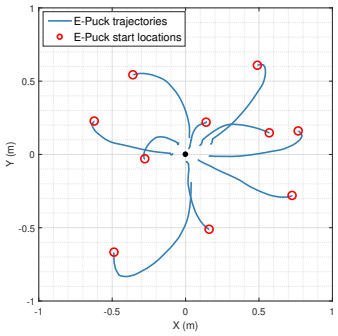

Fig. 5. Simulation with LSPC

The controller that generated straighter paths was the TUC-LQR controller, whereas the LSPC generated smooth curved trajectories. The TUC and the TUC-PID controllers resulted in more irregular trajectories but the convergence time was the same as the LSPC (10s). Convergence time for TUC-LQR was 15s.

Measuring the smoothness of the control signals applied to the robot's actuators was also key to evaluate each controller's performance. Firstly, cubic spline interpolations of the resultant actuator velocity curves (Φ_R and Φ_L) were performed for each controller. Cubic splines have a minimum energy property based on the Euler-Bernoulli beam theory, as shown in [15]. Consequently, the smoothness of a resultant velocity curve could be measured by calculating the minimum energy W required to bend an elastic thin beam to shape it as its interpolated cubic spline $y(x)$. Greater values of W indicate less smooth velocity curves corresponding to aggressive controllers, whereas reduced W values indicate control smoothness. A controller with reduced W is desired to avoid violent velocity changes in the robot's actuators. Wheel velocity data from both actuators of each E-Puck robot in the swarm was collected during the simulations of all four controller. The bending energy W of a total of 20 control curves (two actuator velocities for each of the 10 E-Pucks) was calculated for each controller using (18).

$$W = \frac{1}{2}\int_{x_0}^{x_n} y''(x)^2 dx \tag{18}$$

The next figure shows the smoothness results for all velocity curves generated with each controller:

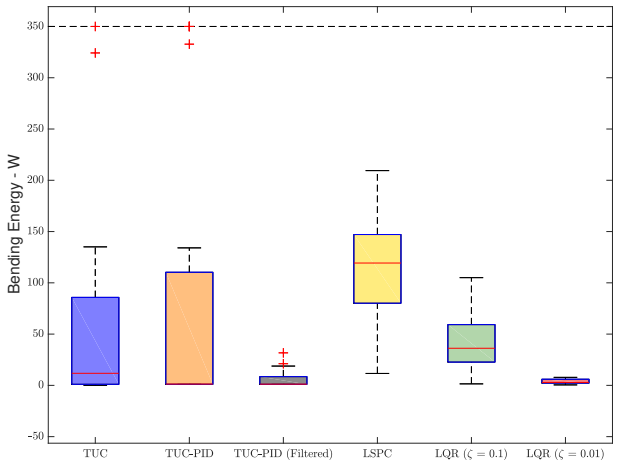

Fig. 6. Control smoothness calculated with bending energy

As shown in Figure 6, the controller that statistically presented the most smooth velocity curves was the TUC-LQR with a reference scaling factor of ζ set to 0.01 in (6). Setting this scaling factor to 0.1 reduced the controller smoothness because of the presence of sharper velocity peaks caused by the constant acceleration when the PSO position for the robot was updated. Because of this small scaling value, the controller was slower than the others, so it was a trade-off between velocity smoothness and controller speed. This controller did not cause any saturation of the robots' actuators, as shown in Figure 7.

The second best velocity controller was the TUC-PID with the implementation of a velocity sharp peak filter. This filter could be implemented to reduce rapid changes in velocity without affecting the controller's performance in terms of convergence and convergence speed. If the change in actuator velocities $\Phi_{R,L}$ between iterations of the simulation was greater than 1 rad/s, the velocities were recalculated as:

$$\Phi_i = \frac{\Phi_i + 2 \cdot \Phi_{i-1}}{3} \tag{19}$$

The difference between the filtered and unfiltered TUC-PID results is evident in Figure 6. Smoothness is greatly incremented with the filter, and the swarm still converges to the goal. Trajectories shown in Figure 4 were the result of the filtered TUC-PID.

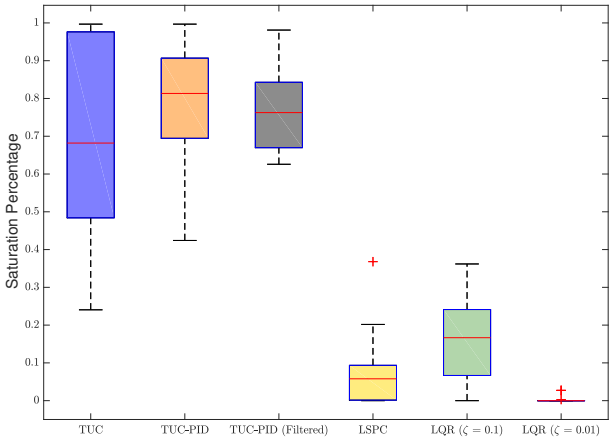

Fig. 7. Percentage of control saturation during simulation

As shown in Figure 7, both filtered and unfiltered TUC-PID controllers presented elevated saturation percentage of the robots' actuators, unlike the TUC-LQR. The robots' actuators were saturated around 80% of the time. These percentages were calculated by determining what portion of the data from every velocity curve was equal to the saturation velocities of the E-Puck actuators ($\pm$ 6.28 rad/s). Figures 8 and 9 show the velocity curves for the TUC-LQR (ζ = 0.01) and TUC-PID (Filtered) implemented on one E-Puck. Some peaks can be seen at the end of the simulation for the TUC-PID in Figure 9, but these were mostly caused by E-Puck collisions when all robots reached the goal.

VII. Conclusions

The PSO parameters that led the robotic swarm to an adequate exploration range of the problem space (70% of the 2x2m space) and an adequate goal exploitation rate (10-15s to converge) were: Natural exponential inertia ω, scaling parameters $C_1 = 2$ and $C_2 = 10$, constriction parameter $\varphi = 0.8$, and $\Delta t = 1$.

The LSPC controller resulted in curved smooth paths unlike the straight trajectories generated by the TUC-LQR. However, even though the paths that it generated showed a smooth behavior, it resulted in an elevated amount of velocity change spikes that could cause damage in the robot's actuators, just like the TUC controller. Therefore, the use of these controllers with no additional modifications, is not recommended for future robotic swarm applications.

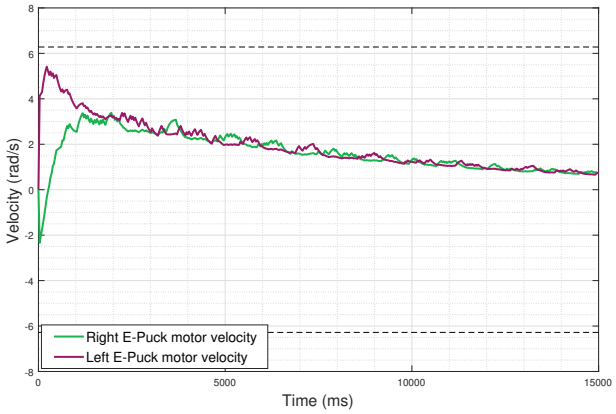

Fig. 8. E-Puck actuator velocities with TUC-LQR

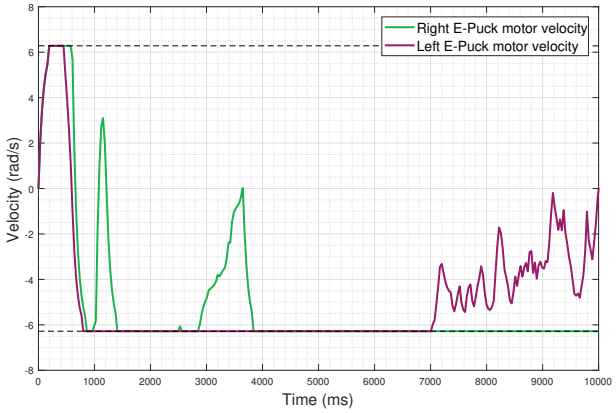

Fig. 9. E-Puck actuator velocities with TUC-PID

The controller that led to straighter and less curved paths towards the goal in the problem space was the TUC-LQR controller. It also resulted in the most smooth actuator velocities, with no saturation and no velocity change spikes. However, this was the slowest and the most rigid controller. For search applications using robotic swarms in environments with low amount of obstacles, this is the recommended controller.

The TUC-PID controller resulted in more irregular paths than the LSPC and TUC-LQR controllers, but the convergence time was the same as the TUC and LSPC (10s). This controller also resulted in the second smoothest velocities. However, it presented high control saturation rate which could damage robot actuators that do not endure extended periods of time working at maximum speed. This controller is also recommended for future robotic swarm applications taking into account proper actuator velocity limitation.

REFERENCES

[1] E. R. Magsino, F. A. V. Beltran, H. A. P. Cruzat, and G. N. M. De Sagun, "Simulation of search-and-rescue and target surrounding algorithm techniques using kilobots," in *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 70–74, IEEE, 2016.

[2] M. Bonani, "Epfl e-puck robot." http://www.e-puck.org/. Accessed: 2019-04-29.

[3] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," pp. 1–7, IEEE, 1995.

[5] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, vol. 1, pp. 84–88, IEEE, 2000.

[6] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pp. 69–73, IEEE, 1998.

[7] J. C. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," in *2011 Third world congress on nature and biologically inspired computing*, pp. 633–640, IEEE, 2011.

[8] T. Beielstein, K. E. Parsopoulos, and M. N. Vrahatis, *Tuning PSO parameters through sensitivity analysis*. Universitätsbibliothek Dortmund, 2002.

[9] F. Martins and A. Brandão, "Velocity-based dynamic model and control." https://bit.ly/32s6jRX, 2018. Accessed: 2019-05-30.

[10] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[11] S. K. Malu and J. Majumdar, "Kinematics, localization and control of differential drive mobile robot," *Global Journal of Research In Engineering*, 2014.

[12] Webots, "http://www.cyberbotics.com." Commercial Mobile Robot Simulation Software.

[13] D. Bingham, "Project homepage deap 1.3.0 documentation." https://deap.readthedocs.io/en/master/api/benchmarks.html#deap.benchmarks.sphere. Accessed: 2019-04-29.

[14] A. Aguilar, "Algoritmo modificado de optimización de enjambre de partículas." https://github.com/AldoAguilar001/MPSO-Algorithm, 2019.

[15] G. Wolberg and I. Alfy, "An energy-minimization framework for monotonic cubic spline interpolation," *Journal of Computational and Applied Mathematics*, vol. 143, no. 2, pp. 145–188, 2002.