Control of E-Puck swarm with goal searching capabilities using Particle Swarm Optimization

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 E-Puck robotic 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 cost function. The classic PSO algorithm is designed for particles with no mass or physical dimensions unlike E-Puck 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 E-Puck 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 the E-Puck robots. It also includes the necessary controllers to map particle velocities into 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-PID, which included a PID controller for the angular velocity of the E-Puck agents, resulted in adequate and dynamic trajectories and well regulated velocities.**

***Index Terms*—E-Puck; swarm intelligence; control; PID; PSO.**

I. Introduction

The swarm intelligence field of investigation has gathered a significant amount of attention in recent years due to its multiple applications, mainly in the execution of tasks that are too complex to be executed by a single robot. For example, robotic swarm applications have been developed for navigation planning, target tracking, search and rescue tasks, etc. The main goal of robotic swarm is to reduce the energy consumption of a system by decomposing a complex task into a set of simpler tasks executed by multiple individual agents. Robotic swarm also focuses on the design of dynamic systems that can adapt quickly to highly changing environments. One of the main challenges when it comes to bio-inspired systems is the fact that there are certain random components needed for the adequate movement of the swarm. These can cause difficulties during the implementation of swarm intelligence algorithms to robotic systems which have limitations in the physical world.

For 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 proposed for the research presented in this paper. The main challenge with using E-Puck robots is the correct implementation of swarm algorithms considering the kinematic restrictions of these agents along with the velocity saturation limits of its wheels. The E-Puck differential robots are non-holonomic systems. Therefore, velocity mapping for the actuator velocities is needed to achieve robot movement in all degrees of freedom (DOF). The main objective of this investigation is to achieve the structuring of an E-Puck robotic swarm comprised of 10 agents that can execute goal searching tasks inside a finite workspace. The goal is to design an algorithm that enables the E-Puck 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 bidimensional plane evaluated in a fitness or cost function. In this case, the goal or target to look for would be the absolute minimum of the fitness function.

The classic PSO algorithm is designed for swarms of particles with no mass or physical dimensions unlike the E-Puck robots. To tackle this system inconsistency, the new MPSO algorithm with PSO parameters optimized for the E-Puck robots is structured along with a set of controllers tasked with the calculation of the differential robot velocities using particle velocities as inputs. This research compares the use of a velocity mapping technique using transformed unicycle kinematics [2], the use of pose controllers with Lyapunov stability and the use of a PID controller for the angular velocity of the E-Puck robots. This is done 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 E-Puck actuators. In section II, the background of this investigation is presented including previous works on the PSO algorithm. In section III, the problem statement is

presented. In section IV, the results of optimal PSO parameter selection via MATLAB simulations are presented. In section V, the structuring of the Webots simulation using MPSO is presented to show the reader how the E-Puck swarm behavior was simulated. 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) [3]. 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 or cost 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:

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

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

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 [4]. The variable ω is an inertia parameter first introduced in [5] 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 [6]. 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 [7]. 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 X_{i+1} is the sum of the particle's current position X_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. EPFL E-Puck differential robot

This differential robot model was designed by Michael Bonani and Francesco Mondada in the ASL laboratories of the École Polytechnique Fédérale de Lausanne University. Its design is open hardware and the on-board control software is also open source. The physical characteristics of the two-wheeled differential robot are:

- Base diameter: 70 mm
- Height: 50 mm
- Wheel radius: 20.5 mm
- Total weight: 200 g
- Maximum linear velocity: 13 cm/s (angular 6.28 rad/s)

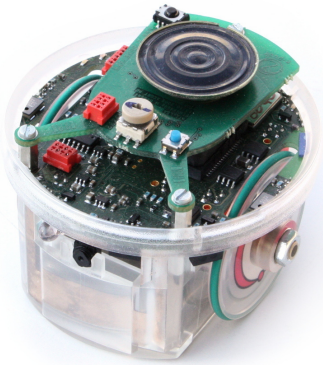

Fig. 1. E-Puck Differential Robot (EPFL, 2019)

C. 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 [2]. 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 [2].

D. 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:

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

where e_x and e_y are the position errors between the goal's coordinates and the robot's position [8]. $I_{x,y}$ 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:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \bar{N} \cdot \begin{bmatrix} x_g \\ y_g \end{bmatrix} - \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{6}$$

where $\bar{N}$ is a scaling factor for the reference $\mathbf{x}_g$ (goal coordinates) and the negative state feedback is structured by multiplying the resulting LQR matrix $\mathbf{K}$, calculated with the MATLAB command *lqr(A,B,Q,R)*, and the system state $\mathbf{x}$ (robot coordinates).

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 [8]. Therefore, the kinematic equations are written as follows:

$$v = u_1 \cdot cos(\theta) + u_2 \cdot sin(\theta) \tag{7}$$

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

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 control rule for the robot's angular velocity ω using the angular error *e(t)* between the robot and the goal would be structured as follows:

$$\omega = K_p e(t) + K_i \int_0^t e(t)dt + K_v \frac{de(t)}{dt} \tag{9}$$

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 [9]. The control laws are structured as follows:

$$v = k_\rho \cdot \rho \tag{10}$$

$$w = k_\alpha \cdot \alpha + k_\beta \cdot \beta \tag{11}$$

Some variants of this last pose controller have been developed. In [10], 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:

$$v = K_\rho \rho \cdot cos(\alpha); \quad K_\rho > 0 \tag{12}$$

$$\omega = K_\rho sin(\alpha) cos(\alpha) + K_\alpha \alpha; \quad K_\alpha > 0 \tag{13}$$

III. Problem Statement

The classic PSO algorithm is structured to work with particle velocities. Particles do not have kinematic restrictions since they have no physical dimensions nor mass, whereas E-Puck robots follow the kinematic restrictions of a differential robot model. The movements that they can execute are limited, unlike particles which can move in any direction at any velocity. E-Puck robots 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 E-Puck 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 first objective of this research is to determine the PSO parameters that cause the desired swarm behavior via MATLAB. 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 adapt the PSO velocities to the E-Puck 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 to simulate the physical world in a realistic way.

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 [11] were used as fitness or cost functions to test the algorithm's performance including the Rosenbrock, Himmelblau, Booth and Sphere functions. 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.

Firstly, the Inertia Parameter ω was tuned in the simulations. The scaling parameters C_1 and C_2 were fixed at values of 1 and 5, respectively. The constriction factor φ was fixed at 1.0, and the Rosenbrock benchmark function with absolute minima at (1,1) was used. PSO performance was analyzed with five different inertia calculations: chaotic (CH), random (RD), constant (CT), linear decreasing (LD), and natural exponential inertia (NE) [6]. As shown in Table I, NE inertia had the most initial swarm dispersion indicating a broader exploration range on the problem space, a quality desired in this application. The convergence time (5.50s) was the highest in this case, although it is still an adequate goal exploitation rate for this application.

TABLE I
PSO PERFORMANCE DEPENDING ON INERTIA CALCULATIONS

Inertia Type	**PSO performance measurements**		
	Convergence Time (s)	***Std. Dev. X***	***Std. Dev. Y***
CH	0.80	0.15	0.15
RD	2.20	0.75	0.75
CT	2.80	0.80	1.40
LD	5.00	1.80	2.50
NE	5.50	1.85	2.60

Secondly, the scaling parametes C_1 and C_2 were tuned in the simulations. The inertia parameter was calculated as NE. The constriction factor φ was fixed at 1.0, and the Himmelblau benchmark function with 4 absolute minima at (3,2), (-2.8,3.13), (-3.8,-3.3), (3.6,-1.8) was used to detect cases of swarm fragmentation. PSO performance was analyzed using four different combinations of scaling parameters: (C_1, C_2) = (2,2), (1,5), (5,1), (2,10).

TABLE II
PSO PERFORMANCE DEPENDING ON SCALING PARAMETERS

Scaling Combination	**PSO performance measurements**		
	Convergence Time (s)	***Std. Dev. X***	***Std. Dev. Y***
(2,2)	7.00	1.80	1.80
(1,5)	6.00	2.80	2.80
(5,1)	–	1.70	2.50
(2,10)	7.00	3.50	3.00

As shown in Table II, the (2,10) combination had the most initial swarm dispersion indicating a broader exploration range on the problem space and adequate convergence time (7.00s). It can be observed that the (5,1) combination resulted in PSO divergence because a higher weighing for the cognitive factor given by C_1 caused local minima convergence and swarm fragmentation.

Finally, the constriction parameter φ was tuned in the simulations. The inertia parameter was calculated as NE. The scaling parameters C_1 and C_2 were fixed at values of 1 and 5, respectively. The Booth benchmark function with absolute minima at (1,3) was used. PSO performance was analyzed using four different constriction values: 0.5, 1.0, 1.1, 1.3. As shown in Table III, a constriction value of 1.0 generated an adequate initial swarm dispersion, unlike constriction values of 0.5 and 1.3 which generated almost no dispersion or complete divergence, respectively. It also had an adequate convergence time (5.50s).

TABLE III
PSO PERFORMANCE DEPENDING ON CONSTRICTION PARAMETER

Constriction Value	**PSO performance measurements**		
	Convergence Time (s)	***Std. Dev. X***	***Std. Dev. Y***
0.5	1.10	0.20	0.20
1.0	5.50	2.00	2.70
1.1	8.20	6.20	8.40
1.3	–	480	740

Therefore, the selected PSO parameters are: $\omega \rightarrow$ NE inertia parameter [6], $C_1 = 2$, $C_2 = 10$, 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.2.

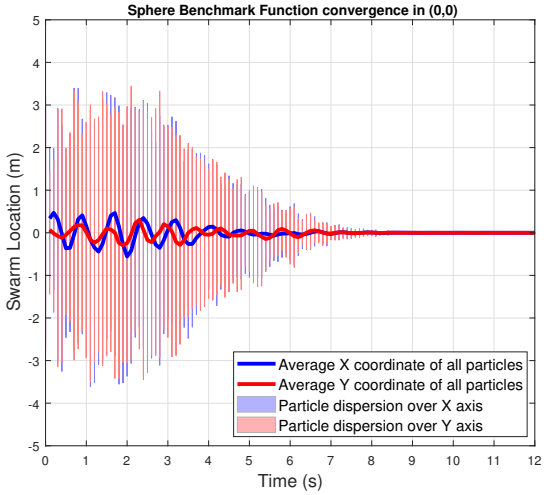

Fig. 2. PSO simulation in MATLAB with selected parameters

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 the E-Puck robots. The performance of 4 different kinematic controllers for velocity mapping was also analyzed. 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 [12]). 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. The MPSO algorithm coupling the selected PSO parameters with the kinematics controllers was programmed as following:

Algorithm 1 Modified Particle Swarm Optimization (MPSO)

1: **procedure** INDIVIDUAL E-PUCK PROGRAMMING
2: $n \leftarrow 0$
3: $l \leftarrow$ *E-Puck base radius*
4: $r \leftarrow$ *Wheel radius*
5: *Loop*:
6: $p_i \leftarrow$ *GPS node value*
7: **if** $n = 0$ **then**
8: $p_l \leftarrow p_i$
9: $p_g \leftarrow p_i$
10: $\phi \leftarrow$ *Compass node value*
11: $\theta \leftarrow 270° - \phi$
12: *Local Best calculation*:
13: **if** $f(p_i) < f(p_l)$ **then**
14: $p_l \leftarrow p_i$
15: *Global Best calculation*:
16: $Emitter\ send \rightarrow p_l$
17: **while** *Receiver node buffer* > 0 **do**
18: $p_x \leftarrow$ *Receiver node value*
19: **if** $f(p_x) < f(p_g)$ **then**
20: $p_g \leftarrow p_x$
21: **if** $f(p_l) < f(p_g)$ **then**
22: $p_g \leftarrow p_l$
23: *PSO calculation*:
24: $V_{i+1} \leftarrow$ *PSO Velocity Eq.(1)*
25: $p_{i+1} \leftarrow$ *PSO Position Eq.(2)*
26: *Kinematic velocities calculation*:
27: *v, w* $\leftarrow$ *velocity mapping controllers Eqs.(5-13)*
28: $\Phi_{L,R} \leftarrow$ *differential model Eqs.(3-4)*
29: *Motors set velocities* $\rightarrow \Phi_{R,L}$
30: $n \leftarrow n + 1$
31: *delay* T_s *ms*
32: **goto** *Loop*

As seen in Line 24 of Algorithm 1 (MPSO), the new agent velocity is calculated using (1). The PSO parameters used in this equation are the ones chosen in the previous section. In Line 27, the mapping of PSO velocities to differential robot velocities is 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 kinematic controllers are 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:

$$k = \frac{1 - exp(-2 \cdot \sqrt{e_x^2 + e_y^2})}{2 \cdot \sqrt{e_x^2 + e_y^2}} \tag{14}$$

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 was null, B and R were identity, and Q was identity scaled by 0.01. Factor $\bar{N}$ was set at 0.01. Note: Vectors $\mathbf{x}_g$ and $\mathbf{x}$ in (6) are equivalent to p_{i+1} and p_i from Algorithm 1 (MPSO), 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 (9) 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 (14). The orientation error e_o for the PID was calculated with the next equations:

$$\theta_g = atan2(y_g - y,\ x_g - x) \tag{15}$$

$$e_o = atan2(sin(\theta_g - \theta),\ cos(\theta_g - \theta)) \tag{16}$$

D. Lyapunov-stable Pose Controller (LSPC)

This controller used the control laws presented in (12) and (13) 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 using the next equations:

$$\rho = \sqrt{e_x^2 + e_y^2} \tag{17}$$

$$\alpha = -\theta + atan2(e_y, e_x) \tag{18}$$

where α belongs to the interval [-π,π]. The controller's parameters in (12) and (13) were set at: $k_\rho = 0.01$, $k_\alpha = 0.5$. Note: 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 controllers to compare the performance results of each one. 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 and regularity of the paths. In Figures 3-6, the trajectory results for all four kinematic controllers can be seen. 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.

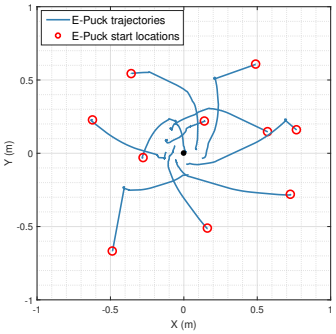

Fig. 3. Simulation with TUC

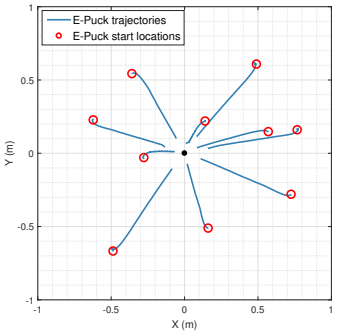

Fig. 4. Simulation with TUC-LQR

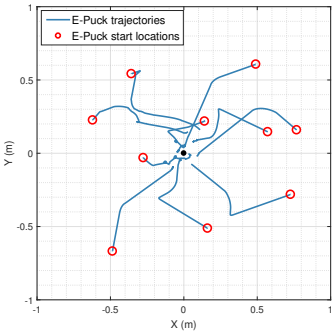

Fig. 5. Simulation with TUC-PID

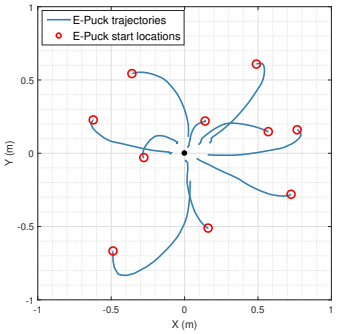

Fig. 6. Simulation with LSPC

The controller that generated more straight 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.

To evaluate the controller's performance in terms of velocity regulation, a single E-Puck was randomly selected from the swarm and both actuator velocities were plotted. The actuator velocities were bounded at -6.28 and 6.28 rad/s (the saturation velocity of the E-Puck robots). In Figures 7-10, the velocity results for all four kinematic controllers can be seen.

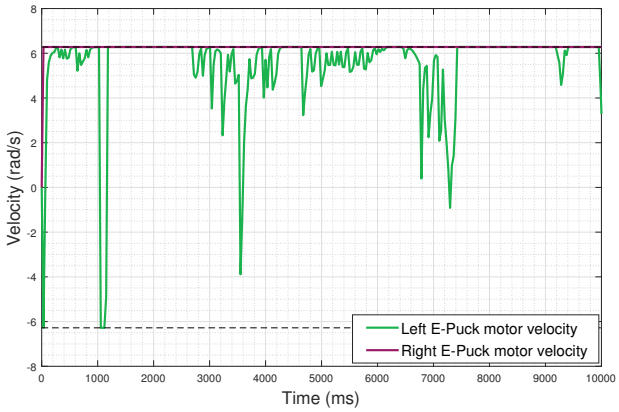

Fig. 7. E-Puck actuator velocities with TUC

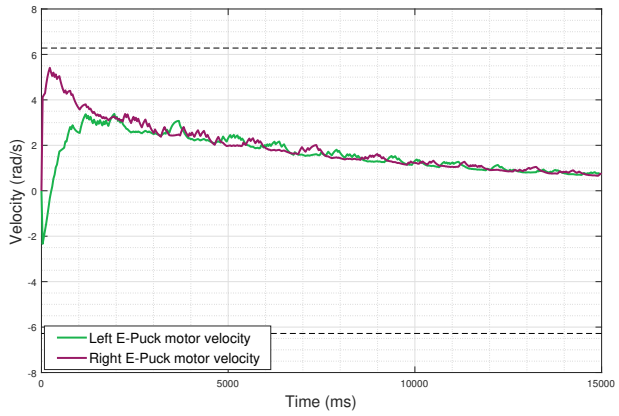

Fig. 8. E-Puck actuator velocities with TUC-LQR

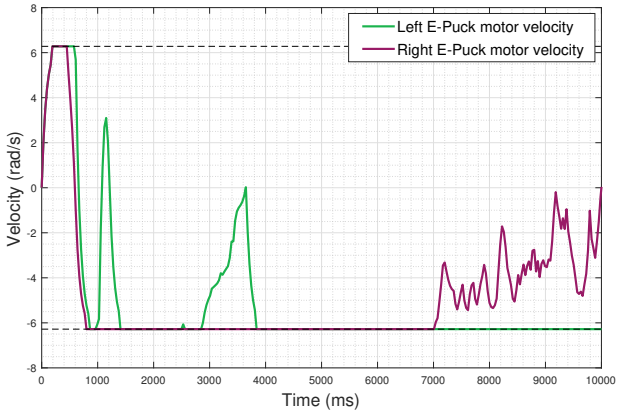

Fig. 9. E-Puck actuator velocities with TUC-PID

The controller that resulted in the most regulated and uniform velocities was the TUC-LQR. As shown in Fig.8, velocities do not present saturation in the robot's actuators and do not present sharp change peaks that could damage the

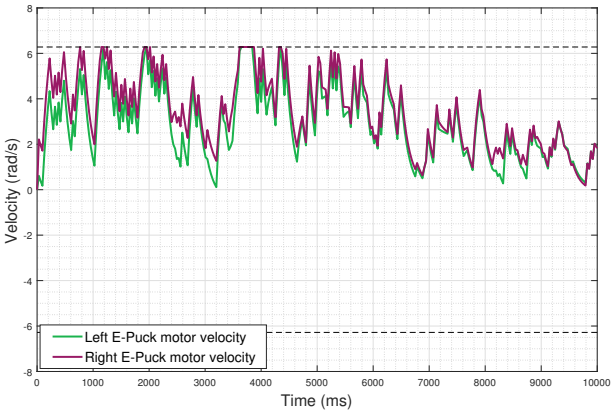

Fig. 10. E-Puck actuator velocities with LSPC

actuators. This last characteristic was directly related to the scaling factor $\bar{N}$ set at 0.01 in (6); the larger the value of this parameter, the larger the spikes in the velocities. Because of this small scaling value, the controller was slower than the others, so it was a trade-off situation between velocity stability and controller speed.

As shown in Fig.7, the TUC controller presented a lot of actuator saturation at 6.28 rad/s and rapid change spikes that could cause unwanted vibrations in real-life robots. The LSPC, as shown in Fig.10, reduced motor saturation but presented rapid unwanted spikes in the change of velocity. This was caused by the constant accelerations when the goal point for the robot moved with every PSO iteration. The TUC-LQR controller diminished the effects of these changes with the $\bar{N}$ factor multiplying the reference point as explained before. In the case of the TUC-PID controller, saturation was present in the actuators but the change spikes were greatly reduced as shown in Fig.9 compared to Fig.7. In this case, a velocity filter could be implemented without affecting the controller's performance to reduce rapid changes in velocity. If the change in actuator velocities $\Phi_{R,L}$ between iterations of the MPSO algorithm was greater than 1 rad/s, the velocities were recalculated as:

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

Some spikes can still be seen in Fig.9 after second 7, but this result was caused by the collision of E-Puck robots when almost all the swarm had already reached the goal. The TUC-PID controller was the second-best controller in terms of velocity performance.

VII. Conclusions

The PSO parameters that caused the robotic swarm to have 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 controller that caused more straightforward paths towards a goal in the problem space was the TUC-LQR controller. It also resulted in the most regulated E-Puck actuator velocities, with no saturation and no velocity change spikes. Although, this was the slowest and the most rigid controller. For search applications with low amount of obstacles, this might be the appropriate controller.

The LSPC controller resulted in curved smooth paths that could be adequate for goal search applications with a larger number of obstacles. Nevertheless, 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 such as they are, is not recommended for future applications.

The TUC-PID controller resulted in more irregular paths than the LSPC and TUC-LQR controllers, but the convergence time did not vary. This controller also resulted in well regulated velocities compared to the TUC and LSPC. It is also a more dynamic controller than the TUC-LQR, so its use along the MPSO algorithm[a] is recommended for future E-Puck swarm applications in search spaces with multiple obstacles.

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] F. Martins and A. Brandão, "Velocity-based dynamic model and control," 2018.

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

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

[11] 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.

[12] A. Aguilar, "Algoritmo modificado de optimizacin de enjambre de partculas," pp. 50–51, UVG, 2019.

[a]The complete code used in Webots including Algorithm 1 (MPSO) can be found in https://github.com/AldoAguilar001/MPSO-Algorithm