PSO Trajectory Planner Using Kinematic Controllers that Ensure Smooth Differential Robot Velocities

Aldo Aguilar, Miguel Zea, *Member, IEEE*, and Luis Alberto Rivera, *Member, IEEE*
Department of Electronics, Mechatronics and Biomedical Engineering
Universidad Del Valle de Guatemala
Guatemala, Guatemala
{agu15170, mezea, larivera}@uvg.edu.gt

***Abstract*—A multi-agent differential robot system requires a definite algorithm to behave as a swarm with goal searching capabilities. The classic Particle Swarm Optimization (PSO) algorithm is designed for particles with no mass or physical dimensions unlike differential robots. Therefore, it cannot be directly used to search for a goal using a robotic swarm. We propose the use of the PSO as a trajectory planner to enable the agents to collectively find the optimal path to a goal. The proposed PSO trajectory planner uses multiple known parameters such as inertia weight, a constriction parameter, two scaling factors and two random weighing factors (for the cognitive and social components of the PSO) optimized for differential robots. The planner takes into account the restrictions derived from the kinematic equations of the robotic agents and the finite dimensions of the search space. For that purpose, it is coupled with the necessary controllers to map particle velocities into smooth and continuous differential robot velocities. Four different controllers were tested, including the Transformed Unicycle Controller (TUC), Transformed Unicycle with LQR (TUC-LQR), Transformed Unicycle with LQI (TUC-LQI), and a Lyapunov-stable Pose Controller (LSPC). The TUC-LQI controller outperformed the others in terms of achieving smooth continuous differential robot velocities, while following the paths generated by the PSO trajectory planner.**

***Index Terms*—PSO, Trajectory planner, Differential robots**

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. It is designed for swarms of particles with no mass or physical dimensions, unlike differential robots. Robots' movements are restricted, as opposed to particles, which can move in any direction at any velocity. The main goal of this work is to adapt the PSO algorithm to be used for swarms of differential robots, in goal searching applications. To achieve this, the use of a PSO trajectory planner (PSO-TP) algorithm is proposed, to make the robots search for a path that minimizes the cost of reaching the goal. The goal corresponds to the absolute minimum of a fitness function.

Three aspects are key when coupling the PSO-TP algorithm to differential robots (i.e. E-Puck robots [1]): the kinematic restrictions of the swarm agents, the speed saturation limits of their wheels and the rate at which the wheels can change speed. To take these aspects into account, a set of kinematic controllers were designed and tested. The controllers are in charge of calculating the differential robot velocities using particle velocities as inputs. We compared the performance of a Transformed Unicycle proportional Controller (TUC), Linear-Quadratic Regulator (LQR) and Linear-Quadratic-Integral (LQI) controllers for the diffeomorphic model of a differential robot [2], and a pose controller with Lyapunov stability. The goal was to determine which controller performs better when it comes to following the goal path generated by the PSO trajectory planner, while at the same time, reducing the saturation and hard stops in the velocities of the robots' actuators.

The rest of the paper is organized as follows: Background is presented in Section II. Section III shows the structure and implementation of the PSO-TP. Details of the kinematic controllers used are given in Section IV. In Section V, the experiments to validate the PSO-TP implementation via the controllers are described, and the corresponding performance results are presented. Conclusions are given in Section VI.

II. Background

A. Classic PSO

The classic PSO algorithm was first proposed in [3]. It consists of a set of possible solutions (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 three components: the current velocity vector of the particle $\mathbf{v}_i$, the cognitive factor, which is the distance between the particle's current position $\mathbf{x}_i$ and the best position found locally by the particle $\mathbf{p}_\ell$, and the social factor, which is the distance between

$\mathbf{x}_i$ and the best position found globally by the entire particle swarm $\mathbf{p}_g$.

$$\mathbf{v}_{i+1} = \varphi[\omega \mathbf{v}_i + c_1\rho_1(\mathbf{p}_\ell - \mathbf{x}_i) + c_2\rho_2(\mathbf{p}_g - \mathbf{x}_i)], \quad (1)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1}. \quad (2)$$

Equation (1) is the update rule for the particle's velocity. φ is a constriction parameter to limit the maximum step that a particle can take when updating its position inside the problem space [4]. ω is an inertia parameter that weighs the contribution of the current velocity in the calculation of the particle's new velocity [5]. c_1 and c_2 are scaling parameters [6] for the cognitive and social factors in Eq. (1). ρ_1 and ρ_2 are random weighing factors, in the interval (0,1], which add a key dynamic 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 Eq. (2), where the new position vector $\mathbf{x}_{i+1}$ is the sum of the particle's current position $\mathbf{x}_i$ vector and the new velocity vector $\mathbf{v}_{i+1}$.

The PSO is ideal for swarm intelligence optimization applications. 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.

Previous works have proposed the use of the PSO as a tool for goal searching applications using a robotic swarm. In [7], the authors presented a 2-layer PSO based on the idea that the inner PSO layer is the one sending velocity and position information to the robots, and the outer PSO layer is in charge of finding the optimal PSO parameters for the inner layer. This approach ensures that the robotic swarm will not overshoot the goal because of the implementation of dynamic PSO parameters instead of static ones. It also ensures good balance between exploration of the search space and exploitation of the goal. In [8], the authors proposed the use of a hybrid algorithm combining the Ant Colony Optimization algorithm and the PSO. This approach facilitates the tracking of one or multiple targets using a robotic swarm. However, the authors mention that the communication overhead is a limitation of this approach due to the amount of parameters used to control the swarm.

Both previous investigations present novel ways of implementing the PSO for robotic swarms. However, they do not take into consideration the kinematic restrictions of real robots and do not present any analysis of actuator velocity smoothness. With this work, we aim at exploring the use of the PSO for robotic swarms considering these restrictions while also minimizing the amount of parameters necessary for the algorithm's execution. We also aim at analyzing actuator velocity smoothness to ensure that real-world wheel actuators of differential robots can actually execute each movement required to reach the tracked goal.

B. Kinematics of Differential Robots

Differential robots are non-holonomic systems having three degrees of freedom (DOF) and only two wheel actuators. Therefore, velocity mapping is required to transform the linear (v) and angular (ω) velocities of the robot into angular velocities of the robot's wheels ($\dot{\phi}_r$, $\dot{\phi}_\ell$). This is done by using the well-known unicycle robot model [2].

C. Diffeomorphism of Differential Robot Kinematics

The differential robot model is highly nonlinear and not controllable. Therefore, a diffeomorphism of the original system into a simpler system is required. 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]. The kinematic equations transforming the planar velocities of the point into differential robot velocities are the following:

$$v = u_1 \cos(\theta) + u_2 \sin(\theta), \quad (3)$$

$$\omega = -\frac{u_1}{\ell}\sin(\theta) + \frac{u_2}{\ell}\cos(\theta), \quad (4)$$

where θ is the robot's orientation and ℓ is the radius of the robot's base.

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 and have resulted in the correct translation of a robot. Control rules can be optimized for velocity smoothness, trajectory smoothness or both [10].

1) Point to point planar velocities: Planar velocity inputs can be calculated using the error between the goal position and the actual position of the controlled object. In bi-dimensional systems, the planar velocities can be computed as follows:

$$\mathbf{u} = I \tanh(k\mathbf{e}), \quad (5)$$

where $\mathbf{e}$ is the vector of the position errors between the goal's coordinates and the robot's position [9]. I is a saturation constant for the hyperbolic tangent limiting the output and k is a proportional scaling factor. The $\tanh()$ function is used to limit the output of the control to I, in case $\mathbf{e}$ is too large. The velocities can also be computed using LQR control as follows:

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

using the negative state feedback structured by multiplying the resulting LQR matrix $\mathbf{K}$ and the error between the system state $\mathbf{x}$ (robot coordinates) and the operational point ($\mathbf{x}_g$, $\mathbf{u}_g$). The LQR performance can be improved by adding integral action and turning it into an LQI controller with the following control law:

$$\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{x}_g) - \mathbf{K}_I\mathbf{z}_i + \mathbf{u}_g, \quad (7)$$

where $\mathbf{z}_i$ is the integral of the difference between a set reference and the actual state of the system calculated during a certain period of time [11].

2) 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 polar coordinates parameters: ρ, α and β. Further descriptions of these parameters and the pose control laws can be found in [10]. Some variants of this standard pose controller have been developed. In [12], 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:

$$v = k_\rho \rho \cos(\alpha), \quad k_\rho > 0, \tag{8}$$

$$\omega = k_\rho \sin(\alpha)\cos(\alpha) + k_\alpha \alpha, \quad k_\alpha > 0. \tag{9}$$

III. PSO Trajectory Planner

The classic PSO algorithm generates highly irregular trajectories for the swarm particles when updating their position, so following these paths with differential robots would be nearly impossible considering their kinematic restrictions. These restrictions include their mass, size, number of actuators and actuator saturation. However, the PSO velocity vectors generated by computing Eq. (1) still work as pointers to new PSO positions $\mathbf{x}_{i+1}$ given by Eq. (2). These multiple reference points can be easily tracked by a differential robot. Based on this idea, the PSO trajectory planner is designed as an algorithm that is tasked with simply updating the position of the reference point, or PSO Marker. The PSO marker is then tracked by the kinematic controller of the differential robot.

The PSO-TP starts by computing the first position of the PSO Marker $\mathbf{x}_{i+1}$ using Eqs. (1) and (2). This PSO Marker is then taken by the kinematic controller of the robot as the goal state that the system state $\mathbf{x}_i$ needs to reach. The kinematic controllers use the error between $\mathbf{x}_{i+1}$ and $\mathbf{x}_i$ to generate the control signals, being $\mathbf{x}_{i+1}$ the equilibrium point where the error is zero. While driving towards equilibrium, the robot will describe a smooth and continuous path. After a certain amount of iterations of the kinematic controller, the PSO-TP becomes active again and it updates the location of the PSO Marker $\mathbf{x}_{i+1}$. The kinematic controller starts tracking the new reference point and describes another continuous segment of the trajectory. This process is repeated continuously.

As the PSO algorithm converges to the absolute minimum of the fitness function, the PSO Marker converges to this goal as well. Therefore, it drives the robot towards this same point by concatenating each continuous segment described by the agent while tracking each location of the PSO Marker. Note that if the update frequency of the PSO-TP gives the kinematic controller enough time to reach each exact PSO Marker position, the robot would describe the same irregular path of a PSO particle. Therefore, the update frequency of the PSO-TP needs to be increased so that the kinematic controller can only drive the robot towards the current PSO Marker position during a short period of time generating a reduced portion of the trajectory needed to reach the marker. These concatenated short trajectory portions create a smooth and continuous path towards the goal for the robot to follow.

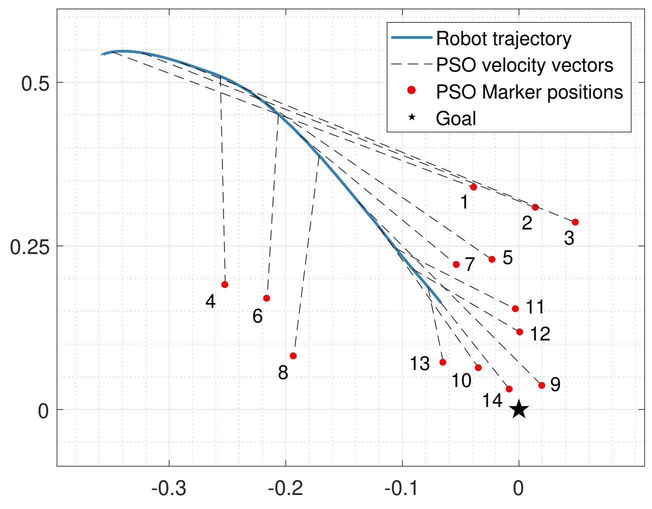

Fig. 1. Robot trajectory using the PSO-TP

Figure 1 illustrates the idea above. A robot navigates towards the goal, denoted by the black star at $(0, 0)$, following a smooth path comprised of short trajectory segments that track each updated position of the PSO Marker, shown as red dots in the plot.

A. *Tuning the PSO-TP*

There are several parameters that influence the behaviour of the PSO trajectory planner. The update rule for the PSO velocity shown in Eq. (1) has four parameter that need to be tuned. For the inertia parameter ω we used the Linear Decreasing Inertia Weight formula presented in [13], inside the interval $(0, 1]$ as recommended in [14] for stability, to reduce the convergence error of the robotic swarm. The scaling parameters c_1 and c_2 were set to a value of 2.05, as used in [4]. The constriction factor φ was calculated using the formula presented in [4]. It is worth noting that the sum of c_1 and c_2 is greater than 4, guaranteeing convergence of the PSO algorithm with the resulting value of φ [15].

In addition to the native parameters of the PSO, there are other added parameters to tune the PSO-TP. A new scaling factor η multiplying $\mathbf{v}_{i+1}$ in Eq. (2) is added to have a direct control over the length of the PSO velocity vectors, thus limiting the acceptable distances between the robot and the PSO Marker. This was done inspired by the boundary constraint techniques presented in [16] to avoid the PSO Marker surpassing the search space boundaries and causing the kinematic controllers to saturate the robots' actuators. The modified update rule for the particle's position is given by:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \eta \mathbf{v}_{i+1}. \tag{10}$$

As explained before, the update frequency of the PSO Marker is directly related to the smoothness of the resulting path of the robot towards the goal. Therefore, a parameter P_s is used in the algorithm to directly configure the amount of iterations of the kinematic controllers that will be executed between each activation of the PSO-TP.

IV. Kinematic Controllers

As explained in the previous section, the PSO-TP returns the position $\mathbf{x}_{i+1}$ of the PSO Marker. The kinematic controllers use the error between this position and the robot's current position $\mathbf{x}_i$ to compute the control signals. Four different controllers were tested to determine the best in terms of convergence speed, trajectory smoothness and smooth wheel velocities (less saturation and hard stops of the robot actuators).

A. Transformed Unicycle Controller (TUC)

The TUC combines Eq. (5) with the diffeomorphism Eqs. (3) and (4) to compute the robot's linear and angular velocities. In this case, the saturation constant used in Eq. (5) was arbitrarily set to $I = 2$, and the proportional scaling factor k was calculated as follows:

$$k = \frac{1 - e^{-2||\mathbf{e}||}}{2||\mathbf{e}||}, \tag{11}$$

so k would decrease as the robot approached the goal.

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

The TUC-LQR combines the LQR controller from Eq. (6) with Eqs. (3) and (4). Since the diffeomorphism assumes $\dot{\mathbf{x}} = \mathbf{u}$, the LQR gain matrix $\mathbf{K}$ was computed using $\mathbf{A} = \mathbf{0}$, $\mathbf{B} = \mathbf{R} = \mathbf{I}$, and $\mathbf{Q} = 0.1\mathbf{I}$. In this case, $\mathbf{x}_g = \mathbf{x}_{i+1}$, $\mathbf{x} = \mathbf{x}_i$ and $\mathbf{u}_g = 0$.

C. Transformed Unicycle with LQI Controller (TUC-LQI)

This controller combines the LQI Eq. (7) with Eqs. (3) and (4). The LQI gain matrices $\mathbf{K}$ and $\mathbf{K}_i$ were computed using:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{0} \end{bmatrix}, \quad \tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix}, \tag{12}$$

$$\mathbf{Q} = \mathbf{I}_{4\times4}, \quad \mathbf{R} = 2000\mathbf{I}_{2\times2}, \tag{13}$$

where diffeomorphism matrices are $\mathbf{A} = \mathbf{0}$ and $\mathbf{B} = \mathbf{I}$. An output matrix $\mathbf{C} = \mathbf{I}$ is used since the GPS sensors allow the direct tracking of the robots' current state $\mathbf{x}_i$. The integral error $\mathbf{z}_i$ is computed by the integration of the difference between the robot's current position $\mathbf{x}_i$ and the swarm's global best position $\mathbf{p}_g$. This was done to give stability to the actuator velocities of the robot by adding continuous movement towards the goal. A new proportional control damping coefficient $b_p \in (0, 1]$ is introduced to diminish sudden actuator velocity spikes caused by the PSO Marker position updates. An integral control damping coefficient $b_i \in (0, 1]$ is also introduced in order to reduce robot position oscillations once the goal is reached. The LQI equations are rewritten as follows:

$$\mathbf{u} = -\mathbf{K}(1 - b_p)(\mathbf{x}_i - \mathbf{x}_{i+1}) - \mathbf{K}_I\mathbf{z}_i, \tag{14}$$

$$\mathbf{z}_{i+1} = (1 - b_i)[\mathbf{z}_i + (\mathbf{p}_g - \mathbf{x}_i)\Delta t], \tag{15}$$

where damping coefficient values were set to $b_p = 0.95$ and $b_i = 0.01$.

D. Lyapunov-stable Pose Controller (LSPC)

This controller used the control laws from Eqs. (8) and (9) to calculate the linear and angular velocity of the E-Puck robots directly. In this case, position and orientation errors between the goal and the robot's location were mapped to polar coordinates as follows:

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

$$\alpha = -\theta + \arctan 2(e_y, e_x), \tag{17}$$

where e is the position error and α is in the interval $[-\pi, \pi]$. The controller's parameters in Eqs. (8) and (9) were set to: $k_\rho = 0.01$ and $k_\alpha = 0.5$. If α pointed at the left quadrants of the XY plane, the sign of the linear velocity was inverted.

V. Experiments and Results

We used the Webots simulation environment for our experiments [17]. Webots allows modeling the physical parameters of differential robots and others. After selecting the proper PSO-TP parameters to achieve the desired swarm behavior, we analysed the PSO-TP implementation on differential robots using the kinematic controllers described in the previous section for velocity mapping.

The simulations were performed using swarms of 10 E-Puck robots in a 2×2 m space and a sampling period (T_s) of 32 ms. The number of agents used in the simulations was determined according to the dimension constrains of the simulated search space to avoid overcrowding. The Sphere benchmark function with absolute minimum at $(0, 0)$ was used as fitness function $f(\cdot)$ to evaluate the algorithm's performance [18]. Simulated GPS and compass sensors were used to obtain position and orientation information of each E-Puck agent [17]. All agents were equipped with a simulated radio receiver to implement a fully connected communication topology in the swarm. This allowed the transmission and reception of each agent's local best position $\mathbf{p}_l$ and consequent polling for the global best position $\mathbf{p}_g$ found by the swarm.

We conducted experiments to evaluate trajectory smoothness, control smoothness and control saturation rate, while tracking the trajectory computed by the PSO-TP. For each case, we ran simulations using the four kinematic controllers, to compare their performance. The selected PSO-TP parameters for each controller are shown in Table I.

TABLE I
PSO-TP configuration depending on kinematic controller

Kinematic Controller	PSO-TP configuration parameters	
	P_s (iterations)	η
TUC	1	0.625
TUC-LQR	5	0.250
TUC-LQI	1	0.250
LSPC	5	0.250

P_s was set to 5 iterations for the TUC-LQR and LSPC to give the actuators 160 ms between velocity shifts, since these controllers showed rapid velocity spikes caused by the PSO

Marker update. The scaling factor η was set to 0.25 m for the TUC-LQR, TUC-LQI, and LSPC, to ensure that the PSO Marker did not surpass the 2×2 m search space's boundaries. The TUC controller used $\eta = 0.625$ m to stretch the distance between the robot and the PSO Marker with the intention of of generating more ample trajectories to minimize early E-Puck collisions caused by the fast controller.

A. Trajectory Smoothness

To evaluate trajectory smoothness, 10 simulations per controller were performed. The E-Pucks were located at different distances and orientations from the goal to vary initial conditions for each robot in every simulation. Figures 2-5 show typical trajectory smoothness results for each kinematic controller tested in the simulations. The 2×2 m grid represents the problem space, the blue lines are individual E-Puck trajectories, the red dots are the initial positions of the 10 E-Puck robots, and the black star at $(0, 0)$ is the absolute minimum of the Sphere fitness function.

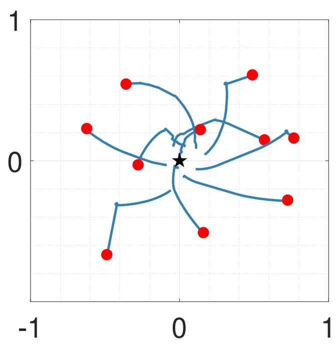

Fig. 2. TUC

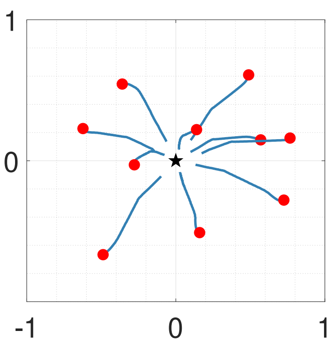

Fig. 3. TUC-LQR

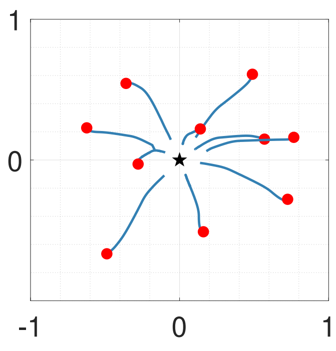

Fig. 4. TUC-LQI

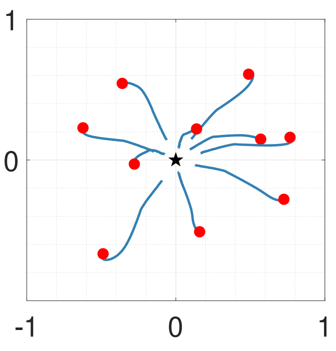

Fig. 5. LSPC

The controllers that tended to generate straighter paths towards the goal were the TUC-LQR and the TUC-LQI. The LSPC generated slightly smoother curved trajectories. The TUC controller resulted in more irregular trajectories but it achieved the fastest convergence time. The average time and standard deviation over the 10 runs were 7.97 s and 1.95. Average convergence times and standard deviations for the other controllers were: 26.69 s (1.27) for LSPC, 26.23 s (2.24) for the TUC-LQR, and 24.94 s (2.73) for the TUC-LQI. These last three controllers were considerably slower than the TUC but generated smoother, more regular trajectories.

B. Control Smoothness

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 all resultant actuator velocity signals ($\dot{\phi}_r$ and $\dot{\phi}_\ell$) were performed for each controller. Cubic splines have a minimum energy property based on the Euler-Bernoulli beam theory, as shown in [19]. Consequently, the smoothness of a resultant velocity curve can 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 smaller W values indicate control smoothness. A controller with a small 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 interpolated control curves (two actuator velocities for each of the 10 E-Pucks) was calculated for each controller using Eq. (18).

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

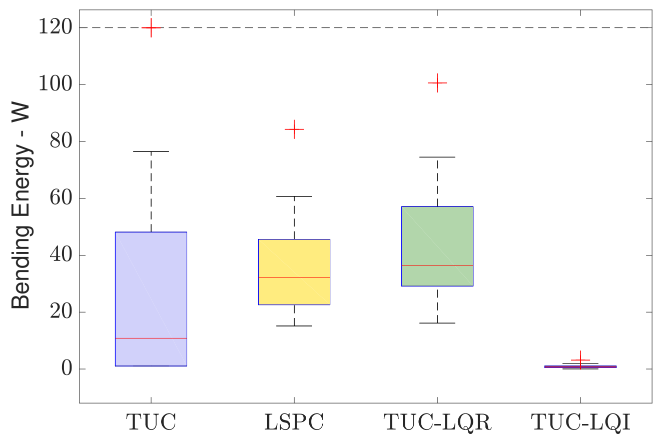

Fig. 6. Control smoothness calculated with bending energy

Figure 6 shows the smoothness results for all velocity signals generated with each controller. The TUC-LQI controller presented the lowest average bending energy results, i.e. greater smoothness. Furthermore, it presented a small standard deviation, which indicates little dependency on the initial positions of the E-pucks. The other three controllers showed similar smoothness distributions. The TUC showed the second lowest average bending energy, but the data presented greater standard deviation, indicating a stronger dependency of the control smoothness on the different initial conditions of the E-Pucks.

C. *Control Saturation Rate*

Another important aspect of the controllers' performance is the rate of actuator saturation caused by them. Every velocity control signal generated by each controller was analyzed to determine the ratio between the time that the actuators were saturated at ± 6.28 rad/s and the whole duration of the simulation. The TUC presented actuator saturation between 50% and 90% of the time, whereas the LSPC, TUC-LQR and TUC-LQI presented no saturation whatsoever.

D. *Further Testing of the PSO-TP with TUC-LQI Controller*

The implementation of the PSO-TP using the TUC-LQI controller was also tested using a modified Keane Benchmark function [20] with absolute minima at $\mathbf{x}^* = [\pm 0.7, 0]$. Figure 7 shows a typical run. In this case, the swarm converged to $\mathbf{x}^* = [+0.7, 0]$, since a robot was closer to that minimum, thus influencing $\mathbf{p}_g$ in Eq. (1). Control smoothness did not vary compared to previous experiments.

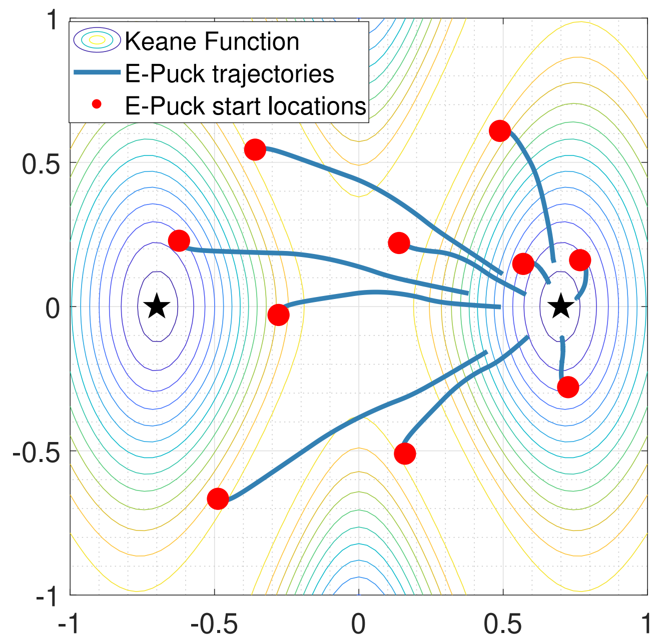

Fig. 7. Swarm trajectories in Keane benchmark function

We also introduced Gaussian noise to the GPS readings ($\mu = 0$, $\sigma_{x,y} = 0.1$), to simulate position errors, and to evaluate their impact on the trajectory planner and the controller. Figure 8 shows the smoothness results for the velocity signals generated by each controller. It can be observed that the smoothness of the signals generated by the TUC, LSPC and TUC-LQR controllers decreased significantly compared to the case where there was no GPS noise. However, the TUC-LQI controlled showed almost no variation in signal smoothness demonstrating its robustness. Figure 9 shows typical swarm trajectories and Figure 10 shows an example path followed by a single robot while tracking the generated PSO Markers, both using the TUC-LQI controller. Finally, Figure 11 shows the resulting velocities for each E-Puck's right actuator.

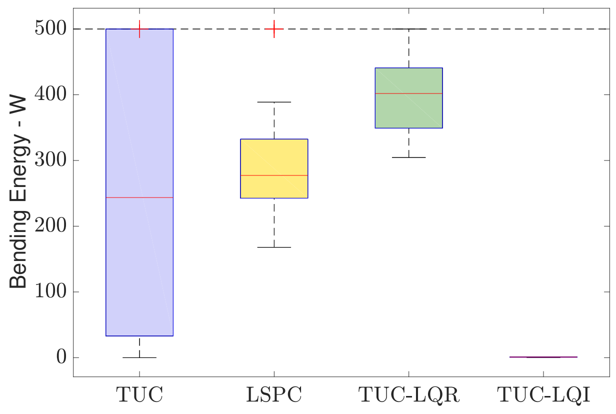

Fig. 8. Control smoothness, GPS with added noise

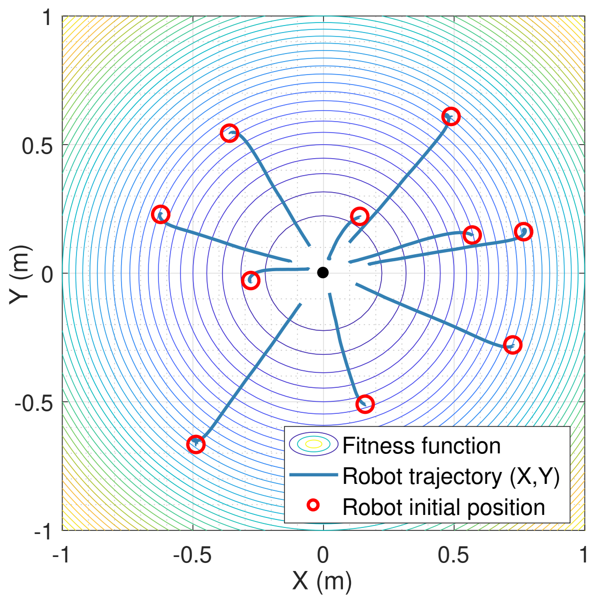

Fig. 9. Swarm trajectories, GPS with added noise

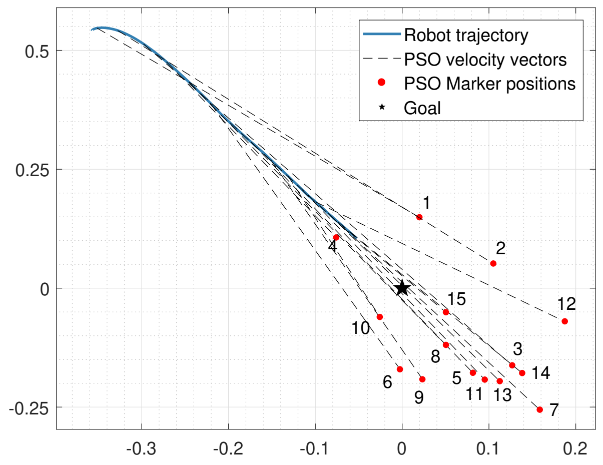

Fig. 10. Robot trajectory using the PSO-TP, GPS with added noise

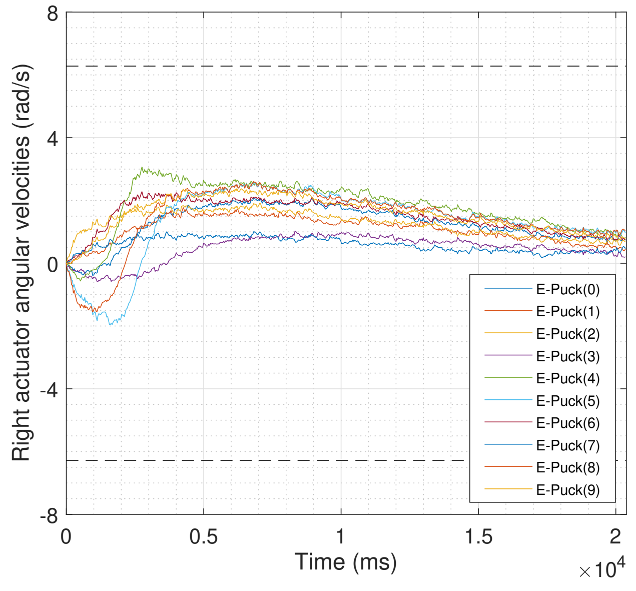

Fig. 11. Velocity smoothness, GPS with added noise

VI. Conclusions

The classic PSO algorithm is designed for particles with no kinematic restrictions unlike differential robots. For this reason, the PSO could not be directly implemented to a robotic swarm in goal searching applications. Therefore, we proposed the use of the PSO as a trajectory planner. As shown in the results, the PSO-TP was able to guide a differential robot swarm towards a goal in a finite amount of time generating smooth trajectories.

The classic PSO parameters incorporated to the PSO-TP (inertia weight ω, constriction factor φ and scaling factors c_1 and c_2) were set to default values used in previous works and the algorithm converged with great accuracy to the goal in a finite amount of time.

The implementation of the PSO Velocity scaling factor η allowed an easier manipulation of the operating range of the PSO-TP. It effectively restricted the positions of the PSO Marker so that they were always within the search space.

The TUC controller presented the fastest average convergence time, but it generated the most irregular paths while tracking the PSO Marker. It also presented a high percentage of control saturation, which would complicate its implementation in real robots, since actuators may be damaged.

The controllers that led to straighter and less curved paths towards the goal in the problem space were the TUC-LQR and TUC-LQI controllers. However, the TUC-LQI performed significantly better in terms of generating the smoothest actuator velocities, with no saturation, no velocity change spikes thanks to the proportional control damper b_p, and no oscillations when the goal was reached thanks to the integral control damper b_i. The LSPC performed similarly to the TUC-LQR in every sense.

The results of the experiments with added noise show that the system would be able to deal with a certain degree of error in the position measurements. The TUC-LQI demonstrated greater robustness compared to the other controllers. The generated trajectories, as well as the resulting velocity smoothness showed little change when the noise was added to the system.

References

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

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

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[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. IEEE, 2000, pp. 84–88.

[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)*. IEEE, 1998, pp. 69–73.

[6] T. Bartz-Beielstein, K. Parsopoulos, and M. Vrahatis, "Tuning PSO parameters through sensitivity analysis," *Universitätsbibliothek Dortmund*, Jan 2002.

[7] S. Doctor, G. K. Venayagamoorthy, and V. G. Gudise, "Optimal PSO for collective robotic search applications," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 2. IEEE, 2004, pp. 1390–1395.

[8] Y. Meng, O. Kazeem, and J. C. Muller, "A hybrid ACO/PSO control algorithm for distributed swarm robots," in *2007 IEEE Swarm Intelligence Symposium*. IEEE, 2007, pp. 273–280.

[9] F. Martins and A. Brandão, "Motion control and velocity-based dynamic compensation for mobile robots," *Applications of Mobile Robots*, Nov 2018.

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

[11] T. K. Priyambodo, "Integral modified linear quadratic regulator method for controlling lateral movement of flying wing in rotational roll mode," in *Journal of Engineering and Applied Sciences*. IEEE, 2018, pp. 463–471.

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

[13] 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*. IEEE, 2011, pp. 633–640.

[14] K. Yasuda and N. Iwasaki, "On the stability and the parameters of particle swarm optimization," in *Ant Colony Optimization and Swarm Intelligence. ANTS 2008. Lecture Notes in Computer Science*, vol. 5217. Springer, 2008.

[15] M. de Oca, T. Stützle, and M. Birattari, "A comparison of particle swarm optimization algorithms based on run-length distributions," in *Ant Colony Optimization and Swarm Intelligence. ANTS 2006. Lecture Notes in Computer Science.*, vol. 4150. Springer, 2006.

[16] E. Oldewage, A. Engelbrecht, and C. C.W., "Boundary constraint handling techniques for particle swarm optimization in high dimensional problem spaces," in *Swarm Intelligence. ANTS 2018. Lecture Notes in Computer Science, vol. 11172. Springer, Cham.* Springer, 2018.

[17] C. Ltd., "Webots: Commercial mobile robot simulation software," http://www.cyberbotics.com, accessed: 2019-05-30.

[18] M. Molga and C. Smutnicki, "Test functions for optimization needs (2005)," http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf, accessed: 2020-04-26.

[19] 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.

[20] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimization problems," *Int. Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013. [Online]. Available: http://arxiv.org/abs/1308.4008