Auto-tuning of PID controllers for Robotic Manipulators Using PSO and MOPSO

Ahmed Zidan, Svenja Tappe, and Tobias Ortmaier

Leibniz Universität Hannover, Institute of Mechatronic Systems, 30167 Hanover, Germany,
ahmed.zidan@imes.uni-hannover.de

Abstract. This work proposes two approaches to automatic tuning of PID position controllers based on different global optimization strategies. The chosen optimization algorithms are PSO and MOPSO, i. e. the problem is handled as a single objective problem in the first implementation and as a multiobjective problem in the second one. The auto-tuning is performed without assuming any previous knowledge of the robot dynamics. The objective functions are evaluated depending on real movements of the robot. Therefore, constraints guaranteeing safe and stable robot motion are necessary, namely: a maximum joint torque constraint, a maximum position error constraint and an oscillation constraint. Because of the practical nature of the problem in hand, constraints must be observed online. This requires adaptation of the optimization algorithm for reliable observance of the constraints without affecting the convergence rate of the objective function. Finally, experimental results of a 3-DOF robot for different trajectories and with different settings show the validity of the two approaches and demonstrate the advantages and disadvantages of every method.

Keywords: Robotic Manipulators, PSO, MOPSO, PID Control, Automatic Tuning

1 Introduction

PID control structures provide simple, robust and effective solutions for most applications of control engineering. As it was stated in [2], PID controller are with more than 95% share by far the most used controller in industrial processes. These good characteristics of PID controllers is conditioned by accurate tuning of the controller gains. However, it was shown in [6] that up to 80% of twenty six thousand PID controllers are not performing perfectly, one of the most important reasons is the poor tuning of the controllers.
Robotic manipulators are highly non linear, highly coupled, Multi-Input Multi-Output (MIMO) dynamic systems. Although PID controllers are widely used to control robotic manipulators, using the conventional tuning methods depending on manual or experimental approaches do not necessarily give satisfactory results for such complex systems [10].
The difficulty of using experimental and manual tuning methods rises in the application fields, where the assigned task of a robot might constantly change or where a robot is of variable configuration or geometry (e. g. modular robots). In such cases, the need for

an auto-tuning method is urgent.
Recently, after the rapid increase in computing power, auto-tuning methods based on optimization techniques have been applied to non linear systems in order to obtain an increased performance with respect to predefined fitness functions, which depend on the performed task. E. g. in tasks that involve trajectory tracking, the integral of the absolute error *IAE* or the integral of the square error *ISE* are widely used.
In the field of robotic manipulators, a number of optimization methods (e. g. Genetic Algorithms (GA) [12], Particle Swarm Optimization (PSO) [11]) has been used to automatically tune the PID controllers. Also, a number of researches introduced comparative studies between different algorithms, e. g. genetic algorithm GA is compared to simulated annealing SA in [13] and found to be giving the best tracking accuracy, while in [17], a comparison study of GA, PSO, and DE (Differential Evolution) is performed with respect to different performance-measuring functions and it is concluded that DE surpasses the other two algorithms. However, theses results are taken from simplified simulations of serial robots.
For some problems, optimizing with respect to only one objective function might not be sufficient. For example in trajectory tracking control, the main objective is to achieve the most possibly accurate tracking, but this accurate tracking might be associated with relatively high variance in the control action (joint torque) or even with high oscillations in the motion. Therefore, it might be helpful to take more than one objective function in consideration and handle the problem as a multiobjective optimization problem. [2] proposed an approach based on a multiobjective evolutionary algorithm (MOEA), which aimed to tune the PID controller gains by taking two conflicting objective functions into consideration: minimization of position errors and minimization of the control signal variation (joint torques). In [19], a comparative study between different multiobjective optimization techniques has been introduced and an improved multiobjective particle swarm optimization (I-MOPSO) has been proposed.
Artificial intelligence techniques like fuzzy logic and neural networks have also been implemented to build PID tuning systems for robotic manipulators. Examples for those systems can be found in [3], [14], and [15]. Those systems give the PID gains variable values depending on the online measurements of the robot joint positions and, therefore, turned the traditional controller into an adaptive controller.
Another approach has been proposed in [16]. Here a combination of both, the GA technique and the fuzzy logic, was used to form a hybrid tuning method for a PID regulator.
Regarding the optimization auto-tuning methods, the known and previously mentioned research test the proposed optimization algorithms only on simplified simulations of robots without considering the practical problems that rise from applying these algorithms on real robots. From a practical point of view, more attention needs to be diverted to the problem in hand, i. e. defining the necessary constraints, which guarantee a safe movement of the robot, and adapting the optimization algorithms in order to handle these constraints. Otherwise, these algorithms will not be practicable. We introduced in a previous work [23] a practical auto-tuning method for a PD controller using PSO algorithm. In this work, the previous approach is extended to PID controllers and is handled from two different perspectives: Firstly as a single objective problem (minimizing the tracking error) after considering sufficient constraints to avoid unwanted and/or un-

stable movements, and secondly as a multiobjective problem, where a second objective function (minimizing the variation of the control action) is introduced. Adding this objective function will restrict the gains from having very high values. These restrictions can not be defined as additional constraints since it is hard to define the maximum limits of gains in advance. The reminder of this paper is organized as follows. In Section 2, the optimization problem and the necessary constraints are defined, while Section 3 introduces the PSO and the MOPSO algorithms and the proposed method to handle the defined constraints. The results of several experiments is presented in Section 4. Finally, Section 5 discusses the conclusions of this work.

2 Robot Controller Optimization

This work considers a robot manipulator controlled by an independent PID controller for every individual joint. The traditional tuning methods (e. g. manual tuning, Ziegler-Nichols...) are unable to obtain critical damping behavior [5] and, therefore, settle for an overdamped one. Recently, more research is directed toward using global optimization algorithms to solve identification and designing problems, especially for complex nonlinear systems. The working principle of optimization methods is based on defining a searching algorithm, which aims to find the optimal values of the parameters that minimize an objective function in a predefined search space. To perform an auto-tuning of PID controllers using an optimization method, it is required to define the optimization problem including the objective function and the optimization parameters. In the problem in hand, it is desired to find the control parameters that lead to the best trajectory tracking accuracy of the robot. A widely known objective function to assess the tracking performance is the integral of the absolute error *IAE*:

$$IAE = \int_{t_s}^{t_e} |\boldsymbol{e}(t)| \mathrm{d}t = \int_{t_s}^{t_e} |(\boldsymbol{q}_{\mathtt{d}}(t) - \boldsymbol{q}(t))| \mathrm{d}t \ . \tag{1}$$

The PID control law is given by the following equation:

$$\tau = \boldsymbol{K}_{\mathtt{p}}(\boldsymbol{q}_{\mathtt{d}} - \boldsymbol{q}) + \boldsymbol{K}_{\mathtt{d}}(\dot{\boldsymbol{q}}_{\mathtt{d}} - \dot{\boldsymbol{q}}) + \boldsymbol{K}_{\mathtt{i}} \int_{t_s}^{t_e} (\boldsymbol{q}_{\mathtt{d}} - \boldsymbol{q}) \mathrm{d}t \ , \tag{2}$$

where $\boldsymbol{K}_{\mathtt{p}}$, $\boldsymbol{K}_{\mathtt{d}}$ and $\boldsymbol{K}_{\mathtt{i}}$ are diagonal matrices with their diagonal elements being the proportional, the derivative and the integral gains associated with the robot joints, $\boldsymbol{q}_{\mathtt{d}}$ is a vector of the desired positions, $\boldsymbol{q}$ is a vector of the measured positions, $\dot{\boldsymbol{q}}_{\mathtt{d}}$ is a vector of the desired velocities, $\dot{\boldsymbol{q}}$ is a vector of the measured velocities, t_s and t_e are the start and the end time of the desired trajectory, respectively, and τ is a vector of the joint torques. The optimization parameters are the diagonal elements of the matrices $\boldsymbol{K}_{\mathtt{p}}$, $\boldsymbol{K}_{\mathtt{d}}$ and $\boldsymbol{K}_{\mathtt{i}}$. The aim of this work is to implement a global optimization to auto-tune the PID controller for robotic manipulators. The proposed method will depend on iterative real movements of the robot along the desired trajectory to evaluate the objective function. For this sake, it is inevitable to define the necessary constraints that ensure the safety and the stability of the robot.

2.1 Definition of Constraints for the Optimization

The ideal approach to define the problem constraints is by modeling the dynamics of the system and designing the controller in order to keep the system stable and achieve a good tracking accuracy. For the proposed method, it is assumed that no knowledge of the system dynamics is available and, therefore, a model-free approach is required. From a practical point of view, one can determine the necessary constraints by monitoring the robot movement while searching for the optimal gains and stopping the movement if any unstable situation is detected. First constraint that comes into mind is to avoid big deviations from the desired trajectory. Similarly, it is helpful to restrict the controller output by allowing a maximum torque limit in order to avoid actuators saturation. Another important constraint is to avoid exciting dangerous oscillations by high gain values.

Detecting violations of the error and the torque constraints can be done simply by comparing the absolute values of the position errors and the motor torques to predefined maximum limits $e_{\mathtt{max}}$ and $\tau_{\mathtt{max}}$ respectively. A bigger challenge, however, is to detect oscillations in the robot movement. This can not be done analytically because no model of the robot dynamics is available. Besides, oscillations must be detected online, in order to stop the robot directly and avoid any possible damages.

We introduced in [23] a method to detect oscillations, which is basically based on the indexes introduced in [8] and [9]. The general concept will be summarized here, further description can be found in [23]. The main idea to detect oscillations is to divide the signal into positive and negative regions separated by zero-crossing points. If the signal is oscillating, then the regions between the zero-crossings will show some form of similarity between each other regrading their *IAE* values and the time intervals between the zero-crossings. Besides, the *IAE* values will be high enough to form the oscillating movement as shown in Figure 1. From this perspective, one can define an index to detect oscillations based on the following equations:

$$h_A(N_{\mathtt{zc}}) = \#\left\{ i < \frac{N_{\mathtt{zc}}}{2}; A_i < A_{i,\mathtt{max}} \wedge \alpha < \frac{A_{i+1}}{A_i} < \frac{1}{\alpha} \wedge \gamma < \frac{\delta_{i+1}}{\delta_i} < \frac{1}{\gamma} \right\} , \tag{3}$$

$$h_B(N_{\mathtt{zc}}) = \#\left\{ i < \frac{N_{\mathtt{zc}}}{2}; B_i < B_{i,\mathtt{max}} \wedge \alpha < \frac{B_{i+1}}{B_i} < \frac{1}{\alpha} \wedge \gamma < \frac{\varepsilon_{i+1}}{\varepsilon_i} < \frac{1}{\gamma} \right\} , \tag{4}$$

$$h(N_{\mathtt{zc}}) = \frac{h_A(N_{\mathtt{zc}}) + h_B(N_{\mathtt{zc}})}{N_{\mathtt{zc}}} , \tag{5}$$

where $\#S$ denotes the number of elements in the set S. A_i is the *IAE* of the positive area i of the error signal, B_i is the *IAE* of the negative area i of the error signal. δ_i and ε_i are the time durations of the positive and the negative areas i, respectively. $N_{\mathtt{zc}}$ is the number of the considered successive zero-crossings. $0 < \alpha < 1$ and $0 < \gamma < 1$ are tuning parameters define the degree of similarity. $A_{\mathtt{max}}$ and $B_{\mathtt{max}}$ are maximum limits of *IAE* of the regions between the zero-crossings. Theses limits are defined as follows:

$$A_{\mathtt{max},i}, B_{\mathtt{max},i} = \int_0^{\Delta t_{\mathtt{zc},i}} 0.01 e_{\mathtt{max}} sin(\frac{\pi t}{\Delta t_{\mathtt{zc},i}}) dt = \frac{0.02 \Delta t_{\mathtt{zc},i} e_{\mathtt{max}}}{\pi} , \tag{6}$$

with $\Delta t_{\mathtt{zc},i}$ being the time between the zero-crossings i and $i+1$, and $e_{\mathtt{max}}$ is the previously defined limit of the position error. It was suggested in [8] to chose $\alpha = 0.5 - 0.7$

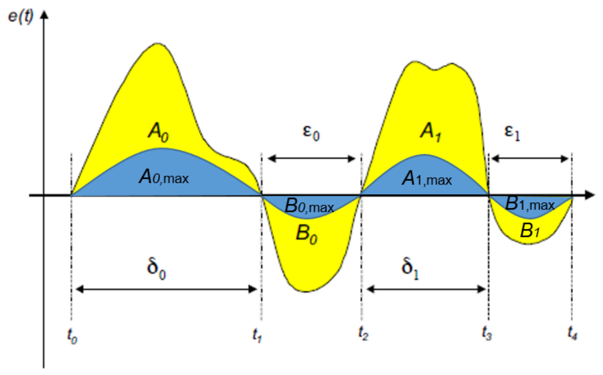


Fig. 1: The Proposed Oscillation Detection Method from [23] and Based on [8]

and $\gamma = 0.7 - 0.8$.
In order to guarantee that the error signal will oscillate around the zero value, it is chosen to use the difference between the error signal and its mean value $e(t) - \text{mean}(e(t))$ instead of using the error signal itself. Subtracting the mean value will rescale the oscillating signal around the zero axis.
The proposed procedure to detect oscillations is then achieved by the following steps:

1. After the beginning of the robot movement, record the values of the position error through a long enough period ΔT. A reasonable value of ΔT is $0.1T_{\mathtt{tot}} - 0.2T_{\mathtt{tot}}$, where $T_{\mathtt{tot}}$ is the total duration of the robot movement.
2. Calculate the function $e_{\mathtt{zc}}(t) = e(t) - \text{mean}(e(t))$ along the period ΔT.
3. Determine and count the zero-crossing points within ΔT, and calculate the values A, B, δ, ε between these points.
4. For every $N_{\mathtt{zc}}$ successive zero-crossings, calculate h_A, h_B and h.
5. If $h > h_{\mathtt{max}}$, oscillations exist and the movement must be stopped immediately. Otherwise, record the values of the position error for the next period ΔT and repeat the previous steps until reaching the end position of the robot.

It was suggested in [8] to choose $h_{\mathtt{max}} \in [0.4 - 0.8]$ and $N_{\mathtt{zc}} \geq 20$.

3 Particle Swarm Optimization

After defining the optimization problem and the necessary constraints, it is possible to choose one of the global optimization algorithms to find the optimal gains. The chosen algorithm in this work is the particle swarm optimization (PSO).
The PSO, first introduced in [7], is a population-based algorithm simulating the movement of a swarm of particles in a predefined space. After a number of iterations, the particles are attracted towards the best location which represents in the ideal case the global optimum.

The PSO has many features that make it efficient in solving optimization problems, e. g. it has less parameters to be identified in comparison with other optimization algorithms and has a very high success rate in finding the global minimum [20], and it has also a relatively high convergence speed to a near optima [1].
PSO has attracted a lot of research efforts, where also different versions and modifications have been proposed. Recently, optimization problems with multiobjectives have been taken into account and different approaches are proposed to form a multiobjective PSO (MOPSO).
The problem in hand is complicated, and performing the optimization with respect to only one objective function (*IAE*) may not achieve the desired task (trajectory tracking) perfectly. For example, an oscillating error signal around the zero-axis could have lower *IAE* than a signal with a steady-state error. However, it might be more desirable to tolerate a small steady-state error than to allow oscillations in the movement. This work gives two possible approaches to address the problem. In the first approach, the auto-tuning is performed by considering only *IAE* as an objective function, while trying to avoid undesirable oscillations by defining an oscillation constraint. In the second approach, a second objective function is added to the first one. The second objective function is the sum of the variations in the controller output and is defined in Subsection 3.2. The two objective functions contradict each other and, therefore, the optimization problem becomes multiobjective and is solved by finding a set of Pareto-front positions.

3.1 Optimization Using PSO

In the first implementation, only one objective function is considered which is the integral of the absolute error *IAE*. For every iteration of the PSO algorithm, every particle of the swarm will have a new position and velocity assigned to it based on the following equations:

$$\boldsymbol{V}_j(i+1) = \omega \boldsymbol{V}_j(i) + c_1\gamma_1(\boldsymbol{P}_j(i) - \boldsymbol{X}_j(i)) + c_2\gamma_2(\boldsymbol{G}(i) - \boldsymbol{X}_j(i)) \,, \tag{7}$$

$$\boldsymbol{X}_j(i+1) = \boldsymbol{X}_j(i) + \boldsymbol{V}_j(i+1) \,, \tag{8}$$

where i indicates the current iteration, j indicates a particle of the swarm, $\boldsymbol{X}_j(i)$ is the position vector of the particle j, $\boldsymbol{V}_j(i)$ is the velocity vector of the particle j, $c1$ and $c2$ are the cognitive and the social acceleration coefficients respectively, ω is the inertia factor and γ_1 and $\gamma_2 \in [0 \;\; 1]$ are random variables with uniformly distributed values.
There is no standard way to choose the swarm size and the maximum number of iterations. However, both parameters must be high enough in order to guarantee a convergence of the objective value toward the global minimum.
The inertia weight is defined to be a linear decreased function as follows:

$$\omega = \omega_{\mathtt{max}} - \frac{(\omega_{\mathtt{max}} - \omega_{\mathtt{min}})N_i}{N_{\mathtt{max}}} \,, \tag{9}$$

where $N_{\mathtt{max}}$ is the maximum number of iterations, N_i is the current number of iterations, $\omega_{\mathtt{max}}$ and $\omega_{\mathtt{min}}$ are the maximum and the minimum values of the inertia weight respectively. The chosen values in this work are $\omega_{\mathtt{max}} = 0.9$ and $\omega_{\mathtt{min}} = 0.4$ as it was suggested

in [21].
Based on the stability conditions that was introduced in [18], it is possible to define the parameters $c1$ and $c2$ in a way that guaranties a convergence toward an equilibrium point eventually. These conditions are:

$$0 < c1 + c2 < 4 , \tag{10}$$

$$\frac{(c1 + c2)}{2} - 1 < \omega < 1 . \tag{11}$$

Considering that $\omega \in [0.4 - 0.9]$, choosing $c1 = c2 = 1$ is suitable.

3.2 Optimization Using MOPSO

In the second implementation, a second objective function beside IAE is considered. This function is meant to limit the variation of the control action, which may change rapidly if the controller gains have very high values. This function is defined as follows:

$$f_2(i) = \sum_{i=1}^{i=M} |\tau_i - \tau_{i-1}| = \sum_{i=1}^{i=M} |\Delta\tau_i| , \tag{12}$$

with τ_i being the torque vector of the robot's joints, M being the total number of measurement samples during the robot movement, and i an index of the samples.
In this work, the MOPSO algorithm defined in [4] is applied. The only modification made here is to ignore the mutation step which is used to prevent the fast convergence of the algorithm. An external repository is used to store the nondominated solutions and an adaptive grid is constructed to produce well-distributed Pareto-fronts. The main contribution of this work is to adapt both PSO and MOPSO to form a practical auto-tuning mechanism for PID controller. The unusual form of the considered constraints requires a special method of handling for these constraints, which is introduced in the next subsection.

3.3 Constraints Handling

The mostly used methods to handle constraints in traditional optimization problems are either penalty-based methods, where a penalty value is added to the objective function depending on the violation of the constraints, or methods that try to define the feasible regions in the search space and restrict the optimization parameters to be always inside these regions. Unfortunately, the practical nature of the optimization problem in this work makes both methods unsuitable. As it was mentioned before, if one of the constraints is violated, the movement of the robot must be stopped immediately, whereas the objective function must be calculated along the entire trajectory for comparability, therefore, no penalty-based method can be used. Besides, the feasible region cannot be defined theoretically in advance and the only way to detect a violation of constraints is by performing the movement that results according to the particle position.
The here proposed method to handle the constraints is inspired by the work in [22], where some modifications of the optimization algorithm are suggested. The handling method is done after considering the following simplifications:

– Assuming that the coupling effects between the robot links can be ignored, one can state, that only the gains $K_{\mathtt{p}}^{l}$, $K_{\mathtt{d}}^{l}$ and $K_{\mathtt{i}}^{l}$ of the corresponding controller of link l are responsible if a violation occurs and, therefore, only these must be modified, i. e. handling the constraints can be done by modifying the particle position in only three dimensions.
– There is only one continuous feasible region inside the search space, i. e. if a particle moves in a direction that leads to a constraint violation, continuing to move the particle in the same direction will also lead to a constraint violation.
– Most of the initial swarm positions are located inside the feasible region.

Figure 2 demonstrates the case of a particle outside the feasible regions. For the sake of simplicity, the third dimension corresponding to $K_{\mathtt{i}}^{l}$ is not shown.
Based on the introductory simplifications, a proper method to bring the particles back to the feasible region can be achieved as follows:
If one of the constraints of the link l is violated, the exploration term $(\omega \mathbf{V}^{l})$ for the corresponding dimensions are set to zero, which will restrict the next movement to be only influenced by the personal and the global best positions. Off course these positions are located in the feasible region, therefore, the resulted velocity vector will bring the particle back to the feasible region.
The assumption of having only one feasible region is a reasonable assumption, because

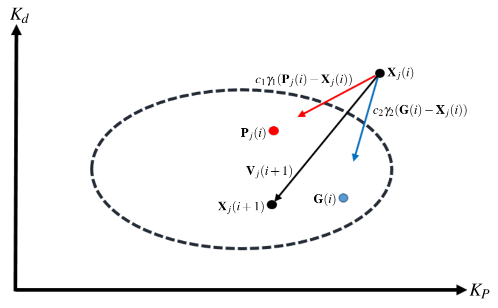


Fig. 2: Constraints Handling Method From [23]

it indicates that if a gain value violated one of the constraints after it was increased, then continuing to increase it (while keeping the other gains with the same values) will keep violating this constraint. The same applies for when the gain is decreased.
Based on the foregoing, it is now only required to locate the initial swarm inside the feasible region. A simple strategy is to define only one suitable position $\boldsymbol{K}_0 = \{\boldsymbol{K}_{\mathtt{p},0}, \boldsymbol{K}_{\mathtt{d},0} \boldsymbol{K}_{\mathtt{i},0}\}$ (e. g. by tuning the controller through trails and errors). For the other particles, one can define an interval $\pm\Delta\mathbf{k}$ in the neighborhood of this position, let it be for example $[\boldsymbol{K}_0 - \Delta\boldsymbol{K}; \boldsymbol{K}_0 + \Delta\boldsymbol{K}]$. Finally, random initial positions can be allocated with a unified distribution in this interval. It is possible that some of the initial positions may violate one or more of the constraints (if the chosen interval is too large), however,

they will return in the following iterations to the feasible region thanks to the proposed method.

4 Experimental Results

The proposed tuning method is tested on a 7-DOF robot, which is built of specially designed modules called PowerCube from the company "Schunk". To validate the efficiency of the proposed auto-tuning method, several experiments are performed. In these experiments, PID controllers are used to control the joints (3, 4, 6) which are shown in Figure 3. All the joints here are rotational and actuated by brushless dc-motors. In the

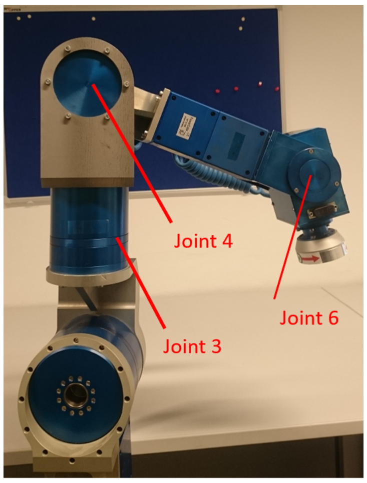


Fig. 3: PowerCube Robot from [23]

first experiment, point-to-point trajectories with sinusoidal velocity profiles are chosen as desired trajectories, see Figure 4a. Only one objective function is defined as follows:

$$f_1(i) = IAE = \sum_{i=1}^{M} |e_3(i)|\Delta T + \sum_{i=1}^{M} |e_4(i)|\Delta T + \sum_{i=1}^{M} |e_6(i)|\Delta T \ , \tag{13}$$

where $e_k(i)$ is the measured position error of the joint k, ΔT is the sample time, M is the total number of measurement samples during the robot movement and i is an index of the samples. To define the constraints for the chosen joints, the value $e_{\mathtt{max}} = \frac{\pi}{90} \approx 0.035\,rad$ is chosen as the maximum limit for the position error. The PowerCube modules provide the user with current measurements for every motor instead of torque measurements, that is why a maximum current constraints is used in the experiment instead of a maximum torque. The maximum allowed current in all the modules equals $15A$, but to insure more safety conditions, only the half of this value is defined as the current limit ($I_{\mathtt{max}} = 7.5A$). Oscillations detection is performed as described previously by determining every 20 consecutive zero-crossings ($N_{\mathtt{zc}} = 20$) from the rescaled error signal $e_{\mathtt{zc}}(t)$ and calculating the oscillation index value $h(N_{\mathtt{zc}})$. if $h(N_{\mathtt{zc}}) > 0.6$ then oscillations are considered to be occurred and the movement of the robot must be stopped.

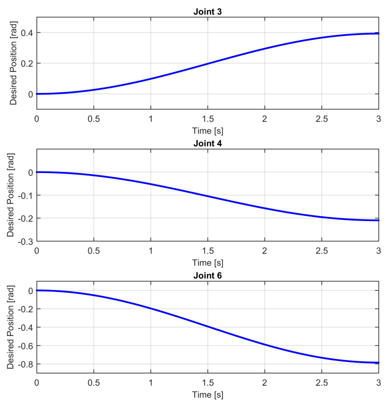
For this first experiment, PSO algorithm is applied with a swarm size of 10 particles and a maximum number of generations equals 20. To define initial positions for the particle swarm, an acceptable set of gain values are defined through trials and errors which are $\boldsymbol{k}_{\mathtt{p},0} = [90, 60, 120]$, $\boldsymbol{k}_{\mathtt{d},0} = [5, 2, 8]$ and $\boldsymbol{k}_{\mathtt{i},0} = [2, 1, 1]$. Based on these values, three intervals for the initial gains are defined as follows: $[\boldsymbol{k}_{\mathtt{p},0} - \boldsymbol{k}_{\mathtt{p},0}/2; \boldsymbol{k}_{\mathtt{p},0} + \boldsymbol{k}_{\mathtt{p},0}/2]$, $[\boldsymbol{k}_{\mathtt{d},0} - \boldsymbol{k}_{\mathtt{d},0}/2; \boldsymbol{k}_{\mathtt{d},0} + \boldsymbol{k}_{\mathtt{d},0}/2]$ and $[\boldsymbol{k}_{\mathtt{i},0} - \boldsymbol{k}_{\mathtt{i},0}/2; \boldsymbol{k}_{\mathtt{i},0} + \boldsymbol{k}_{\mathtt{i},0}/2]$.

The initial positions were then determined randomly from inside these intervals with a uniform distribution. The search space for the $\boldsymbol{k}_{\mathtt{p}}$ gains is defined to be $[1; 500]$ and for the $\boldsymbol{k}_{\mathtt{d}}$ and $\boldsymbol{k}_{\mathtt{i}}$ gains $[0; 50]$. I. e. the maximum gain limits are chosen to be relatively high compared to the initial gains in order to give the particles enough space to find the optimal gains. However, the maximum velocity value $V_{\mathtt{max}}$ is set to be equal 20% of these maximum limits in order to avoid big leaps in particle movement.

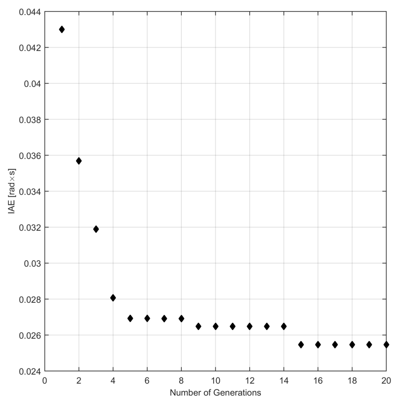
After applying the search algorithm, the following optimal gain values have been found: $\boldsymbol{k}_{\mathtt{p}} = [249.1, 109.9, 312.9]$, $\boldsymbol{k}_{\mathtt{d}} = [0.2, 1.1, 12.4]$ and $\boldsymbol{k}_{\mathtt{i}} = [2.1, 2.4, 4.8]$ with the minimal objective value of $IAE = 0.0255\,rad\,s$.

The convergence of the objective value during the searching procedure is shown in Figure 4b, while Figure 4c shows the position error diagram according to the optimal gain values in comparison with the position error of the initial gains. The accuracy of the tracking is clearly improved after the optimization. However, low-amplitude oscillations started to appear on the third joints.

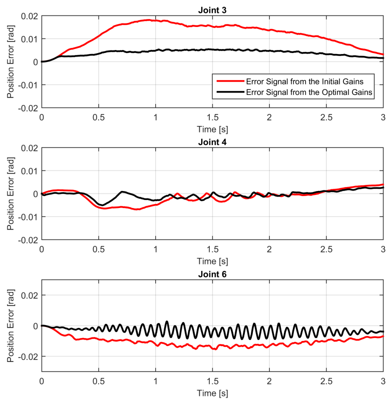
In the second experiment, different desired trajectories are applied, which are shown in Figure 5a. These trajectories generate a circular movement of the end effector. The same setup of the first experiment (constraint limits, swarm size and maximum number of generations) is applied here. The initial gains are defined based on trials and errors to be: $\boldsymbol{k}_{\mathtt{p},0} = [140, 120, 140]$, $\boldsymbol{k}_{\mathtt{d},0} = [1, 2, 2]$ and $\boldsymbol{k}_{\mathtt{i},0} = [1, 1, 1]$. The initial positions of the particles are then determined using the same previous interval. After doing the search for 20 generations, the objective value converged as it is shown in Figure 5b and reached the value: $IAE = 0.0692\,rad\,s$ which indicates a good improvement compared to the initial gains as shown in Figure 5c. The resulted optimal gains are: $\boldsymbol{k}_{\mathtt{p}} = [286.9, 121.9, 371.2]$, $\boldsymbol{k}_{\mathtt{d}} = [0.4, 0.1, 1.5]$ and $\boldsymbol{k}_{\mathtt{i}} = [2.2, 1.3, 1.6]$. It is noticeable that low amplitude oscillations still exist despite the used oscillation constraints. This indicates that the search algorithm pushed the gains (especially the proportional gains) to the limits where it excite oscillations under the tolerated limits. In order to demonstrate the effect of the oscillation index on the oscillations, a lower maximum limit is defined in the third experiment to be ($h_{\mathtt{max}}(N_{\mathtt{zc}}) = 0.4$. I. e. the conditions to detect oscillations will be more restrictive with respect to the tolerated oscillations. After that, the search for the optimal gains is repeated by keeping the other settings exactly the same as in the second experiment. After 20 generations, the following optimal gains resulted: $\boldsymbol{k}_{\mathtt{p}} = [188.8, 144.5, 246.7]$, $\boldsymbol{k}_{\mathtt{d}} = [2.1, 0.8, 1.7]$ and $\boldsymbol{k}_{\mathtt{i}} = [0.9, 0.6, 1.1]$. As expected, the final value of the objective function $IAE = 0.0869\,rad\,s$ is higher compared



(a) Desired Trajectories

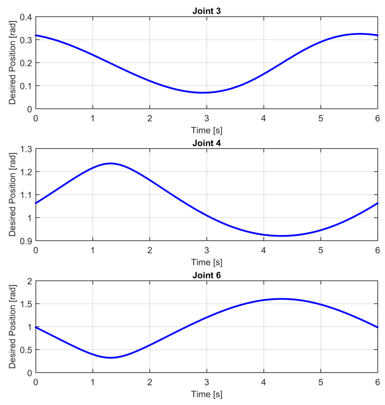


(b) Convergence of *IAE*

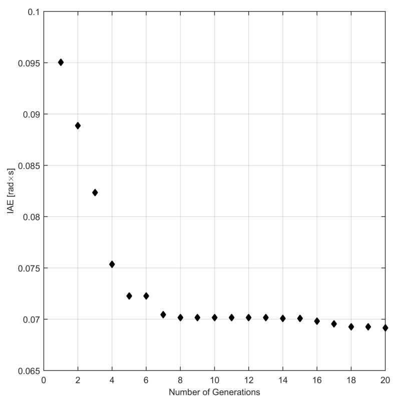


(c) Position Errors

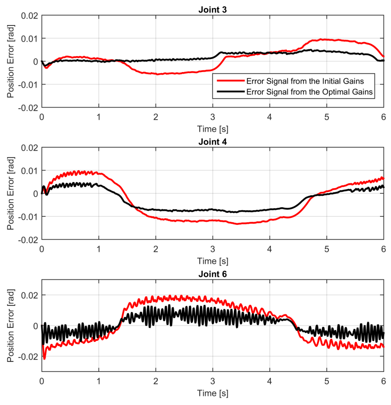
Fig. 4: Optimization of PID Controller for a Point-to-Point Movement Using PSO, with a): the Desired Trajectories of the Point-to-Point Movement, b): the Convergence of *IAE* after 20 Generations and c): the Position Errors Corresponding to the Optimal PID Gains in Comparison to the Position Errors of the initial Gains



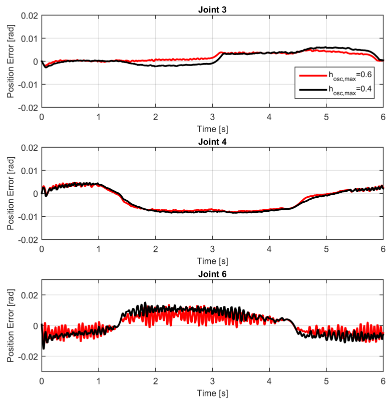
(a) Desired Trajectories



(b) Convergence of *IAE*

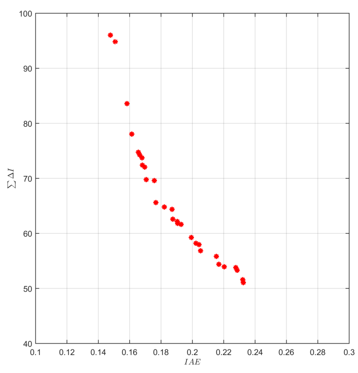


(c) Position Errors

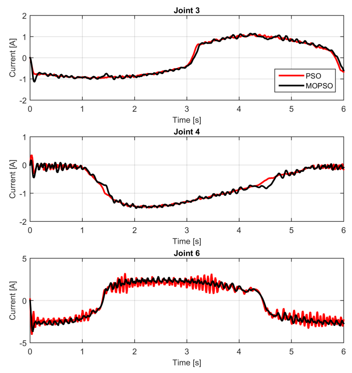


(d) Position Errors after Changing $h_{\mathtt{max}}$

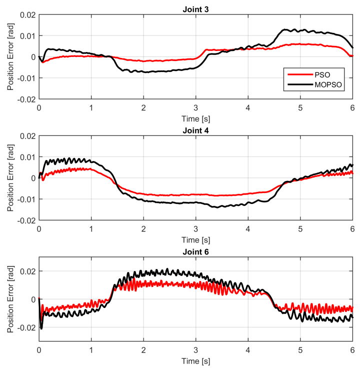
Fig. 5: Optimization of PID Controller for a Circular Movement Using PSO, With a): the Desired Trajectories of the Circular Movement, b): Convergence of *IAE* After 20 Generations, c): Position Errors Corresponding to the Optimal PID Gains in Comparison to the Position Errors of the initial Gains and d): Position Errors Corresponding to $h_{\mathtt{max}}(N_{\mathtt{zc}}) = 0.4$ in Comparison to the Position Errors of $h_{\mathtt{max}}(N_{\mathtt{zc}}) = 0.6$



(a) Pareto-Front Solutions



(b) Current Signals



(c) Position Errors

Fig. 6: Optimization of PID Controller for a Circular Movement Using MOPSO, With a) the Pareto-Front Solutions After 30 Generations, b): Current Signals Corresponding to the Optimal Gains from MOPSO in Comparison to the Current Signals from PSO and c): Position Errors Corresponding to the Optimal Gains from MOPSO in Comparison to the Position Errors from PSO

to the previous value. However, the oscillations are clearly reduced after changing the oscillation index as shown in Figure 5d.
In the fourth experiment, the auto-tuning is handled as a multiobjective optimization problem. The second objective function is calculated as follows:

$$f_2(i) = \sum_{i=2}^{N} |I_3(i) - I_3(i-1)| + \sum_{i=2}^{N} |I_4(i) - I_4(i-1)| + \sum_{i=2}^{N} |I_6(i) - I_6(i-1)| \,. \tag{14}$$

Again, a swarm of 10 particles is generated with same configuration as in the previous experiments. The main difference here is that not only one global optimum is searched but a group of solutions being called the Pareto-front solutions. Finding sufficient number of the Pareto-front solutions usually require more swarm generations compared to the normal PSO algorithm. Therefore, the maximum number of generations is increased to 30 which means that the robot performs the movement 300 times. After finishing all the iterations, a set of Pareto-front solutions is determined as shown in Figure 6a. To compare the results from this method with the results of the last experiment, the Pareto-front position that achieved the best tracking accuracy (lower *IAE* value) is chosen as the best solution, because improving the accuracy is still considered the main and the most important objective. The optimal gains corresponding to this position are: $\boldsymbol{k}_{\mathtt{p}} = [97.8, 115.48, 131.47]$, $\boldsymbol{k}_{\mathtt{d}} = [2.3, 2.3, 2.1]$ and $\boldsymbol{k}_{\mathtt{i}} = [2.9, 2.7, 3.1]$. Figure 6b shows the controller output of this Pareto solution compared to the output of the optimal gains in the third experiment. Adding the second objective function reduced the variation in the controller action. However, it also led to much lower accuracy ($IAE = 0.1559\,rad\,s$) as shown in Figure 6c. This is expected because of the trade-off between the two objective functions. So on one hand, taking the variance of the controller action into consideration makes the searching algorithms safer and stabler, because the algorithms does not push the gains to the limits of the feasible regions, and at the end it gives a group of solutions with different accuracies. On the other hand, using only one objective function led to better accuracy (two times better than MOPSO), but also led to higher oscillations and more variations in the controller output. A suggestion to get the best results might be to combine the two methods. First applying the MOPSO will show the boundaries between the two objective functions. Then one can define a maximum value of $f_2(i)$ based on the Pareto-fronts and set it as an additional constraints for PSO, then PSO can be implemented to get the best possible accuracy under this new constraint.

5 Conclusion

In this paper, an auto-tuning method of PID-controllers for robotic manipulators has been proposed. The suggested approach includes two implementations of a global optimization algorithm. In the first one, the particle swarm optimization is applied to find the optimal control parameters, while in the second one, a multiobjective PSO is applied. The main contribution of this work was to address the challenges arising, when using real experimental data for auto-tuning of PID controls. This includes necessary methods for guaranteeing stability and safety. Therefore, approaches for applying constrains, such as the maximum position error, the maximum joint torque and the oscillation constraint, have been proposed and tested. Additionally, a suitable way to handle these constraints has been suggested. Finally, the proposed approach has been experimented successfully on a real robot by applying different trajectories and different settings of the tuning method. Using PSO has improved the tracking accuracy but also led to low-amplitude oscillations and to high variation of the control action. Adding a second objective function and performing the tuning through a MOPSO algorithm generated a set of Pareto-front solutions with lower variation of the control action but also led to a lower accuracy in comparison to PSO.

References

1. Angeline, P.J.: Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. International Conference on Evolutionary Programming pp. 601–610 (1998)
2. Ayala, H.V.H., dos Santos Coelho, L.: Tuning of pid controller based on a multiobjective genetic algorithm applied to a robotic manipulator. Expert Systems with Applications **39**(10), 8968–8974 (2012)
3. Bekit, B., Seneviratne, L., Whidborne, J., Althoefer, K.: Fuzzy pid tuning for robot manipulators. pp. 2452–2457. IEEE (1998)
4. Coello, C.A.C., Pulido, G.T., Lechuga, M.S.: Handling multiple objectives with particle swarm optimization. IEEE Transactions on evolutionary computation **8**(3), 256–279 (2004)
5. Craig, J.J.: Introduction to robotics: mechanics and control, vol. 3. Pearson Prentice Hall Upper Saddle River (2005)
6. Desborough, L., Miller, R.: Increasing customer value of industrial control performance monitoring-honeywell's experience. 326, pp. 169–189. New York; American Institute of Chemical Engineers; 1998 (2002)
7. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. pp. 39–43. IEEE (1995)
8. Forsman, K., Stattin, A.: A new criterion for detecting oscillations in control loops. Control Conference (ECC), 1999 European pp. 2313–2316 (1999)
9. Hägglund, T.: A control-loop performance monitor. Control Engineering Practice **3**(11), 1543–1551 (1995)
10. Johnson, M.A., Moradi, M.H.: PID control. Springer (2005)
11. Kapoor, N., Ohri, J.: Improved pso tuned classical controllers (pid and smc) for robotic manipulator. International Journal of Modern Education and Computer Science **7**(1), 47 (2015)
12. Kim, E.J., Seki, K., Iwasaki, M., Lee, S.H.: Ga-based practical auto-tuning technique for industrial robot controller with system identification. IEEJ Journal of Industry Applications **1**(1), 62–69 (2012)
13. Kwok, D., Sheng, F.: Genetic algorithm and simulated annealing for optimal robot arm pid control. pp. 707–713. IEEE (1994)
14. Llama, A.M., Kelly, R., Santibañez, V.: A stable motion control system for manipulators via fuzzy self-tuning. Fuzzy Sets and Systems **124** (2001)
15. Melek, W.W., Goldenberg, A.A.: Neurofuzzy control of modular and reconfigurable robots. IEEE/ASME transactions on mechatronics **8**(3), 381–389 (2003)
16. Nahapetian, N., Motlagh, M.J., Analoui, M.: Pid gain tuning using genetic algorithms and fuzzy logic for robot manipulator control. Advanced Computer Control, 2009. ICACC'09. International Conference on pp. 346–350 (2009)
17. Ouyang, P., Pano, V.: Comparative study of de, pso and ga for position domain pid controller tuning. Algorithms **8**(3), 697–711 (2015)

18. Perez, R., Behdinan, K.: Particle swarm approach for structural design optimization. Computers & Structures **85**(19), 1579–1588 (2007)
19. Pierezan, J., Ayala, H.H., da Cruz, L.F., Freire, R.Z., Coelho, L.d.S.: Improved multiobjective particle swarm optimization for designing pid controllers applied to robotic manipulator. pp. 1–8. IEEE (2014)
20. Rezaee Jordehi, A., Jasni, J.: Parameter selection in particle swarm optimisation: a survey. Journal of Experimental & Theoretical Artificial Intelligence **25**(4), 527–542 (2013)
21. Shi, Y., Eberhart, R.C.: Parameter selection in particle swarm optimization. International Conference on Evolutionary Programming pp. 591–600 (1998)
22. Venter, G., Sobieszczanski-Sobieski, J.: Particle swarm optimization. AIAA journal **41**(8), 1583–1589 (2003)
23. Zidan, A., Kotlarski, J., Ortmaier, T.: A practical approach for the auto-tuning of pd controllers for robotic manipulators using particle swarm optimization. 14th International Conference on Informatics in Control, Automation and Robotics pp. 34–40 (2017)