PARTICLE SWARM OPTIMIZATION USED FOR MECHANISM DESIGN AND GUIDANCE OF SWARM MOBILE ROBOTS

Peter Eberhard and Qirong Tang
Institute of Engineering and Computational Mechanics,
University of Stuttgart, Pfaffenwaldring 9,
70569 Stuttgart, Germany
Email: [eberhard,tang]@itm.uni-stuttgart.de

November 13, 2009

Abstract

This chapter presents particle swarm optimization (PSO) based algorithms. After an overview of PSO’s development and application history also two application examples are given in the following. PSO’s robustness and its simple applicability without the need for cumbersome derivative calculations make it an attractive optimization method. Such features also allow this algorithm to be adjusted for engineering optimization tasks which often contain problem immanent equality and inequality constraints. Constrained engineering problems are usually treated by sometimes inadequate penalty functions when using stochastic algorithms. In this work, an algorithm is presented which utilizes the simple structure of the basic PSO technique and combines it with an extended non-stationary penalty function approach, called augmented Lagrangian particle swarm optimization (ALPSO). It is used for the stiffness optimization of an industrial machine tool with parallel kinematics. Based on ALPSO, we can go a further step. Utilizing the ALPSO algorithm together with strategies of special velocity limits, virtual detectors and others, the algorithm is improved to augmented Lagrangian particle swarm optimization with special velocity limits (VL-ALPSO). Then the work uses this algorithm to solve problems of motion planning for swarm mobile robots. All the strategies together with basic PSO are corresponding to real situations of swarm mobile robots in coordinated movements. We build a swarm motion model based on Euler forward time integration that involves some mechanical properties such as masses, inertias or external forces to the swarm robotic system. The results show that the stiffness of the machine can be optimized by ALPSO and simulation results show that the swarm robots moving in the environment mimic the
real robots quite well. In the simulation, each robot has the ability to master target searching, obstacle avoidance, random wonder, acceleration/deceleration and escape entrapment. So, from this two application examples one can claim, that after some engineering adaptation, PSO based algorithms can suit well for engineering problems.

1 Introduction

Particle swarm optimization (PSO) originated in different areas, among them Complex Adaptive Systems (CAS). The theory of CAS was formally proposed in 1994 by Holland and later collected in his book in (1995). PSO was inspired by the main ideas of CAS and from Heppner and Grenander’s biological population model (Heppner and Grenander, 1990). In a complex adaptive system, like a swarm of birds, see Figure 1, each member is taken as an adaptive agent and members of CAS can communicate with the environment and other agents. During the process of communication, they will ‘learn’ or ‘accumulate experience’. Base on this, the agent can change its structure and behavior. Such processes, e.g., in the flock of birds system which is the basis of PSO, include the production of new generations (have young birds), the emergence of differentiation and diversity (the flock of birds is divided into many small groups), and finding new themes (discover new food). Particle swarm optimization comes from the research on a CAS system - a social system of birds. The computational algorithm was first introduced by Kennedy and Eberhart in (1995). The framework of basic PSO can be described in a very simple form, see Figure 2.

After PSO’s emergence, it has evolved greatly. Many research groups focus on this algorithm due to its robustness and simple applicability without the need for cumbersome derivative calculations. Basically, there are four classes of PSO which focus on the algorithm itself, i.e.,
1. variants of the standard particle swarm optimization algorithm,
2. hybrid particle swarm optimization algorithms,
3. binary particle swarm optimization algorithms, and
4. cooperative particle swarm optimization algorithms.

Besides the algorithm itself, there are many research works for engineering applications. This chapter also mainly describes researches of using PSO in engineering problems.

Since 1995, PSO has been modified and improved into many variants. Here we list some main developments of PSO, see Table 1.

<table>
<thead>
<tr>
<th>year</th>
<th>contributors</th>
<th>main contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>Kennedy, Eberhart</td>
<td>provided the algorithm of PSO first</td>
</tr>
<tr>
<td>1997</td>
<td>Kennedy, Eberhart</td>
<td>modified to binary PSO algorithm which is used for structural optimization of neural networks, provided a helpful way to compare the performance of genetic algorithms</td>
</tr>
<tr>
<td>1998</td>
<td>Shi, Eberhart</td>
<td>improved the performance of convergence by putting inertia weight $w$ into the velocity item, dynamically adjust $w$ during the iterative process so as to counterpoise the global superiority and the rate of convergence (standard PSO)</td>
</tr>
<tr>
<td>1999</td>
<td>Clerc</td>
<td>introduced a contraction factor to ensure the convergence of PSO, relaxed the limitation of velocity</td>
</tr>
<tr>
<td>1999</td>
<td>Angeline</td>
<td>used the idea of evolution selection to improve the convergence</td>
</tr>
<tr>
<td>1999</td>
<td>Suganthan</td>
<td>introduced the idea of neighborhood operators into standard PSO so as to maintain the diversity of particles</td>
</tr>
<tr>
<td>2000</td>
<td>Carlisle, Dozier</td>
<td>proposed a PSO model which can be adapted to dynamic environment automatically</td>
</tr>
<tr>
<td>2001</td>
<td>Lovbjerg, Rasmussen, and Krink</td>
<td>introduced the concept of subpopulations from Genetic Algorithm into PSO</td>
</tr>
<tr>
<td>2001</td>
<td>van den Bergh</td>
<td>proved that the standard PSO algorithm can not guarantee that it will converge to the global optimum and even can not guarantee the convergence to a local optimum</td>
</tr>
</tbody>
</table>

Around the year 2000, researchers changed their methods and made further improvements. We can then find hundreds of articles in many application fields. Also, then researchers focused on how to use PSO for practical problems, such as power systems, IC design, system identification, state estimation, lot sizing, composing music, or modeling markets and organizations. Some classical applications are listed in Table 2.

Although the application of basic PSO for practical problems in engineering is often successful, most engineering problems contain problem immanent equality and inequality constraints. The basic PSO or so called standard PSO, can not be applied to constrained
Table 2: Some classical applications of PSO

<table>
<thead>
<tr>
<th>year</th>
<th>contributors</th>
<th>application field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>He, Wei, Yang, Gao, Yao, Eberhart, and Shi</td>
<td>fuzzy neural network</td>
</tr>
<tr>
<td>1999</td>
<td>Fukuyama, Takayama, Nakanishi, Yoshida, and Kawata</td>
<td>electric power systems</td>
</tr>
<tr>
<td>2000</td>
<td>Peng, Chen, and Eberhart</td>
<td>battery</td>
</tr>
<tr>
<td>2001</td>
<td>Cockshott, Hartman</td>
<td>biochemistry</td>
</tr>
<tr>
<td>2002</td>
<td>Tandon</td>
<td>CAD/CAM, CNC</td>
</tr>
<tr>
<td>2002</td>
<td>Blackwell, Bentley</td>
<td>music</td>
</tr>
<tr>
<td>2002</td>
<td>Coelho, Oliveira, and Cunha</td>
<td>greenhouse air temperature control</td>
</tr>
<tr>
<td>2002</td>
<td>El-Gallad, El-Hawary, Sallam, and Kalas</td>
<td>economics</td>
</tr>
<tr>
<td>2003</td>
<td>Emara, Ammar, Bahgat, and Dorrah</td>
<td>induction motors</td>
</tr>
<tr>
<td>2003</td>
<td>Lu, Fan, and Lo</td>
<td>environmental protection</td>
</tr>
<tr>
<td>2003</td>
<td>Srinivasan, Loo, and Cheu</td>
<td>traffic</td>
</tr>
<tr>
<td>2004</td>
<td>Onwubolu, Clerc</td>
<td>drilling</td>
</tr>
<tr>
<td>2004</td>
<td>Teo, Foo, Chien, Low, and You</td>
<td>placement of wavelength converters in an arbitrary mesh network</td>
</tr>
<tr>
<td>2006</td>
<td>Sedlaczek, Eberhard</td>
<td>robot stiffness optimization</td>
</tr>
<tr>
<td>2007</td>
<td>Yu, Zhu, Yu, and Li</td>
<td>bellow optimum design</td>
</tr>
<tr>
<td>2008</td>
<td>Han, Zhao, Xu, and Qian</td>
<td>multivariable PID controller design</td>
</tr>
<tr>
<td>2009</td>
<td>Holden, Freitas</td>
<td>hierarchical classification of protein function</td>
</tr>
</tbody>
</table>

problems. Thus, often constrained engineering problems are treated by inadequate penalty functions when using stochastic algorithms. This chapter will present an algorithm which utilizes the simple structure of the basic PSO technique and combines it with an augmented Lagrangian multiplier approach described in the following section.

2 Algorithm for constrained engineering problems

PSO is traditionally used for unconstrained optimization problems. However the general optimization problem is defined by the objective function $\Psi$, which is to be minimized with respect to the design variables $\mathbf{x}$ and the linear or nonlinear equality and inequality constraints. This can be formulated by

$$\minimize_{\mathbf{x} \in \mathbb{R}^n} \quad \Psi(\mathbf{x}) \quad \text{with} \quad \mathbb{X} = \{ \mathbf{x} \in \mathbb{R}^n \mid g(\mathbf{x}) = \mathbf{0}, \ h(\mathbf{x}) \leq \mathbf{0}, \ x_l \leq \mathbf{x} \leq x_u \}, \quad (1)$$

where $g(\mathbf{x}) = \mathbf{0}$ and $h(\mathbf{x}) \leq \mathbf{0}$ are the $m_e$ equality and $m_i$ inequality constraints, respectively. The lower limit is $x_l$ while $x_u$ is the upper limit.
2.1 General methods for the constrained optimization problem

To solve optimization problems with equality and inequality constraints, different methods and algorithms are used. Generally there are two kinds of methods, deterministic and stochastic ones. The deterministic algorithms usually use gradient information and often locate a minimum within a few iteration steps, but they sometimes get stuck in local minima and require a smooth performance function. Stochastic methods have the advantage that they may find a global minimum without posing severe restrictions on, e.g., the differentiability, convexity, or separability. Unfortunately the number of iterations is also increasing requiring many criteria evaluations.

In recent years, stochastic methods such as evolutionary algorithms, simulated annealing, and particle swarm optimization are applied frequently. However, basic PSO can’t handle constrained optimization problems. Hence, combining it with other strategies from deterministic optimization will greatly widen its applicability.

2.2 Extending the basic PSO to ALPSO for efficient constraint handling

The recursive update equations of basic PSO can be formulated by

$$\Delta x_i^{k+1} = \omega \Delta x_i^k + c_1 r_{1,i}^k (x_{i,\text{best},k}^k - x_i^k) + c_2 r_{2,i}^k (x_{\text{warm},k}^k - x_i^k).$$

(2)

This is the so called ‘velocity’ update equation and the position update is done in the traditional PSO algorithm by

$$x_i^{k+1} = x_i^k + \Delta x_i^{k+1}.$$  

(3)

Here $\omega$ is the inertia weight, usually $c_1, c_2 \in (0, 2)$ are referred to as cognitive scaling and social scaling factors, and $r_{1,i}^k \sim U(0,1), r_{2,i}^k \sim U(0,1)$ are two independent random functions. One saves $x_{i,\text{best},k}^k$ as the best position of particle $i$ reached till now, i.e., the individual best value, and $x_{\text{warm},k}^k$ is the best previously obtained position of all particles in the entire swarm, that is to say, the current global best position. In Eq. (3), the PSO technical term ‘velocity’ really corresponds to a time step $\Delta t$ times the mechanical velocity to give the equation a physically correct format, i.e., $\Delta x_i^{k+1} = \Delta t \dot{x}_i^{k+1}$, but usually in PSO literature the time step $\Delta t$ is omitted or set to be one. So the basic PSO algorithm is summarized for all $n$ particles by

$$\begin{bmatrix} x^{k+1} \\ \dot{x}^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \omega \dot{x}^k \end{bmatrix} + c_1 r_1^k (x_{i,\text{best},k}^k - x^k) + c_2 r_2^k (x_{\text{warm},k}^k - x^k),$$

(4)

where

$$r_1^k = \begin{bmatrix} r_{1,1} I_3 & 0 & \cdots & 0 \\ 0 & r_{2,1} I_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{n,1} I_3 \end{bmatrix}, \quad r_2^k = \begin{bmatrix} r_{1,2} I_3 & 0 & \cdots & 0 \\ 0 & r_{2,2} I_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{n,2} I_3 \end{bmatrix},$$

(5)
and \( n \) is the number of particles. Considering the spatial case, \( I_3 \) is a 3 \times 3 unit matrix. Similarly the matrices

\[
x_i^k = \begin{bmatrix} x_{1,i}^k \\ x_{2,i}^k \\ x_{3,i}^k \end{bmatrix}, \quad x_{\text{best},i}^k = \begin{bmatrix} x_{1,\text{best},i}^k \\ x_{2,\text{best},i}^k \\ x_{3,\text{best},i}^k \end{bmatrix}, \quad x_{\text{swarm}}^k = \begin{bmatrix} x_{1,\text{swarm}}^k \\ x_{2,\text{swarm}}^k \\ x_{3,\text{swarm}}^k \end{bmatrix}, \quad i = 1(1)n, \quad (6)
\]

are defined.

Engineering problems usually have constraints as in the two examples shown in the following sections, i.e., optimizing the stiffness of a hexapod machine and the guidance of swarm mobile robots.

For such constrained optimization problems, the augmented Lagrangian multiplier method can be used where each constraint violation is penalized separately by using a finite penalty factor \( r_{p,i} \). Thus, the minimization problem with constraints in Eq. (1) can be transformed into an unconstrained minimization problem

\[
\text{minimize} \quad L_A(x, \lambda, r_p)
\]

with \( L_A(x, \lambda, r_p) = \Psi(x) + \sum_{i=1}^{m_e+m_i} \lambda_i P_i(x) + \sum_{i=1}^{m_e+m_i} r_{p,i} P_i^2(x), \quad (9) \)

and \( P_i(x) = \begin{cases} g_i(x), & \text{for } i = 1(1)m_e, \\ \max \left(h_i - m_e(x), \frac{\lambda_i}{r_{p,i}} \right), & \text{for } i = (m_e+1)(1)(m_e+m_i). \\ \end{cases} \quad (10) \)

Here \( \lambda_i \) are Lagrangian multipliers and \( r_{p,i} \) are finite penalty factors.

Note that \( \lambda_i \) and \( r_{p,i} \) are unknown in advance and are adaptively adjusted during the simulation. According to Sedlacek and Eberhard (2006) this problem can be solved by dividing it into a sequence of smaller unconstrained subproblems with subsequent updates of \( \lambda_i \) and \( r_{p,i} \). Then, \( \lambda_i \) and \( r_{p,i} \) are changed based on the iteration equations

\[
\lambda_i^{s+1} = \lambda_i^s + 2r_{p,i}^s P_i(x), \quad i = 1(1)(m_e+m_i), \quad (11)
\]

\[
r_{p,i}^{s+1} = \begin{cases} 2r_{p,i}^s & \text{if } |g_i(x^s)| > |g_i(x^{s-1})| \land |g_i(x^s)| > \varepsilon_{\text{equality}}, \\ 0.5r_{p,i}^s & \text{if } |g_i(x^s)| < \varepsilon_{\text{equality}}, \\ r_{p,i}^s & \text{else}, \end{cases} \quad i = 1(1)m_e, \quad (12)
\]

and

\[
r_{p,j+m_e}^{s+1} = \begin{cases} 2r_{p,j+m_e}^s & \text{if } h_j(x^s) > h_j(x^{s-1}) \land h_j(x^s) > \varepsilon_{\text{inequality}}, \\ 0.5r_{p,j+m_e}^s & \text{if } h_j(x^s) < \varepsilon_{\text{inequality}}, \\ r_{p,j+m_e}^s & \text{else}, \end{cases} \quad j = 1(1)m_i, \quad (13)
\]
where $\varepsilon_{\text{equality}}$ and $\varepsilon_{\text{inequality}}$ are user-defined tolerances for constraint violations which are still acceptable. The initial values are $\lambda^0 = 0$ and $r^0_{p,i} = 1$. This work uses the update equations (11) which come from partial differentiation of the Lagrangian function (9) yielding the subproblem

$$
\left[ \frac{\partial L}{\partial x} \right]_{x=x^t} = \left[ \frac{\partial \Psi(x)}{\partial x} + \sum_{i=1}^{m_{r}+m_{f}} \lambda_i \frac{\partial P_i(x)}{\partial x} + \sum_{i=1}^{m_{r}+m_{f}} 2r_{p,i} \frac{\partial P_i(x)}{\partial x} \right]_{x=x^t} \approx 0. \quad (14)
$$

We apply a heuristic update scheme for the penalty factors $r_{p,i}$, see Eqs. (12) and (13). If the intermediate solution $x^i$ is not closer to the feasible region defined by the $i$-th constraint than the previous solution $x^{i-1}$, the penalty factor $r_{p,i}$ is increased. On the other hand, we reduce $r_{p,i}$ if the $i$-th constraint is satisfied with respect to user-defined tolerances.

During the algorithm tests we also experienced that for active constraints a lower bound on the penalty factors yields improved convergence characteristics for the Lagrangian multiplier estimates. This work maintains the magnitude of the penalty factors such that an effective change in Lagrangian multipliers is possible. This lower bound is

$$
r_{p,i} \geq \frac{1}{2} \sqrt{\frac{|\lambda_i|}{\varepsilon_{\text{equality}}, \varepsilon_{\text{inequality}}}}. \quad (15)
$$

So far, the basic PSO algorithm is extended to augmented Lagrangian particle swarm optimization (ALPSO) which is well suited for handling constrained problems. In the following sections 3 and 4, two application examples will be shown which are based on ALPSO with several extensions.

### 3 PSO based algorithm used for mechanism design

#### 3.1 Mechanism design and optimization

Optimization is a very important aspect in mechanism design and production especially for some complex mechanical systems. As a relatively new subject optimal design is the result of applying optimization techniques and computing technologies in the product design area. The basic idea is to choose the design parameters in a systematic way and to try to achieve the most satisfactory particular mechanical property under the condition of meeting requirements from various design constraints.

Following, an example of optimizing the stiffness behavior for a hexapod robot based on ALPSO is presented. Machines with parallel kinematics feature low-inertia forces due to low moving masses of the structure. In contradiction to this are the desired high accuracy and stiffness required in various fields, e.g., like robotics or measurement systems. We investigated and optimized the stiffness behavior of the hexapod robot HEXACT using the ALPSO method. HEXACT is a research machine tool with parallel kinematics, developed by Prof. Heisel and his coworkers at the Institute of Machine Tools, University of Stuttgart, Germany (Heisel et al. 1998), see Figure 3 left.

A frequent drawback of parallel kinematics robots is the appearance of kinematically singular configurations that have to be avoided during operation. For the design of the investigated hexapod machine, such singularities are located along the tool axis in the central...
3.2 Optimization design of the HEXACT

For parallel kinematics machines, the stiffness of the end-effector depends on its translational and rotational position in the workspace, the so-called pose. The relation between the applied forces and torques $F$ on the one hand and the evasive displacements $\Delta y_e = [\Delta r_e \ \Delta \phi_e]$ of the end-effector on the other hand is highly nonlinear. For the evaluation of the stiffness and flexibility behavior, it is sufficient to regard the linearized relations

$$dF = K_t \cdot dy_e \quad \text{and} \quad dy_e = C_t \cdot dF,$$

where the tangential stiffness matrix $K_t$ describes the change in forces caused by a given rotational and translational displacement change and the tangential flexibility (or compliance) matrix $C_t$ gives the relationship between a given force and torque load change and the resulting end-effector’s displacement change.

In robotics, the elastic behavior is usually described in a global coordinate system using the vector of infinitesimal rotation $d\mathbf{s}_e = \omega_e dt$ to describe the deflection of orientation of the end-effector. A simplified description of the compliance behavior can be achieved by assuming all driven joints behavior according to linear elasticity, which is sufficient to show the basic behavior of the machine. In our case, the stiffness of the inner and outer joints of each of the six driving struts can be summarized to a resulting strut elasticity $k$. The relation between the infinitesimal applied load $dF$ and the displacement of the end-effector $dq_e = [dr_e \ ds_e]$ is then given as

$$dq_e = \frac{1}{k} J \cdot J^T \cdot dF$$

$$=: C_{xx}$$
with the Jacobian matrix \( J \) also describing the relation between the actuator velocities \( \dot{\theta} = [\dot{\theta}_1 \cdots \dot{\theta}_6] \), and the end-effector velocity

\[
q_e = J(y) \cdot \dot{\theta},
\]

(18)

see Henninger et al. 2004. It should be mentioned that different definitions of the Jacobian matrix exist in literature and one has to define clearly which one is used. To describe the deflection of orientation by the change of elementary rotation angles like Cardan angles which are applied here with \( d\phi_e = [d\alpha_e \ d\beta_e \ d\gamma_e] \), we can write

\[
dy_e = \begin{bmatrix} I & 0 \\ 0 & H_R \end{bmatrix} \cdot \begin{bmatrix} dr_e \\ ds_e \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & H_R^{-1} \end{bmatrix} \cdot C_{tx} \cdot dF
\]

(19)

with the identity matrix \( I \) and the Jacobian matrix of rotation \( H_R \), see, e.g., Schiehlen and Eberhard 2004. The inverse relation is then given by

\[
dF = K_t \cdot dy_e
\]

(20)

with the tangential stiffness matrix

\[
K_t = -k J^{-T} J^{-1} \cdot \begin{bmatrix} I & 0 \\ 0 & H_R^{-1} \end{bmatrix}.
\]

(21)

In the general case, both matrices \( K_t \) and \( C_t \) are symmetric and fully occupied. This means that a load in one direction causes rotational and translational evasive motions in every direction. Furthermore, they consist of four blocks having different units. For the following considerations, it is useful to normalize the coefficients using characteristic forces, torques, lengths, and angles. These values should be chosen with care as they influence the results of the following optimization process. For our example, we have chosen the normalization as described by Henninger et al. (2004), where more detailed information about the stiffness analysis of the HEXACT machine tool including information about its parameter values can be found.

One way to reduce the flexibility and to eliminate the singular configurations is, to alter the angles \( \zeta \) between the telescope struts and the end-effector, which are perpendicular in the initial design, see Figure 3. In addition, the diameter \( d_e \) and the length \( l_e \) of the end-effector and thus the mounting points of the struts can be varied to improve the stiffness behavior of the hexapod robot, yielding the optimization parameter vector

\[
x = [\zeta \ d_e \ l_e].
\]

(22)

To gain more insight into the global flexibility behavior of the machine, it is necessary to evaluate the stiffness matrix at several poses in the workspace. In this study, \( N = 6^5 \) sample poses on a regular grid are regarded. As a global flexibility criterion is to be minimized, we use the negative average of the minimum principal stiffness of the \( N \) sample poses in the entire workspace

\[
\min_{\mathbf{x}} f_k(\mathbf{x}) \quad \text{with} \quad f_k(\mathbf{x}) = -\frac{1}{N} \sum_{j=1}^{N} \min(k_j^x),
\]

(23)
where \( \min(k^*_j) \) is the minimum eigenvalue of the tangential stiffness matrix \( K_j \) at pose \( j \).

The optimization problem described by Eq. (23) is solved using particle swarm optimization which enables a gradient free and global search without any difficult or expensive gradient calculations or without any restrictions to a local solution. Two different sets of design variables were considered. The first variant considers only \( \zeta \) as a design variable, whereas variant 2 takes into account both \( \zeta \) and the dimension of the end-effector described by \( d_e \) and \( l_e \).

Regarding design variant 1, the average of the minimum principal stiffness could be improved by a factor of 35 with respect to the initial design, see Henninger et al. (2004). The optimization of design variant 2 improved the average stiffness by a factor of approximately 200. As expected, the geometry of the end-effector has a great influence on the stiffness behavior of the hexapod robot. However, the maximization of the minimum stiffness as formulated by Eq. (23) impairs the stiffness distribution of the hexapod machine in the workspace. The standard deviation of the minimum principal stiffness in the workspace increased by a factor of 11 for design variant 1 and by a factor of 82 for design variant 2. The resulting more non-uniformly distributed stiffness behavior is undesirable for manufacturing processes. Therefore, design variant 3 is defined by solving the following modified nonlinearly constrained optimization problem

\[
\begin{align*}
\text{minimize} \quad & f_s(x) \quad \text{with} \quad f_s(x) = \sqrt{\frac{1}{N-1} \sum_{q=1}^{N} (\min(k^*_j) + f_k(x))^2}, \\
\text{subject to} \quad & h(x) = f_k(x) - 0.8 f^*_k \leq 0,
\end{align*}
\]

(24)

where \( f_s(x) \) describes the standard deviation of the stiffness distribution. The inequality constraint restricts the decrease in the optimized average principal stiffness \( f^*_k \) from design variant 2. This nonlinear constrained problem is solved using the augmented Lagrangian particle swarm optimization algorithm with respect to the design variables \( \zeta, d_e, \) and \( l_e \). For this optimization process as well as for design variants 1 and 2, we used \( n = 20 \) particles. As a result, we could improve the standard deviation of the minimum principal stiffness by 25% with an acceptable worsening in the average stiffness of 20% compared to variant 2. The experimental setup and results are summarized in Table 3 and Table 4, respectively. For this engineering problem, these results are useful to analyze the design modifications and to quantify the potentially achievable improvements. Further information about the hexapod robot and its stiffness behavior can be found in (Henninger et al., 2004) and (Henninger, 2009).

4 PSO based algorithm used for guidance of swarm mobile robots

4.1 Background of robot navigation

Robot motion planning or so called path planning is a very important topic in robot development especially for mobile robots, robotic arms or walking robots. This research area
belongs to robot navigation which includes location, path planning, and obstacle avoidance. These are three different topics. In fact, in engineering applications, they are often considered at the same time and are summarized in the term ‘motion planning’. In Table 5 some methods/strategies used in robot navigation are mentioned.

### Table 3: Setup for optimizing the stiffness behavior of the hexapod robot

<table>
<thead>
<tr>
<th>design variant</th>
<th>design variables</th>
<th>objective function</th>
<th>constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>variant 1</td>
<td>[ x = [\zeta] ]</td>
<td>( f_k )</td>
<td>-</td>
</tr>
<tr>
<td>variant 2</td>
<td>[ x = [\zeta \ d \ e \ l] ]</td>
<td>( f_k )</td>
<td>-</td>
</tr>
<tr>
<td>variant 3</td>
<td>[ x = [\zeta \ d \ e \ l] ]</td>
<td>( f_s )</td>
<td>( f_k - 0.8f_k(x^*_2) \leq 0 )</td>
</tr>
</tbody>
</table>

### Table 4: Results of optimizing the stiffness behavior of the hexapod robot

<table>
<thead>
<tr>
<th>design variant</th>
<th>solutions</th>
<th>objective function value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial design</td>
<td>( x_0 = [0^\circ \ 0.4m \ 0.44m] )</td>
<td>( f_k(x_0) = -0.025 ), ( f_s(x_0) = 0.039 )</td>
</tr>
<tr>
<td>variant 1</td>
<td>( x^*_1 = [76.72^\circ] )</td>
<td>( f_k(x^<em>_1) = -0.876 ), ( f_s(x^</em>_1) = 0.435 )</td>
</tr>
<tr>
<td>variant 2</td>
<td>( x^*_2 = [65.77^\circ \ 0.6m \ 1.0m] )</td>
<td>( f_k(x^<em>_2) = -5.083 ), ( f_s(x^</em>_2) = 3.167 )</td>
</tr>
<tr>
<td>variant 3</td>
<td>( x^*_3 = [79.57^\circ \ 0.509m \ 1.0m] )</td>
<td>( f_k(x^<em>_3) = -4.066 ), ( f_s(x^</em>_3) = 2.349 )</td>
</tr>
</tbody>
</table>

### Table 5: Robot navigation strategies

<table>
<thead>
<tr>
<th>robot navigation (motion planning)</th>
<th>location</th>
<th>relative</th>
<th>absolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>path planning</td>
<td>local</td>
<td>APF, genetic algorithms, fuzzy logic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>global</td>
<td>environment modeling</td>
<td>graph methods, free-space methods, grid methods</td>
</tr>
<tr>
<td></td>
<td>path search</td>
<td>A* algorithms, D* optimal algorithms</td>
<td></td>
</tr>
<tr>
<td>obstacle avoidance</td>
<td>VFH, APF, VFH+, VFH*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local path planning strategies include APF (Artificial Potential Field) first proposed by Khatib in 1968, genetic algorithms and fuzzy logic algorithms. Meanwhile global environment modeling can be realized by the methods of graph, free-space, grid and global path search usually achieved by the optimal algorithms of A* (Hart, Nilsson and Raphael, 1968) or achieved by D* (Stentz, 1994). The third part of robot navigation is obstacle avoidance which has many solutions like VFH (Vector Field Histogram) proposed by Borenstein and Koren in 1991, the updated versions VFH+ (Ulrich and Borenstein, 1998) and VFH* (Ulrich and Borenstein, 2000).

Actually, each of the algorithms has its limitations, e.g., in some situations it can work reliably, but with conditions changing, the algorithm may lose its effectiveness. Thus, many researchers enhance old algorithms or develop new algorithms with innovations. In spite of this, for swarm robots there still is no solution which can be used in general and works robustly. So, besides the above mentioned conventional methods, some researchers
dedicated their attention to biology inspired algorithms. Because nature can motivate unusual approaches, it inspired countless scientific innovations, and helps humans to solve practical problems. Some of the most innovative and useful discoveries have arisen from a fusion of two or more seemingly unrelated fields of study just like some algorithms invented from models of biology. In the robotics area, there are several biology inspired algorithms for robot motion planning which are briefly presented in the following.

Genetic algorithms (GA) are famous biology inspired algorithms and are used in many fields. However, in the view of swarm mobile robots they look not so suitable. The main reason is that GAs show sudden large changes during the iterations. Such jumps are not feasible for real robots.

Bacterial colony growth algorithms (BCGA) are new biology inspired approaches which were proposed in 2008 by Gasparri and Prosperi. They use the idea of bacterium growth and bunching to the nutrient areas to be a colony. Each robot is seen as a bacterium in a biological environment which reproduces asexually. Unfortunately, these algorithms are very complex. If used in swarm robots, the calculation cost can be prohibitively high.

Recently, a reactive immune network (RIN) was proposed and employed for mobile robot navigation (Guan and Wei, 2008). This is an immunological approach to mobile robot reactive navigation. The same shortcoming as in the aforementioned BCGA is its complexity. Additionally, it is difficult to be scaled for swarm robots.

The heuristic ant algorithms (AA) use a group of modelled ants to navigate the multi-robot, but the definitions are not simple, and the iterative process is sophisticated (Zhu, 2006).

In contrast to the above mentioned biology inspired algorithms, the PSO algorithm is more appealing due to its clear ideas, simple iteration equations, and also the ease to be mapped onto robots or even swarm robots.

In the following section, the mechanical ‘PSO’ motion model of swarm mobile robots is presented.

### 4.2 Mechanical PSO model of swarm mobile robots

Usually the PSO algorithm is used as a mathematical optimization tool without physical meaning. In the following example, a PSO based algorithm is presented which is used for the motion planning of swarm mobile robots and so their physical background must be considered. Additionally, we want to interpret the PSO algorithm as providing the required forces in the view of multibody system dynamics. Each particle (robot) is considered as one body in a multibody system which is influenced by forces and torques from other bodies in the system but without direct mechanical connection to them. The forces are artificially created by corresponding drive controllers.

First, one starts from the Newton-Euler equations. These equations are presented in a general form. The motion of the particle $i$ in the ‘multi-particle’ system is governed by the two matrix equations

$$m_i \ddot{a}_i = f_i,$$  \hspace{1cm} (25)
$$J_i \ddot{\omega}_i + \omega_i \times J_i \omega_i = l_i, \hspace{1cm} i = 1(1)n$$  \hspace{1cm} (26)

which correspond to the balance of linear and angular momentum. Here $m_i$ is the mass of particle $i$, $a_i$ its linear acceleration and $f_i$ are the forces acting on particle $i$. In Eq.
the matrix $J_i$ is the moment of inertia, $\alpha_i$ the angular acceleration, while $\omega_i$ is the angular velocity. At the right side of Eq. (26), $l_i$ contains the moments or torques acting on body $i$. Equation (25), the so called Newton equation, is consisting of three scalar equations that relate the forces and the accelerations of the particle in the three Cartesian dimensions. Equation (26), on the other hand, relates the rotational acceleration to a given set of moments or torques. This matrix equation consists also of three scalar equations and is called Euler equation. Further more, Eqs. (25) and (26) can be combined in the Newton-Euler equations for one particle $i$

$$\begin{bmatrix} m_i I_3 & 0 \\ 0 & J_i \end{bmatrix} \begin{bmatrix} a_i \\ \alpha_i \end{bmatrix} = \begin{bmatrix} f_i \\ l_i - \omega_i \times J_i \omega_i \end{bmatrix}. \quad (27)$$

If no constraints or joints exist, this equation contains no reaction forces. Otherwise, a projection using a global Jacobian matrix can be performed eliminating them. In both cases, the general form of the equations of motion for swarm robots can be formulated as

$$M \ddot{x} + k = q \quad \text{or} \quad \ddot{x} = M^{-1}(q - k) = M^{-1}F. \quad (28)$$

For a free system without joints, $M = \text{diag}(m_1 I_3, m_2 I_3, \cdots, m_n I_3, J_1, J_2, \cdots, J_n) = M^T \geq 0$ is the mass matrix collecting the masses and inertias of the particles (robots), $\ddot{x}$ is the general acceleration

$$\ddot{x} = \begin{bmatrix} a \\ \alpha \end{bmatrix}, \quad (29)$$

$k$ is a term which comes from the Euler equation, and $q$ contains forces and moments acting on the robots. One can also write Eq. (28) as a state equation with the state vector

$$y = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \quad (30)$$

where $x$ and $\dot{x}$ are the translational and rotational positions and velocities of the robots. Thus, the first order differential equation follows

$$\dot{y} = \begin{bmatrix} \dot{x} \\ \dot{\dot{x}} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ M^{-1}F \end{bmatrix}. \quad (31)$$

Together with the initial conditions, the motion of the swarm robots over time can be computed, e.g., by the simple Euler forward integration formula

$$y^{k+1} = y^k + \Delta t \dot{y}^k, \quad (32)$$

where $\Delta t$ is the chosen time step, and the superscript $k, k + 1$ means the $k$-th and $(k + 1)$-th point in time. Rewriting Eq. (32) yields

$$\begin{bmatrix} x^{k+1} \\ \dot{x}^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \dot{x}^k \end{bmatrix} + \Delta t \begin{bmatrix} \dot{x}^k \\ M^{-1}F^k \end{bmatrix}. \quad (33)$$

Next, the connection between the mechanical motion of a particle or robot and the PSO algorithm should be made. For this reason we first assume that the robot should at the
moment only be driven by forces such that no torques appear, i.e. \( I_i = 0 \). Also, the motion is described for a mean-axis system in the center of gravity so that the second term in Eq. (26) vanishes, too. So, where do the forces \( f_i \) come from? The force \( f_i \) is determined by three parts, \( f_{1i}^k, f_{2i}^k \) and \( f_{3i}^k \), which are defined as

\[
\begin{align*}
  f_{1i}^k &= -h_{f1}^k (x^k - x_{self}^{best,k}), \\
  f_{2i}^k &= -h_{f2}^k (x^k - \hat{x}_{swarm}^{best,k}), \\
  f_{3i}^k &= -h_{f3}^k x^k
\end{align*}
\]

with the matrices

\[
\begin{align*}
  h_{f1}^k &= \text{diag}(h_{1f1}^k I_3, h_{2f1}^k I_3, \ldots, h_{n_f1}^k I_3, 0_{3n}), \\
  h_{f2}^k &= \text{diag}(h_{1f2}^k I_3, h_{2f2}^k I_3, \ldots, h_{n_f2}^k I_3, 0_{3n}), \\
  h_{f3}^k &= \text{diag}(h_{1f3}^k I_3, h_{2f3}^k I_3, \ldots, h_{n_f3}^k I_3, 0_{3n}).
\end{align*}
\]

Here \( I_3 \) is a \( 3 \times 3 \) unit matrix, \( 0_{3n} \) is a \( 3n \times 3n \) zero matrix. The forces \( f_{1i}^k \) and \( f_{2i}^k \) are attraction forces to the last self best robot position and the last swarm best robot position. They are proportional to their distances. The vector \( f_{3i}^k \) represents the force which is proportional to the last velocity and is a kind of inertia which counteracts a rapid change in direction.

This work maps the swarm mobile robots’ ‘PSO’ model to the original PSO algorithm so as to find the theoretical support for motion planning of swarm mobile robots. Inserting Eq. (34) to Eq. (33) yields

\[
\begin{bmatrix}
  x^{k+1} \\
  \dot{x}^{k+1}
\end{bmatrix} = \begin{bmatrix}
  x^k \\
  \dot{x}^k
\end{bmatrix} + \Delta t \begin{bmatrix}
  I_{6n} - \Delta t M^{-1} h_{f3}^k \\
  M^{-1} h_{f1}^k (x_{self}^{best,k} - x^k) + M^{-1} h_{f2}^k (\hat{x}_{swarm}^{best,k} - x^k)
\end{bmatrix}.
\]

Comparing the mechanical ‘PSO’ model of swarm mobile robots in Eq. (36) to the original PSO model in Eq. (4), one can see that they are quite similar and the corresponding relationships are

\[
\begin{align*}
  \Delta t M^{-1} h_{f1}^k &\leftrightarrow c_1 r_1^k, \\
  \Delta t M^{-1} h_{f2}^k &\leftrightarrow c_2 r_2^k, \\
  I_{6n} - \Delta t M^{-1} h_{f3}^k &\leftrightarrow \omega.
\end{align*}
\]

Of course, similar relations can be derived if also torques are entered and rotations of the robots occur since neither the system description (31) nor the integration formula (32) change. The random effects are included in \( h \) and all forces must be created by local drive controllers in the robots.
4.3 Modification of the neighborhood

From Eqs. (4) and (36), one can see that for using a PSO based algorithm for swarm mobile robots, each robot needs the information of the current self best value and the current swarm (global) best value. It is trivial to get the self best value since it can be stored in the local on-board position memory. When the size of the swarm becomes larger, it’s sometimes infeasible or even undesired to distribute and store the swarm best value to all swarm members. Therefore, it is proposed to replace $\hat{x}_{k}^{\text{swarm}}$ with $\hat{x}_{k}^{\text{neigho}}$, that is, using the best value in the neighborhood to replace the global one during the iterative processes.

This replacement has great practical significance since it adds a lot of robustness to the robots, especially for the motion planning. It reduces the communication expense of the algorithm since they only need to communicate with other robots which are close to them. So the original PSO in Eq. (4) and the mechanical ‘PSO’ for swarm mobile robots in Eq. (36) are modified by replacing the global best to the best in the neighborhood.

The strategy to define the neighborhood depends on the size of the swarm and the communication ability of practical robots. Basically, there are two kinds of neighborhood models, i.e., indexed neighborhoods and spatial neighborhoods. In this work, the latter are used. The radius of the neighborhood in this work is flexible as the size and density of swarm are changing. Here, the algorithm selects the nearest one-third particles from the complete swarm to the current particle as the neighborhood field. Then, the neighborhood best particle/robot will be determined after comparing their performance values.

4.4 Extension of the basic PSO algorithm to VL-ALPSO for coordinated movements of swarm mobile robots

It is proposed to use a PSO based algorithm for the motion planning of swarm mobile robots with the goal to search a target in the environment. Usually, the target in the environment can be described by an objective function. Several kinds of information are available to a robot, i.e.,

1. local information like the evaluation of a function value or a gradient at the current position of a robot,
2. information about the surrounding, e.g., obtained from distance sensors, and
3. information communicated by other robots.

The first kind of information is treated in the optimization problem and its usage is basically described in Section 2. The second kind of information will be described later in Subsection 4.4.1 and is only considered by the robot itself. Finally the third kind of information described in Subsection 4.4.2 is the core of the VL-ALPSO algorithm where several particles/robots search in parallel and exchange information.

4.4.1 Swarm particle robots

If PSO should be applied to a real mobile robot, the algorithm must be modified and adjusted, e.g., to include the mechanically feasible motion and to take care for obstacle avoidance. In classical PSO or ALPSO, the particles can move in any direction and with any
velocity. Nevertheless, the robots have to move with limited velocity and acceleration since only finite forces can be generated by the drives yielding only finite accelerations and so any change in velocity needs time and energy. Also, there are obstacles in the environment.

In the process of adapting the algorithm, there are some aspects that need to be considered.

1. The basic structure of ALPSO should be used for guiding the motion including its ability to deal with constraints.
2. The algorithm should not yield too much computational effort since no strong processor is available on a robot.
3. The robots should take care locally for obstacle avoidance.
4. The algorithm should be applied for physical real robots.

On the basis of the before-mentioned requirements, this study proposes the structure of a sub-algorithm for obstacle avoidance, especially for static obstacles in an environment. Of course, obstacles from an internal map could be considered simply as constraints which are treated in the optimization code. However, in robotics it is desirable that the collision detection and avoidance are done locally in each robot based on its obtained own sensor signals. This is also important, e.g., for handling moving obstacles. Constraints treated in the optimization algorithm are, e.g., considering energy, minimal distances between the robots or avoiding prohibited areas.

In the algorithm, we restrict the maximal displacement of a particle in a single step to \( Z \) which is defined as

\[
Z = \max(z_1, z_2).
\]  

(38)

Here \( z_1 \) and \( z_2 \) come from two different aspects. The distance \( z_1 \) is due to the computer performance. In the scenario of a real mobile robot, it depends on the embedded Micro-CPU and the volume of robots. The value \( z_2 \) depends on the braking distance. This braking distance must guarantee that the robot won't collide with obstacles before it can stop or turn away. Because the stopping force or steering force is finite and strongly depends on the maximal velocity, this maximal braking distance must be chosen.

In order to make sure that particles (robots) in the simulation do not pass through (fly over or jump over) obstacles under any circumstances, even for a very small or very thin obstacle, one can set dangerous regions surrounding the static obstacles in the environment. If any particle (robot) enters into such a dangerous region, the algorithm will give alarm and focus on collision avoidance instead of concentrating on optimization. In fact, this principle of obstacle avoidance is achieved in the actual mobile robot because the robot can make the collision avoidance based on the data from distance sensors. In Figure 4, a length of \( Z + T_{ol} \) aside each edge of the obstacle is used to define the dangerous regions.

In the situation shown in Figure 4, during the iterative process of ALPSO for updating the velocity and position it is identified that the next step will enter a dangerous region. Of course the step will not go so far as to collide with the obstacle, because each single iteration step for the particle or the robot is controlled with a maximal velocity. The maximal step is related to \( Z \) and the width of the dangerous region is at least \( Z + T_{ol} \). If the dangerous region is entered, the algorithm takes care for obstacle avoidance first. Here \( T_{ol} \) is a small
positive tolerance value. The flow chart is shown in Figure 5.

![Flow chart of the first sub-algorithm of obstacle avoidance](image)

Figure 5: Flow chart of the first sub-algorithm of obstacle avoidance

If the robot is in danger to collide with an obstacle, this study assumes an additional force to Eq. (36) with the effect of generating an acceleration and then the velocity is changed which yields a steering rotation angle $\theta_i$ or braking of the robot. The sub-algorithm makes a decision for the direction of steering. Due to comparison of $\theta_i > 0$ or $\theta_i < 0$, it controls the rotation direction to left or right, respectively. If the robot is not in danger to enter into the dangerous regions, then the movement to the new position guided by ALPSO will be performed.

However, we need a little further consideration. How to determine the angle and the
steering direction when the robot meets the obstacle? Based on the concept of minimal energy to be used in the system, and also since we assume at first that the obstacles in the environment are static and known locally from the sensor signals, the above problem can be treated by boundary scanning, see Figure 6.

![Figure 6: Boundary scanning](image)

The robot looks for a point \( O' \) which is the closest point on the obstacle. The extremal scanned points on the obstacle are A and B. This yields the the angles \( \alpha \) and \( \beta \), respectively.

If \( \alpha > \beta \), then \( \theta_i \) should be set to rotate in the right direction. If \( \alpha < \beta \), the robot should turn left. In this way, the robot can save energy bypassing the obstacle since a big rotation angle will require a lot of steering and cost much energy for the robot. However, scanning the obstacles for the practical robots will take some time and not every robot has the ability of scanning. Usually, it needs cameras and sophisticated image processing algorithms.

### 4.4.2 Volume constrained swarm robots

So far, the question of obstacle avoidance for non volume restricted swarm particle robot is treated. Several references use PSO based algorithms to navigate a group of mobile robots but only very few of them take into account the size of the robot itself when they map the approach from particles to real robots. Here one must consider motion planning for swarm mobile robots, where the volume of the robots itself is not ignored. For this, one needs to work on the following issues.

1. Collision avoidance between the volume robot and static obstacles,
2. Mutual avoidance during the motion of all volume robots.

In the following descriptions the word ‘robot’ means a volume robot, and a ‘particle robot’ is one with no volume. We enclose each robot in its enclosing circle with radius \( R_i \). Hence, the first question becomes trivial because simply the value of \( T_{ol} \) has to be adjusted considering the volume. This strategy enables the robot to bypass all obstacles including thin barriers.

To the second question of mutual avoidance during the movement of all volume robots, the simulation in this work uses ‘virtual detectors’ on each robot. Take robot A as an example, see Figure 7, around its forward velocity \( v_A \), the virtual detector has, e.g., a 120 degrees field of view. This view area can be taken as a dynamic detection area, and can also be referred to as a dynamic dangerous region. Of course, the angle can be adjusted to a suitable value based on the available sensors.

The radius of the detector area is set slightly larger than \( 2Z \) since the situation of two
robots moving in opposite directions at full speed must be considered. So actually the detector radius is set to be $2(Z + T_{d})$. In this way, during each step of trial calculation in the iterative process, if robot A detects that its next step will lead to other peers entering its dynamic dangerous region, an immediate breaking will be started by this robot. Then it turns to the obstacle avoidance sub-algorithm. However, the question is not so simple since the rotations can fall into an endless loop which is of course not reasonable. Such a case is depicted in Figure 8 where the considered robot W is surrounded by its peers.

![Figure 7: Virtual detector](image)

![Figure 8: Robot surrounded by peers](image)

In this situation the sub-algorithm stops this robot and, at the same time, increases the scope of its virtual detector to 360 degrees. Robot W in Figure 8 waits until the other robots move away, and then it takes the next step guided by ALPSO. Such a robot surrounded by other robots occurs frequently especially at the end of the search since all the individual robots then concentrate in a small area near the target. The flow chart of this sub-algorithm is shown in Figure 9 and the flow chart of VL-ALPSO can be seen in Figure 10.

### 4.5 Simulation and results

Some assumptions for the simulation experiments must be made:

1. The static obstacles are represented by polygons,
2. all simulations treat only the planar case, and
3. during the avoidance of obstacles, the robots can rotate full 360 degrees.

#### 4.5.1 Objective function and constraints

As an objective function, a group of robots should search a target in a certain area. The target could be a light source, odor source, etc., with its potential described, e.g., as

$$f(x) = \frac{1}{(x_1 - x_{m1})^2 + (x_2 - x_{m2})^2 + \epsilon}. \quad (39)$$

In Eq. (39), $x_{m1}, x_{m2}$ is the location of the maximum, $\epsilon$ is a very small positive value used only for avoidance of an infinite value of $f$ if $x = x_m$. Robot $i$ has the information of its position $x_i = (x_{1,i}, x_{2,i})$ and velocity $\dot{x}_i = (\dot{x}_{1,i}, \dot{x}_{2,i})$. The task of the group robots is to search for the source in the environment, and of course they do not know $x_m$ in advance.
Figure 9: Flow chart of second sub-algorithm of obstacle avoidance

The robots themselves are equipped with sensors to measure the local strength of the source $f(x)$, and they can exchange information to their surrounding neighbors. Interferences only exist in certain local districts, they will reduce the individual robot’s ability of exchanging information, but they can not go so far as to undermine this robot since it always has its local information.

If the center of the source $x_m$ is infeasible due to constraints or obstacles, then the robots should at least get as close as possible. For the following simulations we maximize the performance function from Eq. (39) and add three inequality constraints $h_1$, $h_2$ and $h_3$. Later also some obstacles will be added. The problem can then be formulated by

$$
\text{minimize} \quad \Psi(x) \quad \text{(40)}
$$

with \( \Psi(x) = \frac{1}{f(x)} - \varepsilon = (x_1 - x_{m1})^2 + (x_2 - x_{m2})^2 \)

subject to
\[
\begin{align*}
    h_1(x) &= 3 - x_1 \leq 0, \\
    h_2(x) &= 2 - x_2 \leq 0, \\
    h_3(x) &= 1 + x_1^2 - x_2^2 \leq 0,
\end{align*}
\]

Of course this is quite trivial as an optimization problem but it allows us to test the described algorithms. Here, we choose $x_m = (3, \sqrt{10})$ and the constrained minimum is $x_{opt} = x_m$ with $\Psi(x_{opt}) = 0$. 

4.5.2 Experimental setup

The main purpose of first using simulation instead of a real robotic system is convenience. A simulation environment enables the systems to be developed rapidly and transferred to the real system with hopefully minimal change in behavior. In addition, simulation offers a programmer access to data which is not easily attainable with real robots.

This study classifies the simulation experiments as shown in Table 6. In the experiments, the basic PSO parameters are set to $c_1 = 0.1$, $c_2 = 0.8$ and $\omega = 0.6$ which are equivalent to the force factors acting on $h_{f1}$, $h_{f2}$ and $h_{f3}$ in mechanical PSO. The population of the swarm mobile robots is $n = 4$ or 30. The symbol dot ' . ' is used for a particle robot, and a circle ' . ' for a volume robot. Bounds are $x_l = [-10 - 10]$ and $x_u = [10 10]$ in this planar case.

All algorithms are programmed using MATLAB. This work utilizes the idea of modular programming. The main function calls different sub-modules (sub-functions) depending on different situations. The benefits by doing so are not only to be able to focus on a module
Table 6: Type of experiment

<table>
<thead>
<tr>
<th>no.</th>
<th>particles</th>
<th>experiment description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n=4</td>
<td>particle robots coordinated movement under constraints without obstacles based on ALPSO</td>
</tr>
<tr>
<td>2</td>
<td>n=4</td>
<td>as experiment one, but with static obstacles and also using the first sub-algorithm of obstacle avoidance</td>
</tr>
<tr>
<td>3</td>
<td>n=30</td>
<td>volume robots under constraints with static obstacles, considering the mutual collision of robots, the target locates outside of obstacle, based on VL-ALPSO, first and second sub-algorithms of obstacle avoidance, plus virtual detector</td>
</tr>
<tr>
<td>4</td>
<td>n=30</td>
<td>as experiment three, but the target is within an obstacle</td>
</tr>
</tbody>
</table>

to solve a specified problem and to improve the overall efficiency of the codes, but also to prepare the adaptation of the MATLAB code onto other platforms, especially to the robot itself.

At the stages of exploration and final application, in order to get a balance between convergence and accuracy, it is required to modify the basic parameters of VL-ALPSO frequently. So the development of a convenient user interface becomes necessary. Based on the MATLAB GUI functions, a user interface is developed which is shown in Figure 11. All such interfaces allow testing the simulation conveniently. Furthermore, the modules are encapsulated behind the user interface.

4.5.3 Results and discussions

For experiment one and two, we want to show the aggregate behavior of the particles and their trajectories. In order to make their motion easier to be visualized, only 4 particles are used there. The objective function and constraints used are described in Eq. (40). The experiments one and two are performed for 50 times. Almost every run and all particles converge to the exact optimal value of $\Psi(x) = 0$ at position $x_{opt} = (3, \sqrt{10})$.

The trajectories of experiment one are shown in Figure 12 while Figure 13 shows the results of experiment two. In Figure 13, one can see that particles/robots not only search the target, but also take care for collision avoidance with obstacles in the environment.

Experiment three and four use 30 volume robots to search the target under constraints and also with obstacles in the environment. The purpose of this two groups of experiments is to check the algorithm of VL-ALPSO for its performance in mutual avoidance, i.e., that the robots neither during the process nor at the end overlap with each other. Thus, at most, finally only one particle/robot can occupy the exact minimal position, and the other robots should distribute themselves at other positions in the close surrounding. Here comes the important question, how to place the other robots? As VL-ALPSO is designed, other robots should at least get as close as possible to the target. So, the distribution situation becomes an important evaluation indicator to the results of the experiments. In experiment three, a Box&Whisker diagram (Tukey, 1977) is used to evaluate the distribution results. Box&Whisker methods are commonly used in the visualization of statistical analysis in many applications including the robotic area. In this method, a box and whisker graph is used to display a set of data so that one can easily see where most of the members are.

Experiment three runs 50 times and all of them siege the target. One of the runs at four
searching stages is shown in Figure 14. One can see that all volume robots find and siege the target but do not overlap with each other. Furthermore, a statistics from the 50 runs is made which evaluates their distribution status. The distribution status of 50 runs of experiment three can be seen from Figure 15. Also from this figure one can see that the 25th and 34th run have several robots that are far away from the minimum and have big objective function...
values and are even infeasible. Nevertheless, all other 48 runs get the desired results well, even in those two worse runs the main group of robots also finds and surrounds the target. Most runs get the first quartile and the third quartile below objective function value 2 which means that about 75% of the members of the swarm robots locate close to the target with high density. In summary, all 50 runs in this experiment can be accepted as obtaining the correct results from VL-ALPSO.

In experiment four, the difference is that the triangular obstacle is moved with its centroid at \((3, \sqrt{10})\) so that the target is infeasible.

One of the 50 runs in experiment four can be seen in Figure 16. The volume robots try to get close to \(x_m\), and they do not overlap with each other during the whole motion and also the obstacles are avoided. In this case, one of the most important aspects is how the volume robots distribute around the target. Several processes of the 50 runs are chosen randomly and the mean distance between robots is computed during the motion over time, see Figure 17. The distances calculated in Figure 17 are normalized. From this figure one can see that the mean distance between robots declines over time. The results from experiment four are as desired for real applications of swarm mobile robots.

Usually, for the swarm robotic system, there are two critical factors, one is the density of the swarm robots, and another is collision detection and avoidance. In the proposed algorithm design, both of them are satisfied. From the experiment results, there are some aspects which should be emphasized.

First, the algorithm has a robust convergence. During the process of searching for the target in the experiments, the robots converge to the correct location.
Figure 15: Statistics of distribution for experiment three. The horizontal lines correspond to the best, the worst and the four quartiles.
Second, although this study makes many adjustments and improvements to the basic PSO algorithm, the complexity in VL-ALPSO is still low. Moreover, since the robots are taking care of collision avoidance, VL-ALPSO it is not difficult to be modified for avoidance of dynamic obstacles in an unknown environment. This is because during the mutual avoidance of robots, they already take other moving robots as dynamic obstacles, so that the robots won’t collide with each other on the way of moving toward the convergence position which originates from a real engineering problem.

Third, the algorithm uses the information of each single robots neighborhood of certain radius, rather than the swarm best. This reduces the requirements for the robots detection ability to its surrounding environment. So, it can narrow down the exploration scope. Based on this, each single robot will be implemented with its own VL-ALPSO algorithm to realize a distributed computing environment. Thus, it is not necessary to have a central processor. Fortunately, not all of the robots are needed for finding an optimal solution. So the system can be scalable to a large number of robots. In the simulation, even more than 10000 particle robots have been used.
Figure 17: Normalized mean distance of experiment four, (a) 8th run, (b) 15th run, (c) 30th run, (d) 45th run

5 Conclusion

This chapter has presented several extensions to the stochastic particle swarm optimization algorithm and also two application examples. Through the augmented Lagrangian multiplier method we extend PSO to ALPSO which can easily be adjusted for solving optimization tasks with problem immanent equality and inequality constraints. The behavior of the basic PSO method is comparable to other stochastic algorithms, especially to evolutionary strategies. Our extension resulting in ALPSO shows convincing results without the need for infinite penalty factors which are yielding poor conditioning.

The applicability to complex engineering problems was demonstrated by optimizing the stiffness behavior of a hexapod machine tool. To increase the stiffness of the hexapod robot, we successfully applied ALPSO to the optimization problem formulated by the use of the tangential stiffness matrix.

From ALPSO to VL-ALPSO, some extension work for practical use of this algorithm for the motion planning of large scale mobile robots is given. This requires many simulation experiments. The results show that the algorithm is simple, reliable, and transferable to swarm robots. In the simulation, the robots move corresponding to their mechanical properties such as masses, inertias or external forces. This work investigates the concept of
collective coordinated movement under constraints and obstacles in the environment. For the constraints, it also uses the augmented Lagrangian multiplier method. For avoiding obstacles and other robots in the environment, it uses strategies of velocity limiting, virtual detectors, and others.

The proposed algorithm has no need of a central processor, so it can be implemented decentralized on a large scale swarm mobile robotic system and makes the robots move well coordinated. The natural next step after the simulations are finished will be to distribute the simulation to many different processors, to identify clearly the mechanical properties of a single robot and then to run everything on a small swarm of real robots.

Acknowledgements

The authors greatly appreciate the help of Dr.-Ing. K. Sedlaczek, Dr.-Ing. Ch. Henninger and Dipl.-Ing. T. Kurz, as well as of Prof. M. Iwamura from Fukuoka University, Japan. The example of investigating the stiffness behavior of the hexapod machine tool is part of the work financed by the German Research Council (DFG) within the framework of the SPP-1156 ‘Adaptronics for Machine Tools’. The work of motion planning for swarm mobile robots is partly done in the framework of SimTech, the Stuttgart Research Center for Simulation Technology. The authors also want to thank the China Scholarship Council for providing a Ph.D. fellowship to Qirong Tang to study in Germany. All this support is highly appreciated.

References


