2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

A PSO Algorithm for Mapping the Workspace Boundary of Parallel Manipulators

V.B. Saputra, S.K. Ong and A.Y.C. Nee

Department of Mechanical, Engineering, National University of Singapore
9 Engineering Drive 1, Singapore 117576

{g0600940|mpeongsk|mpeneeyc}@nus.edu.sg

***Abstract*— This paper presents a novel method to determine the multi-dimensional workspace boundary of parallel manipulators using a modified *Particle Swarm Optimization* (MPSO) framework. A generic *objective (fitness) function* that gives optimum value at points where a manipulator reaches its limit is formulated. By finding these points using MPSO, the workspace boundary of a parallel manipulator can be constructed. To demonstrate the method, the mapping of the boundary of a Stewart platform workspace is presented. The generic objective function can be modified easily to include any number of parameters and constraints, and this allows the method to be adapted for solving the workspace boundary problem of other parallel manipulators.**

***Keywords*— Parallel manipulator, workspace boundary, particle swarm optimization (PSO), multimodal optimization, Stewart Platform**

I. INTRODUCTION

OVER the last few decades, many researchers have studied the properties and physical behavior of *parallel kinematic manipulators* (PKM). Stewart platforms, which are one of the earliest PKMs invented, and other types of PKMs have been extensively studied and applied in many applications due to their high rigidity and payload-to-weight ratio. However, PKMs usually have coupled motion capabilities, therefore, their workspace representation becomes a problem [1]. In addition, it is impossible to visualize completely the workspace representation of PKMs that have more than three *degrees of freedom* (DOF). Large numbers of studies addressing workspace boundary mapping have been published, but most of them are limited to 3-DOF.

The *discretization* method has been used in numerous research studies [2-4]. This method divides the configuration space based on a particular coordinate system into regular grid of *nodes* and the size of each node is specified as a *sampling step*. This method tests all the nodes to determine whether they belong to the workspace. Therefore, the sampling step determines the accuracy of the workspace boundary. The *discretization* method can be applied to many types of manipulators. However, this method has two main disadvantages [16], namely, it needs a longer computation time when a higher accuracy is required (smaller sampling step) and it usually fails to detect workspace *voids*.

Another well-known method for finding parallel manipulator workspace has been proposed [5-8], and this is often referred to as the *geometrical* method. It is efficient and accurate in mapping the workspace boundary, which is obtained by intersecting geometrical objects that represent feasible ranges of motion of the actuated joints. However, in order to compute the *reachable workspace*, there is a need to transform constraints that limit the workspace (e.g., link interference, maximum range of passive joints) into geometrical representations. This transformation, however, is not always possible in general.

Several researchers [9-11] have proposed an interesting method. Firstly, the kinematic constraint equations that describe the range of motion achievable by the manipulator are formulated. Next, if the *Jacobian* matrix of these equations is found to have a *row rank deficiency*, the corresponding configuration is at the boundary of the workspace. These researchers developed a numerical scheme that can find one of these configurations and use the *continuation method* to build the rest of the boundary. Similarly, Snyman et al. [12] proposed to build the boundary using a *constrained optimization method*. Merlet [13] proposed to use the *interval analysis* to determine the various types of workspace boundaries through the estimation of *bisection* boxes that represent the workspace. Although it can handle any number of DOF and constraints, the computation is quite time-consuming.

In this paper, a simple and fast approach for mapping the multi-dimensional workspace boundary of a general parallel manipulator with any number of DOF and constraints is proposed. A criterion is defined for determining the boundary and a method based on a modified version of the *particle swarm optimization* (PSO) algorithm is developed [14] to map the points on the boundary. This method is illustrated using a Stewart platform (SP) mechanism.

II. PROBLEM FORMULATION

A configuration state of all the working bodies of a mechanism can be characterized by a generalized coordinate vector $\boldsymbol{q} \in \boldsymbol{R}^{nq}$ that satisfies the following m independent holonomic *kinematic constraint equations* [12].

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0} \;.\; \boldsymbol{\Phi}: \boldsymbol{R}^{nq} \rightarrow \boldsymbol{R}^{m} \quad \boldsymbol{q} = [\boldsymbol{v}, \boldsymbol{u}, \boldsymbol{w}]^{T} \tag{1}$$

Vector $\boldsymbol{q}$ consists of the *input*, *output* and *intermediate coordinate*, denoted by $\boldsymbol{v}\in\boldsymbol{R}^{nv}$, $\boldsymbol{u}\in\boldsymbol{R}^{nu}$ and $\boldsymbol{w}\in\boldsymbol{R}^{nw}$, respectively ($nq = nv + nu + nw$) [10]. The *input coordinate* $\boldsymbol{v}$ is the subset of $\boldsymbol{q}$ that can be changed to control a mechanism, and it corresponds to the actuated joints coordinates. The *output coordinate* $\boldsymbol{u}$ constitutes the subset of $\boldsymbol{q}$ that defines the useful functionality of the platform, and it corresponds to the end-effector pose (positions and orientations). The *intermediate coordinate* $\boldsymbol{w}$ is the subset of $\boldsymbol{q}$, and it constitutes all the other uncontrolled passive joints coordinates. For PKMs, (1) can be rewritten explicitly as *inverse kinematics*, $\boldsymbol{v}=\boldsymbol{v}(\boldsymbol{u})$ and $\boldsymbol{w}=\boldsymbol{w}(\boldsymbol{u})$. Therefore, $\boldsymbol{q}=[\boldsymbol{v}(\boldsymbol{u}), \boldsymbol{u}, \boldsymbol{w}(\boldsymbol{u})]^T$.

The workspace of a manipulator is a collection set A of all the reachable *output coordinates*. An output coordinate $\boldsymbol{u}$ is reachable if the manipulator configuration $\boldsymbol{q}=[\boldsymbol{v}(\boldsymbol{u}), \boldsymbol{u}, \boldsymbol{w}(\boldsymbol{u})]^T$ satisfies all the *constraints* that limit the range of the manipulator movement (note that the constraints can also be imposed on $\boldsymbol{u}$). Therefore, the boundary of the workspace is a collection set dA of the output coordinates, where the manipulator reaches any of its constraints. These constraints can be expressed by the *constraint functions* $\boldsymbol{f}_{Ci}$, which have a generic form given below.

$$\boldsymbol{f}_{Ci}(\boldsymbol{q})=\begin{cases}1, \textit{if } \boldsymbol{q} \textit{ satisfies the } i^{th} \textit{ contraint}\\ 0, \textit{otherwise}\end{cases},\ i=1\ldots n \qquad (2)$$

where n is the number of constraints imposed.

Assume that an arbitrary reachable output coordinate $\boldsymbol{c}$ interior to the boundary is selected. To find the boundary, the *multimodal* optimization problem is formulated as follows.

$$\max_{\boldsymbol{u}} \boldsymbol{f}(\boldsymbol{u})=\delta\|\boldsymbol{u}-\boldsymbol{c}\|.\boldsymbol{f}_{C1}.\boldsymbol{f}_{C2}\ldots.\boldsymbol{f}_{Cn},\ \boldsymbol{f}(\boldsymbol{u})\ge 0 \qquad (3)$$

To find N *local maximum* points $\boldsymbol{u}_{Li}$ subject to:

$$\boldsymbol{f}(\boldsymbol{u}_{Li})\ge\boldsymbol{f}(\boldsymbol{u})\ \ \forall\boldsymbol{u}\in\boldsymbol{u}(\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i\ \ \lambda\in R^+, i=1..N \quad (4a)$$

where δ is any real constant greater than zero, and $\|.\|$ denotes the *Euclidean distance*. Equations (2) and (3) imply that the value of $\boldsymbol{f}(\boldsymbol{u})$ will be zero or minimum if any of the constraints is violated. Therefore, $\boldsymbol{u}(\lambda)$ that maximizes $\boldsymbol{f}(\boldsymbol{u})$ will be on the boundary dA (see Fig. 1). The value of λ is found from the optimization, which will be explained in the next section. New constraints can be added to (3) as long as they can be explicitly expressed as functions of $\boldsymbol{q}$. For example, to create a *singularity-free* workspace boundary, a test function can be defined as an additional constraint:

$$\boldsymbol{f}_C(\boldsymbol{q})=\begin{cases}1, \textit{if } \kappa(\boldsymbol{J}(\boldsymbol{q}))<C\\ 0, \textit{otherwise}\end{cases} \qquad (4b)$$

where κ is the condition number of the *Jacobian* matrix, and C is a limiting constant describing the allowable closeness of the end-effector to the singularities [6, 18].

From Fig. 1, the constraint function $\boldsymbol{f}_{Ci}$ shapes the boundary accurately such that the value of $\boldsymbol{f}(\boldsymbol{u})$ is zero if any constraint is violated. Using (4a), the search space can be diversified such that during optimization, N local maximum points $\boldsymbol{u}_{Li}$ on the intersections between the boundary dA and the straight lines $\boldsymbol{u}(\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i$ through the point $\boldsymbol{c}$ (see Fig. 2) can be found. The directions of the unit vector $\boldsymbol{b}_i$ are selected such that the straight lines intersect dA at the broadest range and with approximately equal spacing so that the points $\boldsymbol{u}_{Li}$ are located sufficiently dense to represent the boundary.

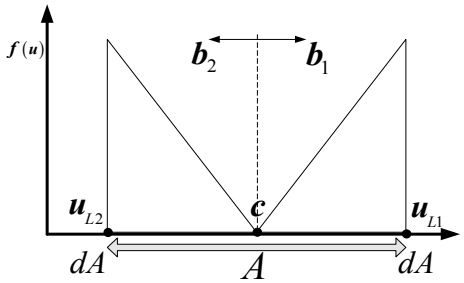

Fig. 1. Mapping of the value of $\boldsymbol{f}(\boldsymbol{u})$ for $nu=1$ (1-dimensional workspace). Points at the boundary $\boldsymbol{u}_{L1}$ and $\boldsymbol{u}_{L2}$ are found by maximizing $\boldsymbol{f}(\boldsymbol{u})$.

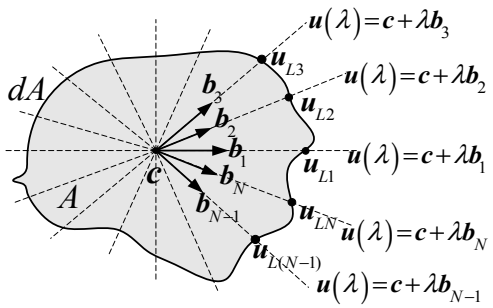

Fig. 2. Mapping of the boundary dA using the intersections between the parameterized straight lines $\boldsymbol{u}(\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i$ with the boundary. At each intersection point $\boldsymbol{u}_{Li}$, $\boldsymbol{f}(\boldsymbol{u}_{Li})$ gives the local maximum value along the corresponding straight line.

III. FINDING THE POINTS ON *dA* USING A MODIFIED PSO

PSO [14,15] can be implemented easily and it is computationally inexpensive in terms of both memory requirements and CPU speed. PSO is an iterative search process to find the optima of an objective function. It is a stochastic population-based search method where the population is referred to as a *swarm S*. The swarm consists of a number of individuals called *particles*. Each particle is characterized by its position in the search space that represents potential solution to the optimization problem. The position of each particle is updated based on the governing equation of PSO at each discrete time step t according to its own experience and that of its neighbors to find the optimum position.

A modified PSO (MPSO) algorithm is proposed to solve the optimization problem described in (3), (4a), and (4b). The positions of the particles are the output coordinates $\boldsymbol{u}_i$ and these positions are updated at every discrete time step t to search for *local maximum* solutions, according to the following equations.

$$\boldsymbol{v}_i(t+1)=\varphi(t)\boldsymbol{v}_i(t)+\alpha.rand().\left[\boldsymbol{u}_{pbest,i}-\boldsymbol{u}_i(t)\right] \qquad (5)$$

$$\boldsymbol{u}_i(t+1)=\boldsymbol{u}_i(t)+\boldsymbol{v}_i(t+1) \qquad (6)$$

where $\varphi(t) \in [0,1]$ is a positive single valued function that drops linearly after each time step and α is the *cognitive acceleration constant*. The function *rand*() generates a uniformly distributed random number in the interval [0,1]. Moreover, $\boldsymbol{u}_{pbest,i}$ is updated for each particle in each iteration. The best position of a particle $\boldsymbol{u}_{pbest\ i}$ is the position with the best objective value $\boldsymbol{f}(\boldsymbol{u}_{pbest\ i})$ that has been travelled by the i^{th} particle. The search process will be considered successful if all the particles have found the best positions such that $\| \boldsymbol{u}_{pbest,i} - \boldsymbol{u}_{Li} \| < \varepsilon$, where ε is a small number. Therefore, the boundary can be approximated by the locations of $\boldsymbol{u}_{pbest,i}$.

The following modifications are made to MPSO.

1) Initial positions and velocities. All the particle positions are initialized as $\boldsymbol{u}_i(0) = \boldsymbol{c}$ and the initial particle velocities as $\boldsymbol{v}_i(0) = v_0\boldsymbol{b}_i$, where v_0 is the *initial velocity constant*. Therefore, the particles will move towards the boundary starting from the point $\boldsymbol{c}$ (the start point $\boldsymbol{c}$ is chosen arbitrarily and is interior to the boundary).

2) Velocity update and position clamping. When the particle velocity vector $\boldsymbol{v}_i(t)$ is updated as according to (5), only its magnitude changes (its direction is still parallel to $\boldsymbol{b}_i$). Therefore, the i^{th} particle is assigned to search for a local solution only on the straight line $\boldsymbol{u}_i\ (\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i$. The unit vector $\boldsymbol{b}_i$ determines the search direction of the i^{th} particle. Furthermore, since λ cannot be negative (this is to avoid overlapping with another search direction), the i^{th} particle is constrained from moving in the reverse direction of $\boldsymbol{b}_i$ across the point $\boldsymbol{c}$. Hence, for any particle position $\boldsymbol{u}_i(t)$ that has a position where $\boldsymbol{u}_i(\lambda) = \boldsymbol{c}+\lambda\boldsymbol{b}_i$ and $\lambda < 0$, $\boldsymbol{u}_i(t) = \boldsymbol{c}$.

3) Termination condition. In MPSO, more iterations tend to give a better result. However, the search process has to stop. A general *stopping condition* is when the *maximum number of iterations* has been exceeded. Another stopping condition is when *no improvement* in the best positions of the particles is observed over a number of iterations. In the examples in this paper, the maximum number of iterations in the MPSO was set as 40 based on an observation of the convergence speed of the MPSO simulation. The framework based on the MPSO algorithm to solve the workspace boundary for a general PKM is summarized in Fig. 3.

IV. Application to 6-DOF Stewart Platform

Based on the proposed method, the *reachable* workspace boundary of a 6-DOF SP (see Fig. 4) is determined. A coordinate frame A is fixed to the base and the coordinate frame M is attached to the mobile platform (end-effector) at a reference point O_M. The generalized *output coordinate* $\boldsymbol{u}$ is the position and orientation of the frame M with respect to frame A and nu=6.

$$\boldsymbol{u} = \left(X, Y, Z, \phi, \theta, \psi\right)^T \tag{7}$$

The generalized *input coordinate* $\boldsymbol{v}$ is composed of six actuated prismatic joints lengths (leg lengths) l_i, i=1…6, and nv=6.

$$\boldsymbol{v} = \left(l_1, l_2, l_3, l_4, l_5, l_6\right)^T \tag{8}$$

It is well known that the *inverse kinematics* of the SP gives a unique solution. For a known complete description in the output coordinate system, the leg lengths or the input coordinate can be calculated using the following equation.

$$l_i = \left\|\boldsymbol{L}_i\right\|, \text{where } \boldsymbol{L}_i = M_i - A_i\ , M_i = \boldsymbol{u}_P + {}^A\boldsymbol{R}_M \cdot {}^M M_i\ \ ; i = 1..6 \tag{9}$$

M_i and A_i are the coordinates of the spherical and universal joints attached to the mobile platform and the base respectively with reference to frame A. The vector $\boldsymbol{u}_P = (X, Y, Z)^T$ is the position of the reference point O_M with respect to frame A. The *rotation matrix* ${}^A\boldsymbol{R}_M$ is computed from the orientation of frame M with reference to frame A. Lastly, ${}^M M_i$ is the coordinates of the spherical joints attached to the mobile platform with reference to frame M.

The constraints that limit the workspace of the SP are modeled, namely, the ranges of the prismatic joints, collisions between the legs, and the physical limitations of the passive joints. These constraints are formulated as (2) and included in (3). Hence, the objective function $\boldsymbol{f}(\boldsymbol{u})$ will have a zero value if any of these constraints are violated.

For conciseness, the calculations of the angles of the passive joints and the distances between the legs have been

```
//Create and initialize an nu – dimensional swarm
for each particle i = 1,….,N do
   // Set particle positions at starting point c
   u_i = c
   // Set initial particle velocities
   v_i = v_0 b_i
end
// begin the iterative search process
repeat
   for each particle i = 1,….,N do
      // set the personal best position
      if f(u_i) < f(u_pbest i) then u_pbest i = u_i
      update the velocity using (5)
      update the position using (6)
      // position clamping
      If u_i (λ)=c+λb_i and λ < 0, then u_i = c.
   end
   calculate the necessary variables for stopping condition (e.g.,
   workspace size V)
until stopping condition is true;
```

Fig. 3. MPSO pseudo code (the time-step t is omitted for simplicity; however it is incremented in every **repeat** – **until** loop).

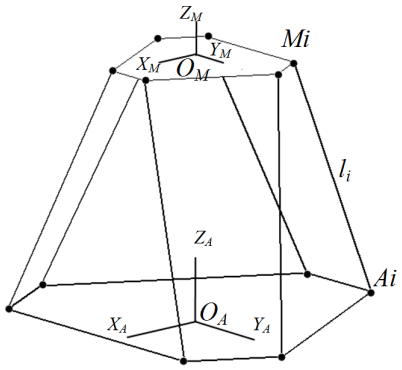

Fig. 4. A Stewart platform diagram.

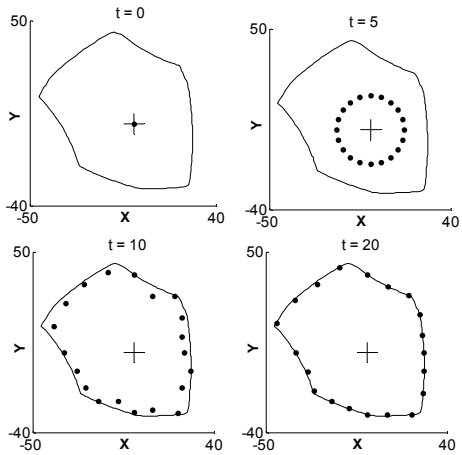

Fig. 5. 2D workspace boundary generation using MPSO (Z = -270 mm and $[\Phi, \theta, \psi]^T = [0,0,0]^T$. The MPSO searches from the start point $\boldsymbol{c} = [0, 0]^T$ denoted by the cross sign.

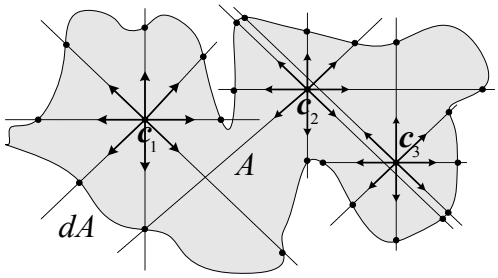

Fig. 6. Illustration of boundary mapping with *multi-swarm* MPSO with three swarms and 8 particles in each swarm. The starting points $\boldsymbol{c}_1$, $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ are distributed randomly at initialization.

omitted. Please refer to [17] for more details.

A. 2D workspace boundary

For the 2D workspace of the SP, some components of the output coordinate are fixed, namely, the translational component Z and the three rotational components $[\Phi, \theta, \psi]^T$. Therefore, a 2D workspace boundary on an XY plane will be determined. The MPSO parameters used in the simulation example are given in the appendix. Fig. 5 shows the result of the MPSO for a 2D workspace boundary of the SP. A discretization method was also used to generate the boundary (solid curve in Fig. 5) for comparison with the MPSO approach. The search direction is selected such that $\boldsymbol{b}_i = [\sin \beta_i, \cos \beta_i]^T$ where β_i are distributed at equal angular interval, $\beta_i = i.360/N$ degree [12].

Several improvements have been made to the proposed MPSO algorithm.

1) Velocity boost. From a preliminary simulation, it was observed that due to the monotonically increasing objective function in (2), the particles start to converge (stop moving) before reaching the local maxima on the boundary. To avoid premature stagnation, a velocity boost is injected for every M iterations, agitating all the particles with a new velocity $\boldsymbol{v}_i(0)=v_1\boldsymbol{b}_i$, where v_1 is the *velocity boost constant*.

2) Multi-Swarm MPSO. To find more points on a boundary, a *multi-swarm* MPSO is implemented. The multi-swarm MPSO has multiple K swarms S_k, k=1…K, with a start point $\boldsymbol{c}_k$ in each swarm. Each swarm works as a single MPSO outlined earlier. By using several swarms, a complex boundary can be traced. The center points $\boldsymbol{c}_k$ can be selected arbitrarily or computed before the multi-swarm MPSO starts. The *randomly scattered* or *equally spaced* starting points within the search space can be used (see Fig. 6). The number of swarms, K, is determined by the users. The feasible $\boldsymbol{c}_k$ is selected if and only if they satisfy the constraints and are within the boundary.

3) Two-Phase Dynamic MPSO. One of the advantages of the MPSO-based workspace boundary mapping is the ability to store the states of the particles. After the particles have converged, the particle positions can be stored for further analysis. A *two-phase* MPSO is proposed. In the *first phase*, the search process begins from the *zero state*, which is when all the particles are at the initial starting point $\boldsymbol{c}$. In the *second phase*, the particles continue the search process from the last positions when the first phase ends (the particles have converged). The two-phase MPSO has the ability to track changing local maxima on the boundary without the need for repeating the search process from the zero state. The second phase is used when the fixed pose parameters of $\boldsymbol{u}$ or the constraints imposed on $\boldsymbol{q}$ change slightly. For instance, in finding the 2D workspace boundary in this example, $[Z, \Phi, \theta, \psi]^T$ is fixed, and if any of these pose coordinates (Z, Φ, θ, or ψ) is changed on a small scale, the workspace boundary will be altered slightly. Therefore, the second phase can be used to search for this new boundary starting from the particle positions acquired from the previous search process (the first phase).

At the beginning of the second phase, the second initial velocity $\boldsymbol{v}_i(0)=v_{02}\boldsymbol{b}_i$ is injected again but with a smaller magnitude than that of the first phase ($v_{02} < v_0$). Since the initial velocity direction is always diverging from the start point, the PSO may fail to track the new local maxima if they are closer to the start point than the last positions. Therefore, the particles must be re-initiated closer to the start point before the second phase starts.

$$\boldsymbol{x}_i(0) = \gamma(\boldsymbol{x}_i^* - \boldsymbol{c}) + \boldsymbol{c} \tag{10}$$

where $\gamma \in [0, 1]$ is the *tracking coefficient* and $\boldsymbol{x}_i^*$ is the last particle position from the first phase. The lower the γ, the closer will be the initial positions to the start point $\boldsymbol{c}$ and the MPSO can track a greater variety in the new local maxima locations, but would require more iteration to converge.

B. 3D workspace boundary

To generate 3D workspace boundary, the three rotational components of $[\Phi, \theta, \psi]^T$ are kept constants. The subset $\boldsymbol{u}_P = (X, Y, Z)^T$ of $\boldsymbol{u}$ will be the input parameters to the MPSO (see Fig. 7).

In 3D, it is convenient to use the *spherical coordinate system* with MPSO, where the parameterized straight line $\boldsymbol{u}_i(\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i$ in (3) can be defined easily. The direction of $\boldsymbol{b}_i$

can be selected based on equal angular intervals of the *inclination angle* and the *azimuth angle*. Therefore, the particle positions are parameterized by the radial distance λ from point $\boldsymbol{c}$.

C. Higher dimensional workspace

To generate the boundary of workspace such that $nu > 3$, the selection of the search direction $\boldsymbol{b}_i$ must be viewed in a higher dimensional setting. Although there would be no graphical visualization available for these workspace boundaries, the method developed here can still be applied efficiently. From 2D workspace analysis in a XY plane, an additional variable Z in $\boldsymbol{u}$ is introduced so it becomes a 3D positional workspace (with a fixed orientation). Likewise, more variables can be introduced for analyzing a higher dimensional workspace. However, there is no specific coordinate system in this setting. Therefore, the direction of $\boldsymbol{b}_i$ is selected by taking equally spaced values of coordinate variables as vectors and normalizing them. Then, the MPSO will proceed as in previous cases. This process can be extended to include any number of coordinate variables (e.g., 5D, 6D workspace).

D. Comparison with other approaches

The workspace boundary obtained using the MPSO was more accurate than the boundary obtained using the *discretization* method. This is because the constraints are defined exactly using equation (4a) such that the particles could find the optimum position precisely. In terms of speed, the MPSO is significantly faster than the *discretization* method as the search speed of MPSO is as fast as the velocities of the particles and is only constrained by the initial velocities v_1 and v_{01}.

Regardless of the number particles used in MPSO, the boundary generated will contain some errors due to the approximation of the boundary (gaps between the real and the interpolated boundaries) and the randomness in MPSO. Therefore, more exact computations, such as the *direct search* or the *interval analysis* based methods, can be used to refine the result after the MPSO has found the near optimum solutions. Increasing the number of particles or iterations will lead to a decrease in the error. From a practical point of view, a full and accurate analysis may not be strictly necessary as a good approximate may be sufficient and the region near the boundary is generally not preferred for performing tasks.

V. Conclusion and Future Work

The mapping of parallel manipulator workspace boundary is formulated as an optimization problem and solved using the MPSO algorithm. The performance in terms of speed is significantly better than the standard *discretization* method. However, in terms of accuracy, it is relatively less accurate than the *geometric* or other *numerical* approaches.

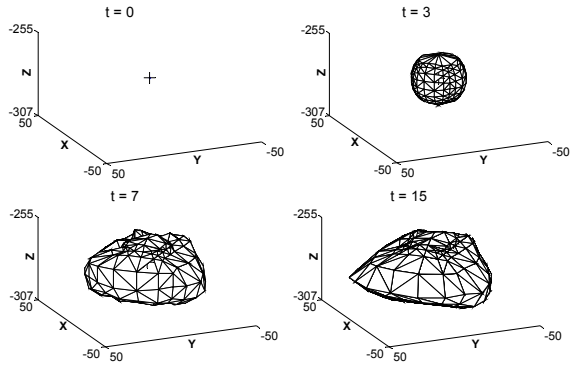

Fig. 7. 3D workspace boundary generation using MPSO $([\Phi, \theta, \psi]^T = [0,0,0]^T$. The MPSO searches from the start point $\boldsymbol{c} = [0, 0, -270]^T$ denoted by the cross sign. The surface boundary is shown as tessellated from the positions of particles.

Several aspects of MPSO have been addressed to tune the algorithm for this type of problem. Future work may address some of the following issues.

A. Finding better start points

The particles in MPSO will search for points on the boundary along the lines $\boldsymbol{u}_i$ $(\lambda)=\boldsymbol{c}+\lambda\boldsymbol{b}_i$. It is possible that some regions of the workspace boundary are not represented. As illustrated in Fig. 6, some regions of a complex boundary *dA* are not covered when the multi-swarm PSO is used. Therefore, a proper selection of the number of start points ($\boldsymbol{c}_k$) and their locations are important issues to address.

B. Intelligent tuning of MPSO parameters

From the examples in this paper, it can be seen that is feasible to use the MPSO approach with pre-defined parameters, which have been obtained through observations. However, these parameters may not work well for a parallel manipulator with a different geometric configuration. In future work, research will be conducted on the automatic tuning of these parameters based on the geometric configuration of a manipulator. For example, it was observed that there is a relationship between the initial velocities and the size of a manipulator. Thus, the algorithm to be developed will generate the workspace from the geometric data of a manipulator, such that automatic parameter tuning can be achieved by the algorithm itself.

C. Extension to general manipulator

The method presented in this paper can be extended to solve the workspace boundary of serial kinematic manipulators, which in this case, the *forward kinematics* is easier to be solved. Therefore, the *input coordinate space* is the search space for the MPSO to find the workspace boundary. However, the collections of the achievable *input coordinates* may be irregularly distributed over the search space. Therefore, the *multi-swarm* MPSO technique is

essential for detecting all the feasible configurations. Future work will address the general framework for using the MPSO to generate the workspace boundary for a general manipulator.

APPENDIX

Table I shows the data for the SP geometry used in the example in this paper. In addition, the range of the allowable leg lengths is 327 mm < l_i < 280 mm. The maximum allowable spherical joints angle is 29 degrees, the maximum allowable universal joints angle is 45 degrees and D = 36.1 mm. Table II summarizes the parameters used in the MPSO algorithm. These parameters are selected based on the MPSO simulations.

TABLE I
GEOMETRY OF THE STEWART PLATFORM*

i	A_i	${}^{M}M_i$
1	$\begin{bmatrix} -183.350 \\ -158.786 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} -77.942 \\ 45.000 \\ 0.000 \end{bmatrix}$
2	$\begin{bmatrix} -158.786 \\ -91.675 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} -45.000 \\ -77.942 \\ 0.000 \end{bmatrix}$
3	$\begin{bmatrix} 91.675 \\ -158.786 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} 0.000 \\ -90.000 \\ 0.000 \end{bmatrix}$
4	$\begin{bmatrix} 158.786 \\ -91.675 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} 90.000 \\ 0.000 \\ 0.000 \end{bmatrix}$
5	$\begin{bmatrix} 91.675 \\ 158.786 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} 77.942 \\ 45.000 \\ 0.000 \end{bmatrix}$
6	$\begin{bmatrix} 0.000 \\ 183.350 \\ 0.000 \end{bmatrix}$	$\begin{bmatrix} -45.000 \\ 77.942 \\ 0.000 \end{bmatrix}$

* all units are in millimetres.

TABLE II
MPSO PARAMETERS

Parameters	Value
Number of particles	20 (2D case), 200 (3D case)
Number of max. iterations	40
Number of swarms, K	3 (multi-swarm MPSO)
M	15
Initial velocity v_0	10 mm/iteration
Boost velocity v_1	2 mm/iteration
δ in equation (4a)	1
φ in equation (5)	From 0.9 gradually down to 0.6 over t
α in equation (5)	2,1
Number of local maxima N	20 (for 2D case), 8 (for 2D multi-swarm), 200 (3D case)
Tracking coefficient γ	0.9
Second initial velocity v_{02}	4 mm/iteration

REFERENCES

[1] J.-P. Merlét, "Parallel Robots: Open Problems," in *Ninth International Symposium of Robotic Research,* 1999, pp. 27-32.

[2] E.F. Fichter, "A Stewart Platform-Based Manipulator: General Theory and Practical Construction," *International Journal of Robotic Research*, vol. 5, no. 2, pp. 157-181, 1986.

[3] Z. Wang, Z.-X. Wang, W.-T. Liu and Y.-C. Lei, "A study on workspace, boundary workspace analysis and workpiece positioning for parallel machine tools," *Mechanism and Machine Theory,* vol. 36, no. 5, pp. 605-622, 2001.

[4] F. Pernkopf and M. Husty, "Workspace Analysis of Stewart-Gough Manipulators using Orientation Plots," in *International Symposium on Multibody Systems and Mechatronics*, 12-14 Sept. 2002, Mexico City.

[5] J.-P. Merlet, "Determination of the Orientation Workspace of Parallel Manipulators," *Journal of Intelligent and Robotic Systems,* vol. 13, pp. 143-160, 1995.

[6] C.M. Gosselin and J. Angeles, "The optimum kinematic design of a spherical three degree-of-freedom parallel manipulator," *ASME Journal of Mechanisms Transmissions Automation Design,* vol. 111, no. 2, pp. 202-207, 1989.

[7] J.-P. Merlet, C.M. Gosselin and N. Mouly, "Workspaces of planar parallel manipulators," *Mechanism and Machine Theory*, vol. 33, no. 1-2, pp. 7-20, 1998.

[8] C. Gosselin, "Determination of the workspace of 6-dof parallel manipulators," *ASME Journal of Mechanical Design,* vol. 112, no. 3, pp. 331-336, 1990.

[9] D.Y. Jo and E.J. Haug, "Workspace Analysis of Multibody Mechanical Systems Using Continuation Methods," *ASME Journal of Mechanism Transmissions Automation Design*, vol. 111, pp. 581-589, 1989.

[10] E.J. Haug, C.M. Luh, F.A. Adkins and J.-Y. Wang, "Numerical Algorithms for Mapping Boundaries of Manipulator Workspaces," *ASME Journal of Mechanical Design*, vol. 118, no. 2, pp. 228-234, 1996.

[11] L.-C.T. Wang and J.-H. Hsieh, "Extreme Reaches and Reachable Workspace Analysis of General Parallel Robot Manipulators," *Journal of Robotic Systems*, vol. 15, pp. 145-159, 1997.

[12] J.A. Snyman, L.J. Du Plessis and J. Duffy, "An Optimization Approach to the Determination of the Boundaries of Manipulator Workspaces," *ASME Journal of Mechanical Design*, vol. 122, no. 4, pp. 447-456, 2000.

[13] J.-P. Merlét, "Determination of 6D workspaces of Gough-type parallel manipulator and comparison between different geometries," *International of Robotics Research,* vol. 18, no. 9, pp. 902-916, 1999.

[14] J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization," *in Proc. of the IEEE International Joint Conference on Neural Networks,* pp. 1942-1948, 1995.

[15] R. Eberhart and J. Kennedy, "A New Optimizer using Particle Swarm Theory," in *Proc. of 6th International Symposium on Micro Machine and Human Science,* 1995, pp. 39-43.

[16] J.P. Merlet, *Parallel Robots* (Solid Mechanics and Its Applications vol. 128), Springer-Verlag, 2006, pp. 156-159.

[17] O. Masory and J. Wang, "Workspace evaluation of Stewart platforms." *Advanced Robotics* , vol. 9, no. 4, pp. 443-61, 1995.

[18] P.A. Voglewede and I. Ebert-Uphoff, "Measuring "closeness" to singularities for parallel manipulators." in *IEEE Int. Conf. on Robotics and Automation*, pp 4539--4544, New Orleans, 2004.