An open-source tool for path synthesis of four-bar mechanisms

J.L. Torres-Moreno[a,*], N.C. Cruz[b], J.D. Álvarez[b], J.L. Redondo[b], A. Giménez-Fernandez[a]

[a] *Department of Engineering, University of Almería, CIESOL-ceiA3, Almería, Spain*
[b] *Department of Informatics, University of Almería, CIESOL-ceiA3, Almería, Spain*

Abstract

Four-bar mechanisms are widespread in industrial and quotidian applications, but their design requires deep knowledge and experience in Mechanisms and Machine Theory. In this area, path synthesis is a common problem. It consists in calculating the length of the bars defining a mechanism so that a particular point of its structure follows the desired trajectory. This paper presents SALAR Mechanism Synthesizer, a new open-source software package that aims to be an easy-to-use yet powerful tool for addressing this kind of problem. This tool offers four different optimizers from the literature, one of them specially designed for mechanism synthesis. It also includes a new method for expressing and comparing paths, and hence, to define the optimization criterion. This method, referred to as normalized shape-descriptor vectors (NSDV) and presented in this paper, allows us to handle paths without being affected by translation, rotation, and scale changes. The software package has been used for addressing six well-known problems from the literature. According to the results achieved, it arises as a valuable tool in terms of accuracy, convenience, and availability.

Keywords: four-bar mechanisms, optimization, path synthesis.

1. Introduction

For years, four-bar mechanisms, also known as linkages, have been widely used in industrial and everyday applications. There exist two different approaches for studying them: analysis and synthesis. Analysis may focus on i) kinematics, that is, the movement that mechanisms describe without attending forces, and ii) dynamics, i.e., when considering forces and inertial parameters. In both cases, the analyzed mechanisms must have been previously designed. On the contrary, synthesis deals with initially determining their geometrical parameters to perform the desired motion. Depending on the requirements, this second approach can be further classified into function, motion, and path synthesis, which are explained below.

Function synthesis techniques focus on defining a relation between the crank input angles and the angular displacement of the rocker, which usually acts as the output link of the mechanism. Automobile steering systems [1] and solar trackers [2] are application examples of this approach. In motion synthesis, the goal is to control the movement of the coupler link, also known as the floating link or connecting rod, including its position and orientation. For instance, this strategy is valuable for loading machines [3] and airplane high-lifts [4]. Concerning path synthesis techniques, they focus on the motion described by a specific point of a linkage, referred to as the coupler point, for a given set of input angles. These angles can be either part of the requirements, leading to a prescribed-timing problem, or can be freely chosen by the designer. Walking machines [5] and machining tools [6] are some fields in which this kind of problem arises.

This work deals with path synthesis, which requires minimizing the difference between the desired path and the one described by the coupler point of the linkage. In this context, it is common to use the Euclidean distance between the points of both curves to define the cost function [7, 8, 9], which is known as an absolute error-based metric. This strategy is straightforward yet limited: It cannot appreciate similarity when the path produced by the considered mechanism is either translated, rotated or scaled with respect to the reference.

*Corresponding author
Email addresses: jltmoreno@ual.es (J.L. Torres-Moreno), ncalvocruz@ual.es (N.C. Cruz), jhervas@ual.es (J.D. Álvarez), jlredondo@ual.es (J.L. Redondo), agimfer@ual.es (A. Giménez-Fernandez)

However, these variations might not prevent the engineer from accepting the studied configuration after transforming its variables adequately. For this reason, there exist alternatives to describe paths regardless of their size and position. For instance, Nadal et al. [10] propose to use turning functions [11], which are popular for pattern recognition [12, 13, 14], to compare paths. Another valid option considered for this purpose is the use of Fourier descriptors [15, 16]. Despite not being widely used in mechanism synthesis yet, they are promising strategies according to the results found by the referred authors. This work also proposes a new method to represent paths: normalized shape-descriptor vector(s) (NSDV(s)), which are similar to turning functions but simpler to define.

There exist three main resolution approaches to address the three types of synthesis problems explained above: graphical, analytical, and optimization-based methods. Graphical strategies allow obtaining approximate solutions fast, but they lack accuracy. Analytical methods are the most accurate, but their complexity significantly increases with the number of target points. Finally, optimization-based strategies offer accuracy levels similar to the previous group but without the limitations in the number of target points that can be handled. The apparition of computers boosted these methods. The recent literature reflects numerous applications of them [17], and they are also the choice in this work. In this context, the use of evolutionary optimizers is significantly extended [7, 8, 9, 10, 18, 19]. However, there are alternatives. One of them is the proposal in [20, 21], which combines global evolutionary algorithms and local gradient-based optimization. Other alternatives are the approach presented in [6], based on the Haar Wavelet, the utilization of artificial neural networks in [16], non-convex optimization [22], and particle swarm algorithms [23].

Concerning software packages, there exist powerful CAD/CAE tools for the analysis and simulation of linkages. Solidworks, Solidedge, MSC-Adams, and the open-source package FreeCAD are the most popular tools for these tasks. However, as far as the authors of this paper know, there are fewer options for mechanism synthesis. One of the most popular packages is ASOM [24], which is commercial and can be used for both analysis and synthesis of different types of multi-bar linkages. There exist other commercial alternatives such as SAM [25], WinMecC [26], and the tool presented in [27]. It is also possible to mention MotionGen [28], which is for mobile devices and can solve problems of up to five precision points. Unfortunately, none is open-source; some are expensive and do not permit trying different optimization strategies.

The principal motivation of this work is to reduce the lack of open-source tools for path synthesis of mechanisms by releasing a fully-featured tool, written in MATLAB, for this purpose. Besides, this work presents a new methodology for defining the underlying optimization problem, and it is included in the software package. The released tool also features an easy-to-use graphical user interface that allows users to choose between the different built-in optimization algorithms and cost functions. The rest of the paper is structured as follows: Section 2 states the problem formulation and the simulation context. Section 3 introduces the new method proposed for shape encoding and comparison. Section 4 presents the optimization methods included in the tool. Section 5 describes the test problems used to assess the tool. Finally, Section 6 contains the conclusions and future work.

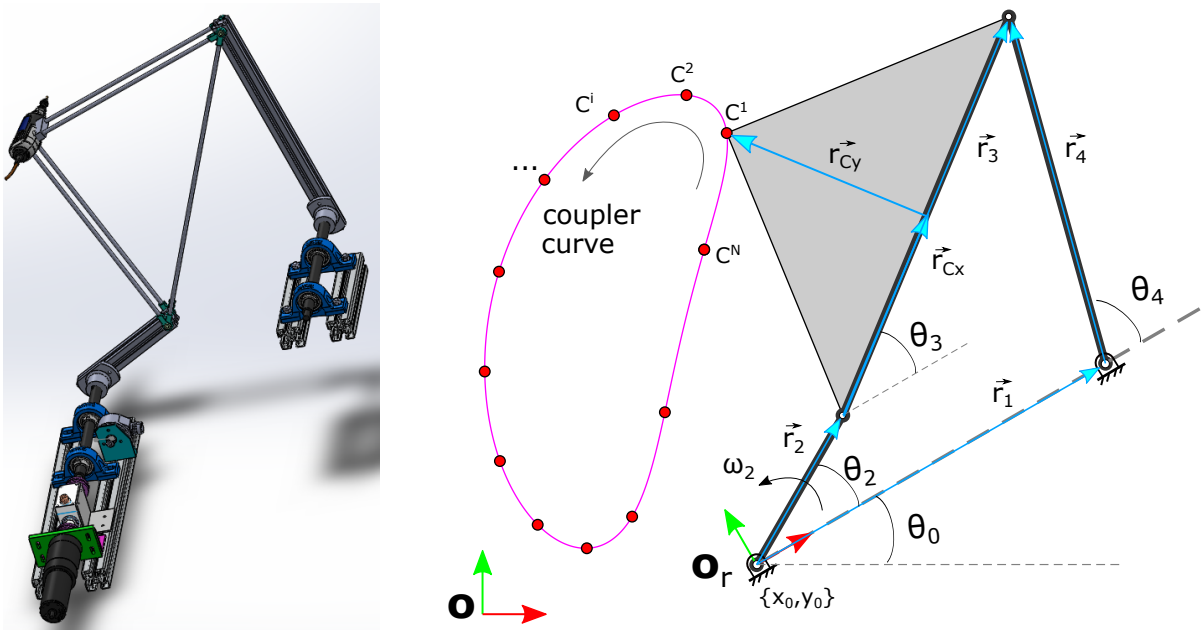
2. Problem description

This section explains the different types of path synthesis problems supported by the implemented software package. Before that, it provides the reader with the underlying concepts related to four-bar mechanism modeling and their implications in the synthesis problem.

2.1. Preliminary concepts

The path synthesis problem consists in determining the length and orientation of the bars comprising a linkage. The objective is that a particular point of its floating link, known as coupler point, describes the desired path when the crank moves. Thus, in a machine like the one depicted in Figure 1a, the milling tool connected to the coupler link must describe the specified working path.

According to the nomenclature used in Figure 1b, the tip of the tool corresponds to coupler point $\mathbf{C}^i = \left[C_X^i \; C_Y^i\right]^T$ when projected onto a two-dimensional plane. The linkage parameters to optimize can be gathered into a vector $x = [r_1 \; r_2 \; r_3 \; r_4 \; r_{Cx} \; r_{Cy} \; x_0 \; y_0 \; \theta_0]$. The components from r_1 to r_4 refer to the length of the different bars. Namely, r_1 is that of the fixed link, r_2 is that of the crank, r_3 is that of the coupler link containing point C, at distance $\{r_{Cx}, r_{Cy}\}$ from its origin, and r_4 is that of the rocker link. These variables are in a local coordinate system, O_r, which is fixed to the rotation center of the crank. Hence, It is possible



(a) Virtual prototype of a machine tool

(b) Mechanism schematic

Figure 1: Four-bar cutting machine and its corresponding kinematic diagram.

to locate the coupler point in a global coordinate system, O, through x_0, y_0, and θ_0. Proceeding in this way, one can compare the positions of the points defining the desired path to those reached by the coupler point for a given sequence of turns performed by the crank. These coordinates can be expressed as follows:

$$C_{Xr} = r_2 \cos\theta_2 + r_{Cx} \cos\theta_3 - r_{Cy} \sin\theta_3 \tag{1}$$

$$C_{Yr} = r_2 \sin\theta_2 + r_{Cx} \sin\theta_3 - r_{Cy} \cos\theta_3 \tag{2}$$

They can be related to the global coordinate system, O, as follows:

$$\begin{bmatrix} C_X \\ C_Y \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 \\ \sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} C_{Xr} \\ C_{Yr} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3}$$

The four-bar mechanism has a single degree of freedom that corresponds to the input motion provided by the crank link, θ_2. Hence, it is possible to obtain θ_3 from the closed-loop equations defining the mechanism as follows:

$$\vec{r}_1 + \vec{r}_4 = \vec{r}_2 + \vec{r}_3 \tag{4}$$

Equation (4) can be expressed in polar notation as follows:

$$r_1 e^{j\theta_0} + r_4 e^{j\theta_4} = r_2 e^{j\theta_2} + r_3 e^{j\theta_3} \tag{5}$$

By applying Euler's formula to Eq. (5) and separating the real and imaginary parts, the following expression is obtained:

$$r_1 \cos\theta_0 + r_4 \cos\theta_4 = r_2 \cos\theta_2 + r_3 \cos\theta_3 \tag{6}$$

$$r_1 \sin\theta_0 + r_4 \sin\theta_4 = r_2 \sin\theta_2 + r_3 \sin\theta_3 \tag{7}$$

By setting $\theta_0 = 0$ in Freudensteins equation, it is possible to express θ_3 and θ_4 in terms of θ_2 as follows:

$$K_1 \cos\theta_4 - K_2 \cos\theta_2 + K_3 = \cos(\theta_2 - \theta_4) \tag{8}$$

$$K_1 \cos\theta_3 + K_4 \cos\theta_2 + K_5 = \cos(\theta_2 - \theta_3) \tag{9}$$

with K_1, K_2, K_3, K_4 and K_5 defined in the following way:

$$K_1 = \frac{r_1}{r_2} \tag{10}$$

$$K_2 = \frac{r_1}{r_4} \tag{11}$$

$$K_3 = \frac{r_2^2 - r_3^2 + r_4^2 + r_1^2}{2r_2r_4} \tag{12}$$

$$K_4 = \frac{r_1}{r_3} \tag{13}$$

$$K_5 = \frac{r_4^2 - r_1^2 - r_2^2 - r_3^2}{2r_2r_3} \tag{14}$$

Finally, for a linkage configuration as depicted in Figure 1b, the angles θ_4 and θ_3 can computed from Eq. (6) and Eq. (7) as follows:

$$\theta_4 = 2\tan^{-1}\left(\frac{-B - \sqrt{B^2 - 4AC}}{2A}\right) \tag{15}$$

$$\theta_3 = 2\tan^{-1}\left(\frac{-E - \sqrt{E^2 - 4DF}}{2D}\right) \tag{16}$$

with:

$$A = \cos\theta_2 - K_1 - K_2\cos\theta_2 + K_3 \tag{17}$$

$$B = -2\sin\theta_2 \tag{18}$$

$$C = K_1 - (K_2 + 1)\cos\theta_2 + K_3 \tag{19}$$

and with:

$$D = \cos\theta_2 - K_1 + K_4\cos\theta_2 + K_5 \tag{20}$$

$$E = -2\sin\theta_2 \tag{21}$$

$$F = K_1 - (K_4 - 1)\cos\theta_2 + K_5. \tag{22}$$

2.2. Problem variants

There exist three types of problems depending on the relation between the crank angles and the points described by the coupler point, C:

- **Prescribed timing (type 1):** For $i = 1, .., N$, where N is the number of points defining the desired path, each point C^i must be achieved for a given angle of the crank, θ_2^i. The parameter vector to optimize only consists of the length of each bar along with the location and orientation of the coordinate system of the mechanism.
- **Prescribed timing with free starting angle (type 2):** This is a variant of the previous type in which the initial angle of the crank also has to be calculated. Hence, the parameter vector includes it as variable θ_2^1 along with the geometrical parameters of the linkage.
- **Non-prescribed timing (type 3):** In this case, each coupler point C^i of the curve can be achieved for any arbitrary angle of the crank θ_2^i. Thus, the parameter vector contains as many angles as reference points in the target path, in addition to the previous geometrical variables.

Depending on the cost function used to define the optimization problem, it is possible to distinguish between methods based on the absolute error and those linked to the shape error:

□ **Absolute-error-based approach:** This is the classical strategy for path synthesis. It relies on the Euclidean distance between the desired points and their equivalent ones achieved by the synthesized coupler to assess solutions. This method will be referred to as the absolute error cost function in what follows. The corresponding optimization problem can be formulated in the following way:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) = \sum_{i=0}^{N} \left[\left(C_X^i - C_{Xd}^i\right)^2 + \left(C_Y^i - C_{Yd}^i\right)^2 \right] \\ \text{subject to} \quad & l_b^i \leq x_i \leq u_b^i, \forall x_i \in x, \\ & r_1 + r_3 > r_2 + r_4, \\ & r_1 + r_4 > r_2 + r_3, \\ & r_3 + r_4 > r_1 + r_2 \end{aligned} \tag{23}$$

The first constraint requires every variable to be in its valid range defined by a lower (l_b) and an upper (u_b) bound. The others ensure that the resulting mechanism meets Grashof's law. Notice that the definition followed herein is more flexible than the one used in [8, 9] because the longest bar is not fixed, only the shortest one. Additionally, take into account that for the case of non-prescribed problems, the sequentiality of the crank displacement must be ensured by adding the following constraint:

$$\theta_2^j > \theta_2^{mod(j+1,N)} > ... > \theta_2^{mod(j+N,N)}, \theta_2^j = min\left\{\theta_2^i\right\}. \tag{24}$$

□ **Shape-error-based approach:** Instead of comparing coordinates, this approach focuses on measuring the similarity of the desired and the obtained curve by studying their intrinsic properties. Fourier's descriptors, turning functions, and the new shape codification method proposed in Section 3 can be used for this purpose. This approach increases the flexibility at search because the assessment is not biased by the necessity of matching coordinates, but it evaluates the underlying differences in shape. However, as it occurs with turning functions or the method proposed herein, shape codification strategies achieve flexibility by becoming invariant to changes in scale, rotation, and translation of the curves. Thus, it is ultimately necessary to transform any configuration found to make the output of the corresponding mechanism not only equivalent to the target in shape but also in size and orientation. Otherwise, the resulting mechanism might replicate the target shape, but it could not work in the same context.

Figure 2 summarizes the transformation procedure mentioned above. The perimeter (total length) of the path found through optimization might differ from the target one. Thus, it is necessary to start by scaling the bars of the corresponding mechanism by the ratio of both perimeters. After that, a process that tries to superimpose the desired path and the one obtained with minimal error starts. It iteratively seeks the most appropriate rotation angle θ_0 and translation of the synthesized mechanism. For every angle considered, the translation is performed by defining the coordinates x_0 and y_0. They are the difference between the X and Y coordinates of the centroids of the desired path and the one achieved, (G_{Xd}, G_{Yd}) and (G_X, G_Y), respectively. Mathematically, these values are computed as follows:

$$x_0 = G_{Xd} - G_X, \; y_0 = G_{Yd} - G_Y \tag{25}$$

with:

$$\{G_{Xd}, G_{Yd}\} = \left\{ \frac{\sum_{i=0}^{N} C_{Xd}^i}{N}, \frac{\sum_{i=0}^{N} C_{Yd}^i}{N} \right\}, \; \{G_X, G_Y\} = \left\{ \frac{\sum_{i=0}^{N} C_X^i}{N}, \frac{\sum_{i=0}^{N} C_Y^i}{N} \right\} \tag{26}$$

Finally, for closed paths like the one shown in Figure 2, it is also necessary to find the most appropriate starting point to compare them from a common origin. This is achieved by the reordering stage, which involves shifting the array with the crank angles and evaluate the absolute error to select the best option.

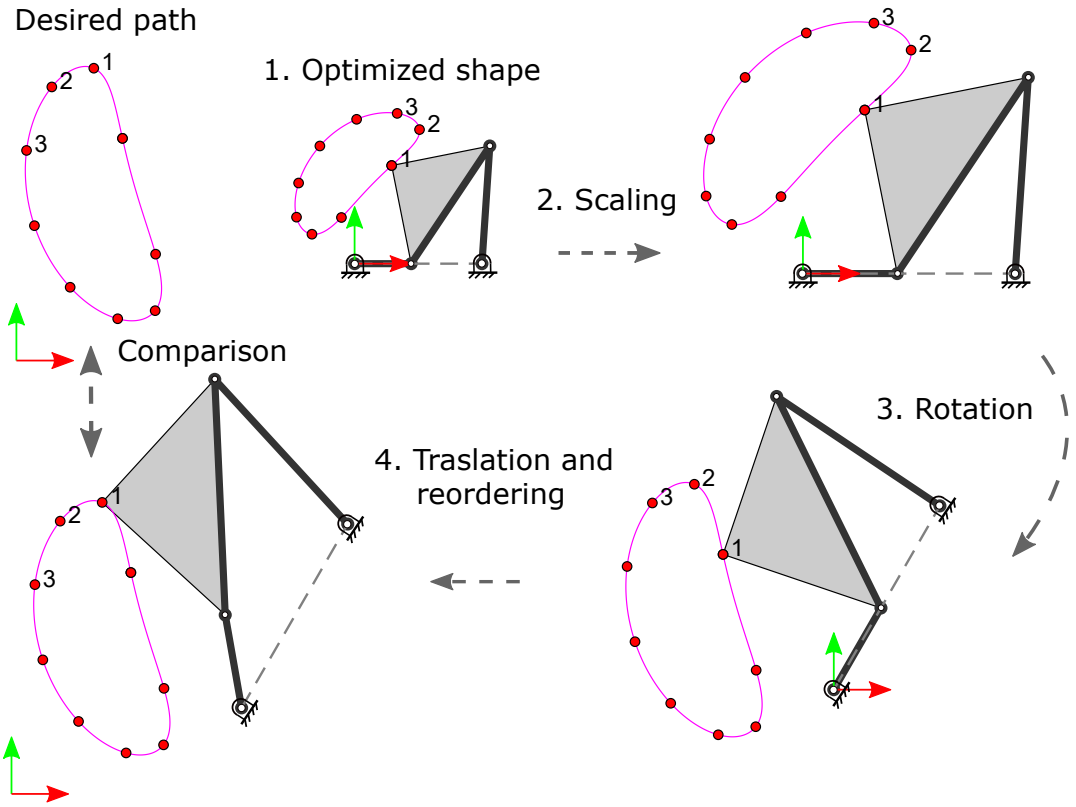


Figure 2: Transformations after shape-based optimization for direct comparison of the desired and optimized path.

Having described the variants of the problem supported by the software tool implemented, Table 1 summarizes the variables considered in each one of them. As can be seen, x_0, y_0 and θ_0 appear in every absolute-error-based problem. However, this is not the case for shape-error-based ones since these variables are computed through the transformation process explained above. Therefore, this approach involves less optimization variables to handle.

Table 1: Variables involved in each problem type and method

	Absolute-error-based approach	**Shape-based approach**	
Type	Optimization variables	Optimization variables	Post-processed variables
1	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}\ x_0\ y_0\ \theta_0]$	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}]$	$[x_0\ y_0\ \theta_0]$
2	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}\ \theta_2^1\ x_0\ y_0\ \theta_0]$	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}\ \theta_2^1]$	$[x_0\ y_0\ \theta_0]$
3	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}\ \theta_2^1 ... \theta_2^N\ x_0\ y_0\ \theta_0]$	$[r_1\ r_2\ r_3\ r_4\ r_{cx}\ r_{cy}\ \theta_2^1 ... \theta_2^N]$	$[x_0\ y_0\ \theta_0]$

3. Normalized shape-descriptor vectors

3.1. Overview

As introduced, the tool presented in this work also includes an ad-hoc method to represent paths without being affected by translation, rotation, and scale changes. The approach, known as normalized shape-descriptor vector(s) (NSDV(s)), is explained in this section.

3.2. Definition

Let P be an N-vertex simple polygon, i.e., a single region without self-intersections and either convex or concave. Its corresponding NSDV, v_P, is a description of its shape in $2N$ components. It would allow anyone to reproduce P in an arbitrary 2D-Cartesian coordinate system at any scale. The meaning of every

component depends on its position: For any odd i in $1, \ldots, 2N$, v_{P_i} refers to the length of a certain side divided by the longest one, L. However, for any even i, v_{P_i} contains the interior angle of P defined by the sides of $v_{P_{i-1}}$ and $v_{P_{i+1}}$, divided by 360°, and with $v_{Pi+1} = v_1$ for $i = 2N$. Thus, every component v_{P_i} is in $[0, 1] \subseteq \mathbb{R}$, and the use of local angles and normalized lengths make them independent of the size and orientation of P.

By definition, v_P describes the shape of P counterclockwise, starting at the longest side. Accordingly, every v_{P_1} will always be $\frac{L}{L} = 1$. The starting side is the one that results in the highest initial sequence of values, which covers the possibility of more than a single side of length L. Every P must have a single corresponding v_P based on its shape, regardless of how its information (vertices) is stored or modified (e.g., scale changes). This aspect is crucial for using NSDVs to seek and compare polygons. In fact, if v_P is used to reproduce P, the scale will depend on the length assigned to the first component. Similarly, its position and orientation will vary with the coordinate system defined. Hence, it is impossible to replicate the original P from v_P unless the initial longest side is known in position and length.

3.3. Computation procedure

Algorithm 1 describes the main steps to compute the NSDV of a simple polygon, P, with N vertices. P is defined as an ordered list of the X and Y coordinates of each vertex in an arbitrary 2D-Cartesian coordinate system, i.e., $P = \{P_1, \cdots, P_N\} = \{P_{1_x}, P_{1_y}, \cdots, P_{N_x}, P_{N_y}\}$. The last vertex is assumed to form a side with the first one.

Algorithm 1: Pseudo-code to compute the NSDV of a given polygon

Input: Polygon $P = (P_{1_x}, P_{1_y}, \cdots, P_{N_x}, P_{N_y}) \in \mathbb{R}^{2N}$
Output: NSD Vector $v_P \in [0, 1]^{2N}$

```
1  if IsClockWiselyOrdered(P) then
2  |  P = ReversePolygon(P);
3  end
4  Real L, Integer focus = SeekLongestSide(P);
5  v_P = CreateVector(Values = 0, Length = 2N);
6  2D-Point A = (P_focus, P_focus+1);                    /* 1st vrt. of the lgst. side */
7  2D-Point B = (P_focus+2, P_focus+3);                  /* Go to P_1 if focus = 2N − 1 */
8  for i = 1 to 2N do
9  |  if i%2 == 1 then
10 |  |  Side AB = (B−A)/||B−A||;
11 |  |  v_Pi = ||B−A||/L;
12 |  |  focus = focus + 2;                              /* Wrap-around if needed */
13 |  |  F = A;                                  /* Tracking the point for concavity check */
14 |  |  A = B;
15 |  |  B = (P_focus+2, P_focus+3);                     /* Wrap-around if needed */
16 |  else
17 |  |  Side CD = (A−B)/||A−B||;                        /* Origin adaptation */
18 |  |  Real angle = cos^-1 (AB · CD);
19 |  |  if CheckLocalConcavity(F, A, B) then
20 |  |  |  angle = (360° − angle);
21 |  |  end
22 |  |  v_Pi = angle/360;
23 |  end
24 end
25 return v_P = StandarizeOrder(v_P)
```

As shown between lines 1-3, the procedure starts by detecting whether the list of points is in clockwise order or not. If so, it inverts the order by using the '*ReversePolygon*' function, which is expected to transform the list from clockwise to counterclockwise order. A valid option is to swap the last vertex and the first one, the penultimate and the second one, and so on. The orientation detection applies Gauss's area formula, also

known as the 'shoelace' method [29, 30]. It allows computing the signed area, A, of an arbitrary simple 2D polygon according to Eq. (27), which assumes that the last vertex of the list is linked to the first one. Namely, if A is positive, the points are listed in counterclockwise order, while it is negative otherwise.

$$A = \frac{1}{2}\sum_{i=1}^{N}\left(P_{i_x}P_{i+1_y} - P_{i+1_x}P_{i_y}\right) \tag{27}$$

In line 4, the longest side of P is sought by using the '*SeekLongestSide*' function. It goes through every vertex of the list and computes the length of the side that it forms with the next one. This function must return both the length of the longest side, L, and the position of the x coordinate of its leftmost vertex, $focus$. If there is more than a side of length L, the result can be any of them (e.g., the last found in the implementation proposed). This position will define the initial order of the NSDV, v_P, which must be standardized, as explained later in this section. The vector is first initialized to 0 and length $2N$ in line 5.

In lines 6 and 7, the two vertices of the longest side are loaded. They are stored in variables A and B, respectively. Notice that reading the next vertex may require wrapping around to the first point of the list, as commented in line 6. This possibility must be considered every time the focus changes and its next point has to be read. Storing the first side prepares the execution of the main loop in line 8. This loop performs two main tasks: At odd iterations, it analyzes the current side to store its normalized length and moves the focus to the next one. At even iterations, it uses the former next side to compute the angle that it forms with the previous one.

The actions linked to the first task occur between lines 10 and 15. v_P is updated in line 11 by saving the result of dividing the length of the current side, whose unitary vector was computed in line 10, by that of the longest one, L. After that, the polygon focus is moved by setting $focus = focus + 2$, which goes from the x component of point i to that of $i + 1$ (and may require wrapping around, as introduced). In line 13, the first vertex of the current side is stored in F. This variable will serve to check the concavity of the interior angle defined by the current side and the following, as explained below. Finally, the vertices of the contiguous side replace those of the current one by setting $A = B$ (line 14) and B to its next one. Wrapping around is necessary when the new A comes from the last two components of P.

The actions of the second task start in line 17 with the computation of $\overrightarrow{CD}$. It is the unitary vector of the side after the one used to obtain the last normalized length in v_P (stored at the end of the previous set of steps). Notice that its direction is opposite to that of $\overrightarrow{AB}$. By proceeding this way, the arc-cosine of the dot product of $\overrightarrow{AB}$ and $\overrightarrow{CD}$, computed in line 18, is equal to the interior angle formed by both sides when P is convex. If so, $angle$ is not further modified and is used to compute the normalized angle as $angle/360$ in line 22. However, the local concavity between the two sides involved is checked in line 19. This analysis may require setting $angle = 360 - angle$ (line 20).

The local concavity check studies the three vertices that define the angle between both sides involved, i.e., the last one used to compute a particular length component of v_P and its next side. According to Algorithm 1, the vertices are F (i.e., A of the first side), A (i.e., the first vertex of the next side or B of the first side) and B (i.e., the last vertex of the next side). They are used to define the orientation matrix O in Eq. (28). Since P must be counterclockwise, the determinant of O will be negative when the studied interior angle is concave. Analogously, this value will be positive for convex angles, and it will be 0 if the studied points are collinear.

$$O = \begin{pmatrix} 1 & x_F & y_F \\ 1 & x_A & y_A \\ 1 & x_B & y_B \end{pmatrix} \tag{28}$$

Algorithm 1 ends with the re-ordering of v_P to return it in line 25. As introduced, NSDVs are required to start with the highest sequence of values so that the result does not depend on the longest side selected when there are more than a single one of length L. Figure 3 presents an imaginary situation in which it is necessary to adapt the order of an NSDV and how it is accomplished. As shown, Algorithm 1 could return any of the three alternatives as its result before the standardization in line 25, which must ultimately choose the last option.

For this purpose, the '*StandarizeOrder*' function proceeds as follows: It identifies every component $v_{P_i} = 1$ for $i = 1, 3, \cdots, 2N - 1$, i.e., the odd positions, which refer to side lengths. Then, it reads the next value

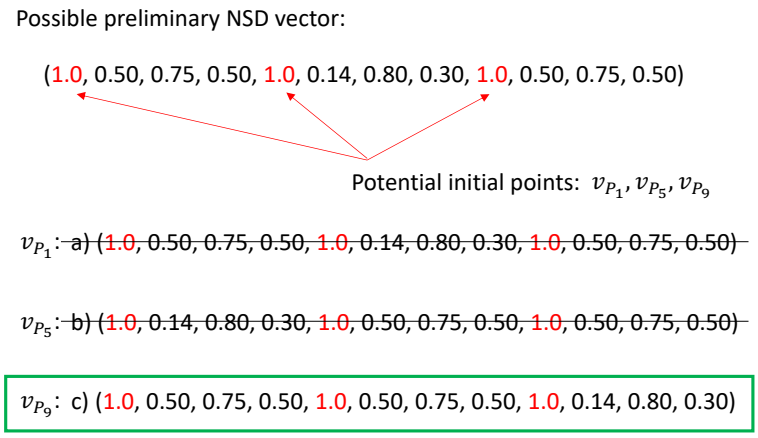


Figure 3: Example of NSDV with different possible starting points and the expected result of standardization.

of each selected position, and only those equal to the highest one of them keep as valid options. After that, the next value of each valid option is read, and only those equal to the maximum found are kept. This procedure, which might require wrapping around the vector when reading, is repeated until no more than a single option exists. It is also possible that the whole vector is iterated through with more than a single option remaining, which indicates shape symmetry. Finally, the vector is rewritten starting from the initial point selected. Regarding the example in Figure 3, v_{P_5} is discarded when 0.14 is found to be less than 0.5. Both v_{P_1} and v_{P_9} remain possible until the first path reaches 0.14 and the other one is at 0.5 (v_{P_2} after wrapping around).

Figure 4 shows the NSDV of a sample concave polygon. As can be seen, the same shape is presented in two different positions and sizes, but the resulting NSDV, which starts counterclockwise where highlighted by an arrow, does not vary.

Addressing path generation problems with NSDV requires defining the cost function as the Euclidean distance between the NSDV of the desired path and that achieved by any candidate solution. The lower the distance is, the better result it represents. Thus, it leads to a minimization problem as it occurs with the absolute-error-based approach.

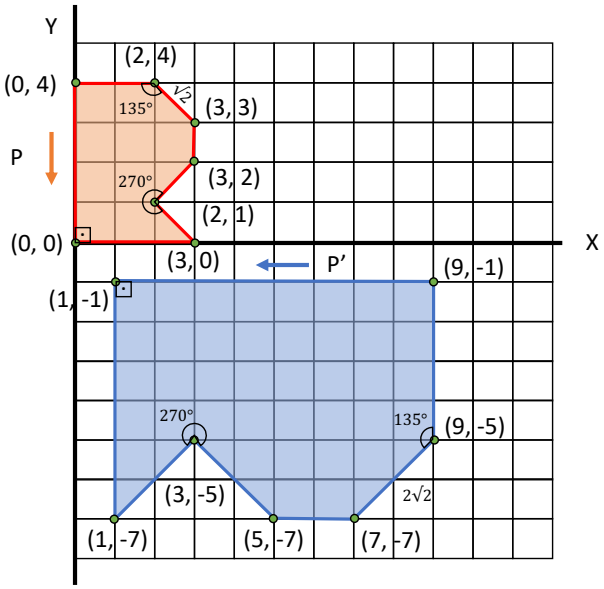
4. Optimization methods

Four state-of-the-art optimization methods have been considered to find the optimal variables for the mechanisms to be designed: i) Teaching-Learning-Based Optimization [31], ii) Differential Evolution [32], iii) Málaga University Mechanism Synthesis Algorithm [8], and iv) Interior-Point [33] (provided by MATLAB's FMinCon in its Optimization Toolbox [34]). The first three optimizers are stochastic population-based algorithms with global scope. The latter is a local deterministic method that has been combined with a multi-start component. The interested reader is invited to consult the references provided for further information. Nevertheless, this section briefly describes them for the sake of completeness. It is also relevant to highlight that users may add new optimizers since the tool is open-source.

4.1. Teaching-Learning-Based Optimization (TLBO)

TLBO, recently proposed by Rao et al. [31], is a numerical optimization algorithm that falls into the meta-heuristics category [35]. This aspect means that it applies a problem-independent strategy to find the extremes (optima) of a function. Namely, it simulates an academic context defined by a classroom of students who iteratively improve their skills. This situation is modeled by a pool (population) of randomly generated candidate solutions (individuals), which classifies this method as a stochastic population-based optimizer [35], as introduced. This algorithm has attracted considerable attention because of its simplicity, performance, and minimalist set of parameters [36].

TLBO takes as input the size of the population and the number of iterations to execute. It then generates a random set of individuals. Each of them is a vector in which the component i corresponds to the optimization



$$v_{P'} = \left(\frac{8}{8}, \frac{90}{360}, \ldots\right) = v_P = \left(\frac{4}{4}, \frac{90}{360}, \frac{3}{4}, \frac{45}{360}, \frac{\sqrt{2}}{4}, \frac{270}{360}, \frac{\sqrt{2}}{4}, \frac{135}{360}, \frac{1}{4}, \frac{135}{360}, \frac{\sqrt{2}}{4}, \frac{135}{360}, \frac{2}{4}, \frac{90}{360}\right)$$

Figure 4: Polygon in two different positions and sizes featuring the same NSDV.

variable i and must be within its valid range. All of them have to be evaluated by comparing the target path to the one produced by the mechanism resulting from the evaluated configuration. Thus, assessing an individual implies simulating the corresponding mechanism and computing one of the error metrics previously explained, which becomes the objective function in Optimization terms. After this initialization stage, TLBO executes the user-given number of iterations. Each of them starts with the teacher phase and ends with the learner one.

The teacher phase aims to improve the solutions by trying to shift each student towards the best one, which becomes the teacher, T. It starts by calculating a vector M with the mean of every component i from the individuals in the current population. After that, every student S is modified to obtain a variant S' according to Eq. (29), which is defined in terms of every component i. r_i is a random real number between 0 and 1 linked to component i. T_F, referred to as the 'teaching factor', is a random integer that can be either 1 or 2. Both r_i and T_F are globally computed for the current step. Finally, every S' is evaluated and replaces S if its error metric is better, while it is discarded otherwise.

$$S'_i = S_i + r_i \left(T_i - T_F M_i\right) \tag{29}$$

The learner phase simulates how students interact with each other to improve their skills. At this step, every student S is paired with any other different one, W. Again, a modified individual S' is generated and will replace S after the process if it has a better (lower) error value. However, every component i is computed according to Eq. (30) this time. r_i is a random real number between 0 and 1, linked to component i, and globally computed for the current step. As can be seen, movements go towards the best individual solutions.

$$S'_i = \begin{cases} S_i + r_i \left(S_i - W_i\right) & \text{if } error(S) < error(W) \\ S_i + r_i \left(W_i - S_i\right) & \text{otherwise} \end{cases} \tag{30}$$

After executing all the iterations, TLBO returns the best solution in its population.

4.2. Differential Evolution (DE)

DE, proposed by Storn and Price in [32], is a prevalent method for numerical optimization [37]. Like TLBO, DE relies on a population of randomly generated candidate solutions that progressively converge to the optima of the studied function. Every candidate solution or individual is also a vector concatenating

every variable under optimization. However, its procedure is different after the initial generation of a solution pool of the constant size decided by the user, NP. Its terminology is closer to traditional Genetic Algorithms [38]. Namely, its iterations or generations consist of mutation, crossover, and selection steps.

Mutation defines, for each individual $S^j, j = 1, \cdots, NP$, a mutant vector, v_{S^j}, which is created according to Eq. (31). r_2 and r_3 are different random integer indices in the range $[1, \cdots, NP]$. r_1 can be either a random integer like the two previous ones or the index of the best candidate solution in the population. The former approach is called 'rand' strategy, while the latter is known as 'best'. Similarly, $S^{r_2} - S^{r_3}$ defines a single difference vector, but it is possible to use more. The most popular alternative combines two instead, which redefines the term multiplied by F as $S^{r_2} - S^{r_3} + S^{r_4} - S^{r_5}$. Finally, F is a real factor between 0 and 2 that controls the amplification of the differential variation. It can be a user-given constant for the whole process or be randomly redefined in the range $[0.5, 1]$ for either each generation or mutant vector. The latter strategy is called dither, and it improves the convergence rate for noisy objective functions.

$$v_{S^j} = S^{r_1} + F\left(S^{r_2} - S^{r_3}\right) \tag{31}$$

Crossover combines each candidate solution, S^j, with its associated mutant vector, v_{S^j}. Combination results in the trial vector $S^{j'}$, which can be seen as the descendant between the plain individual and its mutant vector in terms of genetic algorithms. It takes place according to Eq. (32), which describes how to compute any component i of the trial vector. As can be seen, the component i of the trial vector of S^j comes from its mutant one or remains unaltered. It depends on a real user-given parameter, $\mathrm{CR} \in [0, 1]$, known as the crossover rate or probability. It controls every selection with a real random number generator in the range $[0, 1]$. This method is called 'binomial' crossover because the number of selected locations follows the so-named distribution. However, notice that at least one of the components of the trial vector must come from the mutant one. For this purpose, a random index is set before the crossover starts. The imaginary function *chosen* models the equality test throughout the process.

$$S_i^{j'} = \begin{cases} v_{S_i^j} & \text{if } rand(i) \leq \mathrm{CR} \text{ or } chosen(i) \\ S_i^j & \text{otherwise} \end{cases} \tag{32}$$

Each generation ends with the evaluation of every trial vector to select the individuals of the next one. Every $S^{j'}$ that outperforms its progenitor, S^j, replaces it, while those that do not are discarded. After executing all the iterations, DE returns the best individual in the population.

4.3. Málaga University Mechanism Synthesis Algorithm (MUMSA)

MUMSA, a recent method proposed by Cabrera et al. in [8], can be seen as an extension of DE. According to its name, the algorithm is specially designed for mechanism synthesis. That is also its application scope in the later paper by the same authors in [10]. However, like the previous methods, MUMSA is a general-purpose optimizer in reality. It is based on a DE method with a single difference vector, 'best' selection strategy, and binomial crossover. In this context, MUMSA adds a new mutation procedure to crossover, while the rest of the process remains unaltered.

Mutation allows further modifications of every trial vector according to Eq. (33). As can be seen, each component of the trial vector will be randomly moved in $\pm$ *range* around its current value with probability MP. This modification is independent of whether the component comes from the current individual or its mutant vector. Therefore, *range* and MP are MUMSA-specific parameters added to those inherited from DE.

$$S_i^{j'} = \begin{cases} rand() \in \left[S_i^{j'} \pm \text{ range}\right] & \text{if } rand(i) < MP \\ S_i^{j'} & \text{otherwise} \end{cases} \tag{33}$$

This addition is an extra in-breadth search component that increases variability (like the standard 'mutation' phase of traditional genetic algorithms [35]). It aims to attenuate a potential problem of plain DE called 'stagnation' [39], which complicates escaping from local optima, especially when the population size is small. In this situation, the number of candidate solutions that DE can explore is too small because it ultimately depends on the components that it can interchange from the initial population.

4.4. FMinCon's Interior-Point (FMC)

Interior-point methods refer to a widespread type of algorithm for solving linear and non-linear optimization problems [33]. A defining characteristic of an interior-point algorithm is that its iterations stay in the feasible region of the search space by using different techniques. As introduced, the interior-point method used in this work is the one shipped with the Optimization Toolbox of MATLAB in its solver FMinCon (FMC). This algorithm addresses the original optimization problem by solving a sequence of approximate ones resulting from adding slack variables and a barrier function [33, 34]. The approximations are less difficult to solve than the underlying problem.

FMC starts exploring the search space at a given initial point that can significantly affect the quality of the result. At each iteration, it takes one of two kinds of steps to solve the corresponding approximate problem [34]. The first one is a direct or Newton step that applies a linear approximation. The alternative is a conjugate gradient step using a trust region. By default, FMC first opts for the direct one. If that is not possible, for instance, because the approximate problem is not locally convex near the current position, it chooses the other method.

This kind of method is known to be optimal for convex problems and may return local solutions otherwise. Up to the authors' knowledge, the convexity of the problem at hand has not been demonstrated, and the results vary with the initial point at experimentation. For this reason, FMC is included in a generic multi-start procedure [35]. Accordingly, the local optimization is launched from a user-given number of different initial points to return the best final solution. In general, they are randomly generated, but the first one can be specified by the user. By proceeding this way, the exploration quality improves, and so does the probability of achieving the global optimum if the selected initial point is in its region.

5. Results

5.1. The SALAR mechanism synthesizer

The proposed software tool, which is called SALAR Mechanism Synthesizer, has been developed in MATLAB. It provides a graphical user interface (GUI) that allows users to input the required information, such as the desired curve, the type of problem, the design constraints, and the configuration of the selected optimization methods. These data can be introduced using the editable fields of the GUI or loaded from an appropriate '**.mat*' file.

The tool supports all the processes explained in the previous sections. Namely, it can use either the absolute or the shape (NSDV-based) error cost function to launch any of the optimizers described in Section 4 (notice that FMC requires a valid license of the MATLAB's Optimization Toolbox). The calculation of turning distances is partially supported, that is, to study the results, but not for optimization. The tool also includes the required post-processing steps for the results. After obtaining a solution, it is possible to compare the obtained curve to the desired one by inspecting their coordinates, observing the graphics, and studying the error metrics provided. The input information and the results can be saved to a '**.mat*' file for later use. For instance, FMC lets the user start from an existing solution to try to improve it. This option allows hybridizing global procedures with this local method, which is similar to what is done by Sedano et al. [20]. Furthermore, SALAR includes simple yet convenient capabilities of performing kinematic simulations with the results achieved.

Figure 5 contains an image of the GUI. The software can be publicly downloaded from (*Not available until acceptance. Attached copy to the reviewers*). The referred repository contains both the source code and the standalone version, which can be used by simply installing the MATLAB Runtime Environment. It also includes the documentation of the tool and the configuration files used for the experimentation.

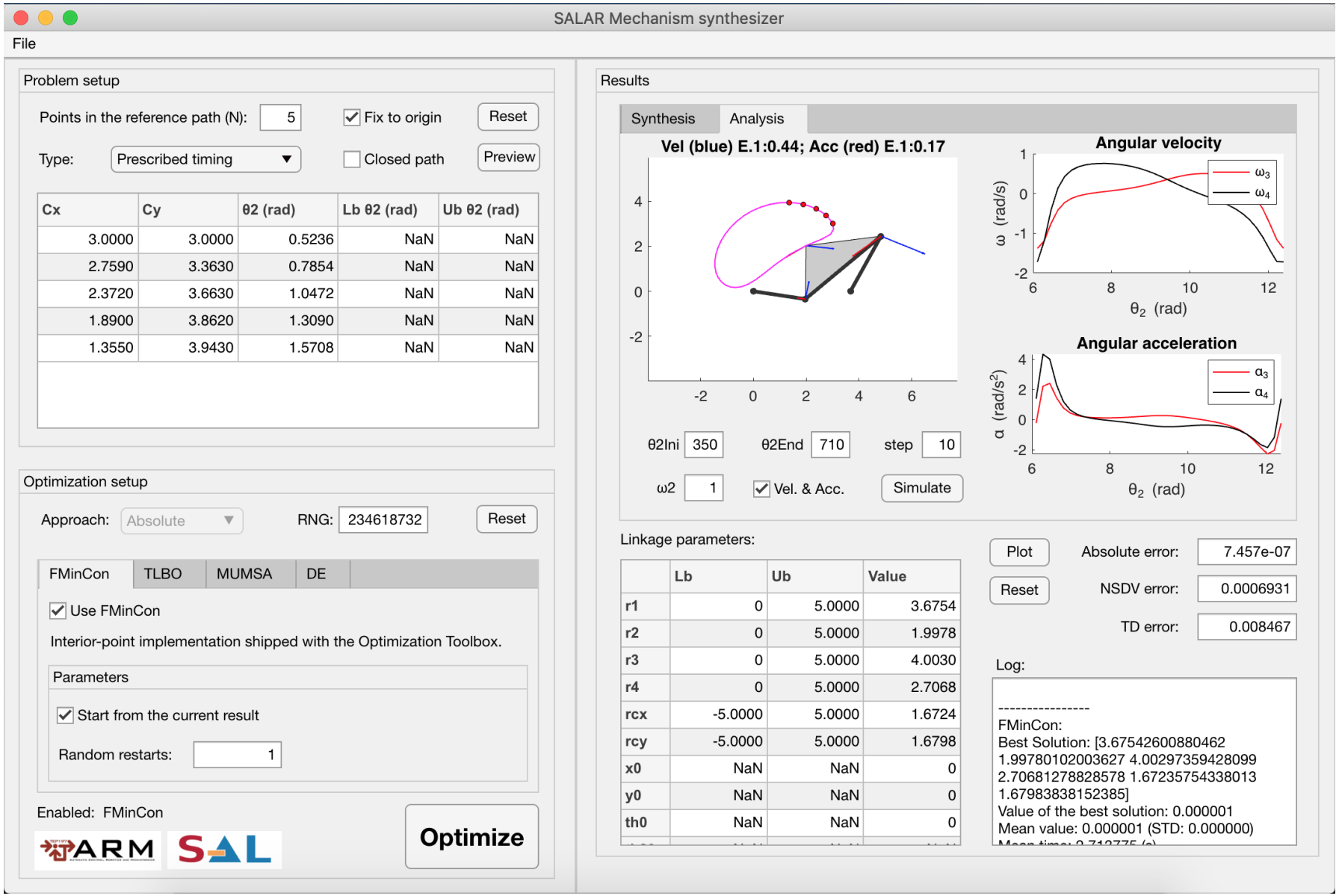


Figure 5: Screenshot of the proposed software GUI, denominated SALAR Mechanism Synthesizer

5.2. Numerical examples and tests

SALAR has been tested with six different problems from the literature, including a comparison with the results obtained by other authors. The following subsections contain the problem statement and the comparative analysis for each one. Notice that all the angles are in radians, while length units are omitted because they depend on the implementation size of the mechanism. The execution platform is MATLAB 2020a running in Microsoft Windows 10 on an Intel Core i7-3770 CPU at 3.40 GHz.

5.2.1. Configuration guidelines

Since there is no way to know the most appropriate cost function in general, each problem has been solved with the absolute and the NSDV error cost functions independently. First, all the available optimizers were enabled without initial information and under competition. After that, all the optimizers except FMC were disabled. This method was then reconfigured to take the former solution as its initial point to try to improve its quality. This strategy allows implementing a hybrid optimizer combining an evolutionary method with a local one, as suggested in [20].

The methods have been configured for each problem independently, and the corresponding files can be found alongside the tool. The configuration strategy is as follows: When FMC starts from an existing result, it has been launched once. This approach is suitable when a promising solution is known in advance or to refine a previous one. Otherwise, the number of random starts is iteratively increased until achieving good results for the target problem.

Regarding TLBO, its base configuration starts with 100 individuals and 50 cycles. In this context, increasing the population size allowed ensuring finding feasible solutions, while doing so with the number of cycles lead to improving their final quality when needed.

Concerning DE, the configuration process considers some of the guidelines provided by its authors. Namely, the number of individuals starts as ten times that of variables. The mutation is binomial and uses a single difference vector with random selection. In general, the cross rate and F have been set to 0.9 and 0.8, respectively, but per-generation dither was used when there were convergence problems. As done with TLBO, the lack of feasible solutions has been addressed with a larger population and low quality with more cycles.

In relation to MUMSA, it inherits the main guidelines from DE. However, its F factor is 0.6 in general (and lower for the hardest cases). Its mutation probability has been set to values significantly lower than the cross rate, i.e., 0.1 (and slightly raised, e.g., to 0.2, for the most difficult cases to enhance exploration capabilities).

Additionally, notice that all the random initial points of FMC are filtered to ensure that the shortest bar is the one expected by the simulator and that the angles fulfill the sequence requirements when needed. For the population-based methods, 10% of their initial and random solutions are filtered in the same way to provide the population with some problem-specific knowledge and improve the convergence rates. SALAR performs these tasks autonomously.

Each population-based method has been executed five times independently to store the best result, which is also natively supported by SALAR. This option allows checking the means of error and the standard deviations, which might be relevant for stochastic methods.

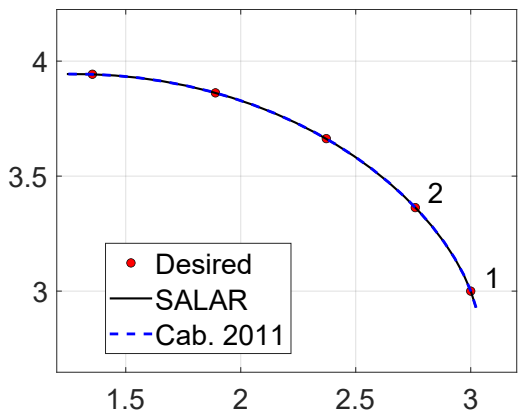
5.2.2. Problem 1.1

This is a type 1 problem that was first formulated and solved in [40]. The solution was later improved by Cabrera et al. [8]. In this problem, the coupler point has to describe an open curve (C^i_{des}) for a given sequence of angles, θ^i_2:

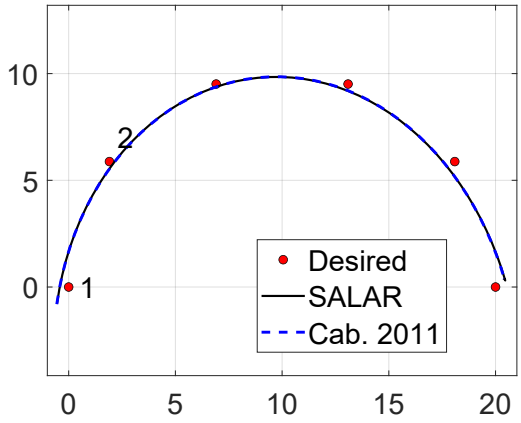
$$C^i_d = \{(3,3),(2.759,3.363),(1.890,3.862),(2.372,3.663),(1.890,3.862),(1.355,3.943)\}$$
$$\theta^i_2 = \left\{\theta^1_2,\theta^2_2,\theta^3_2,\theta^4_2,\theta^5_2\right\} = \{\pi/6,\pi/4,\pi/3,5\pi/12,\pi/2\}$$

The variables to optimize must be in the following ranges:

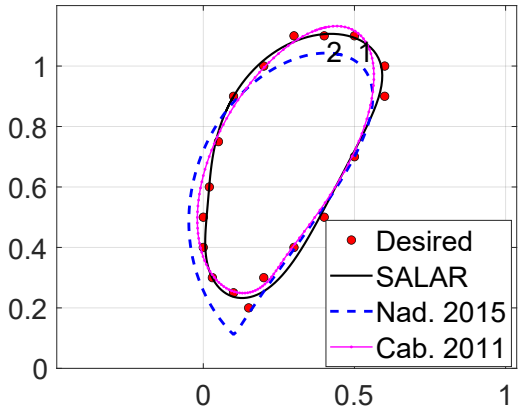
$$r_1,r_2,r_3,r_4 \in (0,5]$$
$$r_{cx},r_{cy} \in [-5,5]$$



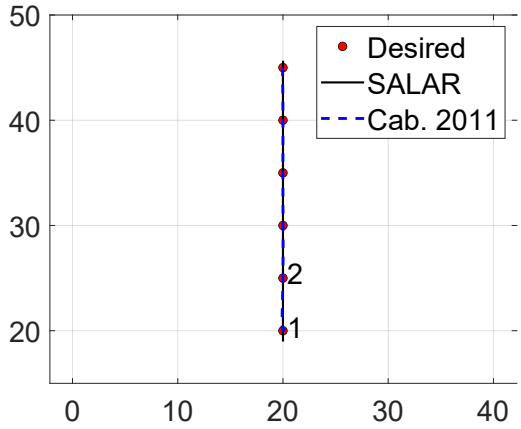
(a) Problem 1.1



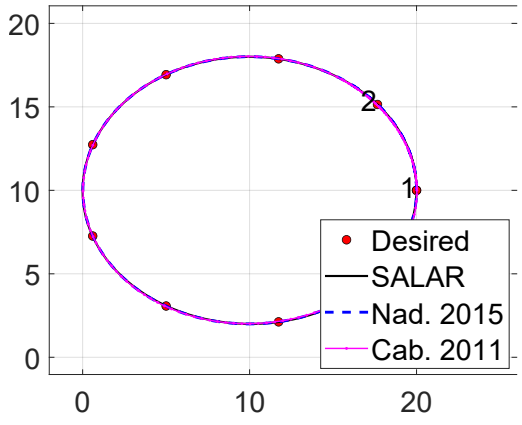
(b) Problem 1.2



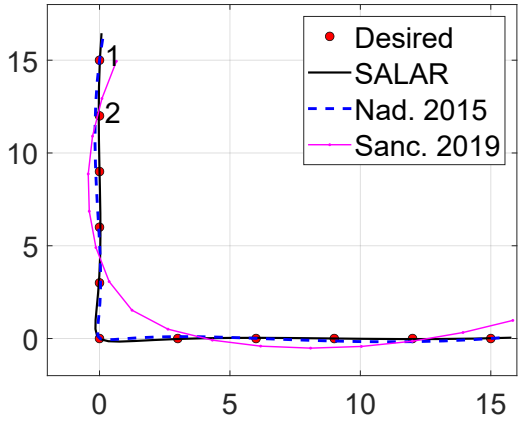
(c) Problem 2.1



(d) Problem 3.1



(e) Problem 3.2



(f) Problem 3.3

Figure 6: Coupler curves of the selected problems. The points define the desired path, and the curves represent that achieved by the optimized mechanism. In each case, number one shows the initial point, and number two indicates the sense.

The best solution provided by SALAR was achieved with FMC, using the absolute error cost function, and configured to perform 10 random restarts. The absolute error of the configuration found is 7.790×10^{-7}. According to Table 2, this value is better (lower) than the one achieved in [8], which is 1.768×10^{-6}. Regardless, both are very accurate solutions, as shown in Figure 6a. For the sake of completeness, Table 2 shows the linkage parameters that correspond to the solution achieved and the error values of NSDV and turning distance (TD). However, the NSDV cost function is not available for optimization because this is a fixed-to-origin problem.

Table 2: Solution for each problem and the corresponding metrics: Absolute error (AE), normalized shape-descriptor vector (NSDV), and turning distance (TD).

		r_1	r_2	r_3	r_4	r_{cx}	r_{cy}	x_0	y_0	θ_0	AE	NSDV	TD
1.1	Cabrera 2011 [8]	3.77	2.00	4.12	2.75	1.68	1.67	0.00	0.00	0.00	1.768×10^{-6}	-	-
	SALAR	3.56	2.00	3.87	2.67	1.67	1.69	0.00	0.00	0.00	7.790×10^{-7}	7.200×10^{-4}	7.359×10^{-3}
1.2	Cabrera 2011 [8]	50.00	1.35	1.35	50.00	11.38	4.44	10.19	-3.69	6.22	1.216	-	-
	SALAR	49.75	1.34	1.34	49.75	11.45	4.26	10.22	-3.68	6.23	1.219	7.259×10^{-2}	2.489×10^{-1}
2.1	Cabrera 2011 [8]	4.45	0.30	3.91	0.85	-2.07	1.66	-1.31	2.81	2.74	1.960×10^{-2}	-	2.615×10^{-1}
	Nadal 2015 [10]	2.46	0.24	0.94	3.16	0.03	0.48	0.57	0.35	4.79	1.135×10^{-1}	-	1.848×10^{-1}
	SALAR	1.05	0.42	0.91	0.60	0.37	0.40	0.27	0.15	0.28	9.026×10^{-3}	6.810×10^{-1}	2.377×10^{-1}
3.1	Cabrera 2011 [8]	31.79	8.20	24.93	31.39	34.19	14.42	-6.37	56.84	4.02	2.057×10^{-4}	-	-
	SALAR	30.42	9.50	28.30	40.12	43.02	2.79	-12.80	54.21	4.12	6.812×10^{-7}	7.779×10^{-5}	-
3.2	Cabrera 2011 [8]	79.52	9.72	45.84	51.43	8.21	-2.95	2.02	13.22	5.60	4.700×10^{-3}	-	1.795×10^{-2}
	Nadal 2015 [10]	560.2	8.97	406.1	337.9	13.19	-88.48	-50.43	75.94	-0.05	6.899×10^{-4}	-	1.481×10^{-2}
	SALAR	79.57	8.07	50.59	42.01	-10.72	-1.53	8.20	-0.65	3.93	4.121×10^{-4}	1.496×10^{-1}	5.861×10^{-3}
3.3	Nadal 2015 [10]	33.07	21.37	29.04	26.49	2.51	-9.73	25.69	18.10	3.81	1.099×10^{-1}	-	2.746×10^{-2}
	Sancibrian [21]	28.16	16.94	23.05	32.68	-0.42	-7.12	19.05	12.69	3.42	*	-	-
	SALAR	46.53	28.81	68.18	67.56	1.78	-14.49	33.90	27.11	3.83	6.062×10^{-3}	1.212×10^{-2}	3.530×10^{-2}

5.2.3. Problem 1.2

This is a type 1 problem that was first formulated and solved by Acharyya and Mandal [9]. The solution was later improved by Cabrera et al. [8]. In this problem, the coupler point must follow a semi-circular arc for a given sequence of angles. Whereas the path of the coupler must be described clockwise, the prescribed motion of the crank is counterclockwise, which increases the difficulty. The coupler points and the crank angles are the following:

$$C_d^i = \{(0,0), (1.9098, 5.8779), (6.9098, 9.5106), (13.09, 9.5106), (18.09, 5.8779), (20,0)\}$$
$$\theta_2^i = \left\{\theta_2^1, \theta_2^2, \theta_2^3, \theta_2^4, \theta_2^5\right\} = \{\pi/6, \pi/3, \pi/2, 2\pi/3, 5\pi/6, \pi\}$$

The variables to optimize must be in the following ranges:

$$r_1, r_2, r_3, r_4 \in (0, 50]$$
$$r_{cx}, r_{cy} \in [-50, 50]$$
$$\theta_0 \in [0, 2\pi]$$

According to Table 2, the best solution provided by SALAR is similar to that presented in [8]. It has been found by the MUMSA optimizer, using NSDV as the cost function, and configured with the following parameters:

NP : 80	CP : 0.90	MP : 0.10	$Range$: 0.50
F : 0.60	$Itermax$: 100	$Restarts$: 5	

This configuration leads to an absolute error of 1.295. After being refined by FMC, the error was later reduced to 1.219, while that presented by Cabrera et al. is 1.216 [8]. Figure 6b shows the coupler curve obtained by both approaches. As can be seen, their performance is very similar and acceptable, considering the challenging nature of the problem.

5.2.4. Problem 2.1

This is a type 2 problem that was first formulated and solved by Kunjur and Krishnamurty [40]. Their solution was later improved in [8, 10]. In this problem, the coupler point must describe a closed curve, counterclockwise, for a given sequence of angles. It has to start at an arbitrary angle, θ_2^1. The coupler points and crank angles are the following:

$$\begin{gathered} C_d^i = \{(0.5, 1.1), (0.4, 1.1), (0.3, 1.1), (0.2, 1.0), (0.1, 0.9), (0.05, 0.75), (0.02, 0.6), (0, 0.5), \\ (0, 0.4), (0.03, 0.3), (0.1, 0.25), (0.15, 0.2), (0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.5, 0.7), (0.6, 0.9), (0.61)\} \\ \theta_2^i = \left\{\theta_2^1, \theta_2^1 + (\pi/9)i\right\}, i : 1..17 \end{gathered}$$

The variables to optimize must be in the following ranges:

$$\begin{aligned} r_1, r_2, r_3, r_4 &\in (0, 50] \\ r_{cx}, r_{cy} &\in [-50, 50] \\ \theta_0, \theta_2^1 &\in [0, 2\pi] \end{aligned}$$

The best solution provided by SALAR is achieved with MUMSA and NSDV as the cost function. This solution leads to an absolute error of 4.800×10^{-2} using the following parameters:

$NP : 100$	$CP : 0.90$	$MP : 0.10$	$Range : 0.50$
$F : 0.60$	$Itermax : 120$	$Restarts : 5$	

After refining it, the absolute error is reduced to 9.026×10^{-3}. Table 2 shows this solution and those achieved in [8, 10]. The linkage parameters found with SALAR include an initial crank angle $\theta_2^1 = 1.938$. As can be seen, this solution is of high quality in terms of absolute error and shape similarity. Figure 6c allows comparing these results visually.

5.2.5. Problem 3.1

This is a type 3 problem that was first formulated and solved by Cabrera et al. [7]. The results were later improved in [8]. In this problem, the desired path is a straight line traversed by the coupler point upwards (C_{des}^i), for any arbitrary sequence of angles θ_2^i. The points of the desired path are the following:

$$C_d^i = \{(20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45)\}$$

The variables to optimize must be in the following ranges:

$$\begin{aligned} r_1, r_2, r_3, r_4 &\in (0, 60] \\ r_{cx}, r_{cy} &\in [-60, 60] \\ \theta_0, \theta_2^1, ..., \theta_2^6 &\in [0, 2\pi] \end{aligned}$$

The best solution returned by SALAR results from using FMC and the absolute error as its cost function. It is achieved without initial conditions after 300 random restarts. The refinement stage reduces the error to 6.812×10^{-7}. Table 2 shows the variables found and the associated errors. It also includes the solution obtained in [8], which leads to an absolute error of 2.057×10^{-4}. Figure 6d shows both. As can be seen, they meet the requirements accurately. The crank angles of the solution obtained by SALAR are the following:

$$\theta_2^i = \left\{\theta_2^1, \theta_2^2, \theta_2^3, \theta_2^4, \theta_2^5, \theta_2^6\right\} = \{2.3643, 2.808, 3.232, 3.665, 4.128, 4.6763\}$$

5.2.6. Problem 3.2

This a type 3 problem that was first formulated and solved by Acharyya and Mandal [9]. The solution was later improved by Cabrera et al. first [7], and by Nadal et al. later in [10]. In this problem, the coupler point must follow a counterclockwise elliptical path (C_{des}^i) for any arbitrary sequence of angles θ_2^i. The points of the target path are the following:

$$\begin{gathered} C_d^i = \{(20, 10), (17.66, 15.142), (11.736, 17.878), (5, 16.928), \\ (0.60307, 12.736), (0.60307, 7.2638), (5, 3.0718), (11.736, 2.1215), (17.66, 4.8577)\} \end{gathered} \tag{34}$$

The variables to optimize must be in the following ranges:

$$r_1, r_2, r_3, r_4 \in (0, 80]$$
$$r_{cx}, r_{cy} \in [-80, 80]$$
$$\theta_0, \theta_2^1, ..., \theta_2^9 \in [0, 2\pi]$$

The best solution provided by SALAR is obtained with FMC using the absolute error cost function and 1000 random restarts. After the refinement stage, the error of the solution is 4.121×10^{-4}. As shown in Table 2, it is better than those presented in [8, 10], which lead to absolute errors of 4.700×10^{-3} and 6.899×10^{-4}, respectively. Figure 6e confirms that all of them meet the requirements. The crank angles of the solution found with SALAR are the following:

$$\theta_2^i = \left\{\theta_2^1, \theta_2^2, ..., \theta_2^9\right\} = \{2.415, 3.122, 3.806, 4.489, 5.173, 5.866, 0.2827, 0.9890, 1.704\}$$

5.2.7. Problem 3.3

This problem was first formulated and solved by Sedano et al. [20] as a type 2 problem. Their solution was later improved in [10, 21]. However, the prescribed angles were not provided. Thus, the problem has been modeled as a type 3 instance herein, which increases the difficulty. In this case, the coupler point must follow an open and counterclockwise path (C_{des}^i) resulting in a right angle, for any arbitrary sequence of angles θ_2^i. The coordinates of the points defining the desired path are the following:

$$C_d^i = \{(0, 15), (0, 12), (0, 9), (0, 6), (0, 3), (0, 0),$$
$$(3, 0), (6, 0), (9, 0), (12, 0), (15, 0)\}$$

Despite being omitted in the original problem, the variables to optimize must be in the following ranges:

$$r_1, r_2, r_3, r_4 \in (0, 100]$$
$$r_{cx}, r_{cy} \in [-100, 100]$$
$$\theta_0, \theta_2^1, ..., \theta_2^{11} \in [0, 2\pi]$$

The best solution of SALAR is obtained by FMC using the absolute error as the cost function, without initial conditions, and with 1500 random restarts. After the refinement process, the absolute error is 6.062×10^{-3}. Table 2 shows the solutions achieved in [10, 21] together with the one obtained herein. As can be seen, the result of SALAR features less absolute error than the configuration presented in [10]. This value could not be compared to that published in [21], since different metrics are used. However, according to the graphical representation of the solutions shown in Figure 6f, the solution obtained by SALAR achieves the best performance. Its crank angles are listed below:

$$\theta_2^i = \left\{\theta_2^1, \theta_2^2, ..., \theta_2^{11}\right\} = \{5.538, 5.643, 5.752, 5.869, 6.008, 6.274, 0.258, 0.397, 0.514, 0.623, 0.728\}$$

6. Conclusions

This work has highlighted the relevance of four-bar mechanisms in industry and reviewed the main approaches to design them. After that, it has focused on the path synthesis design approach by describing the underlying problem formulation and presenting a new open-source MATLAB software package for this purpose. The referred problem is defined by the path that the coupler point of the mechanism has to follow and the constraints imposed on the bar lengths and angles.

The proposed tool, called SALAR Mechanism Synthesizer, supports three variants of this problem, depending on the relationship between the coupler points and the crank angles. The main advantages of this proposal are three key aspects: First, the tool includes four different optimizers that can compete and collaborate. Secondly, SALAR offers a novel cost function for shape-error-based problems, called normalized shape-descriptor vectors (NSDV), along with the well-known absolute error metric. Thirdly, it is an open-source tool that can be freely used, distributed, and modified. SALAR also features an intuitive graphical user interface that facilitates its usage. It allows checking the results obtained both graphically and numerically. Furthermore, SALAR includes an analysis section to perform kinematic simulations.

Six experiments from the literature have been conducted to assess the performance of the tool. In these experiments, the different optimizers and cost functions included in SALAR have been tested. The best solution obtained for each problem has been compared to the results presented in the corresponding reference papers. In all the cases, similar or even better results were achieved by SALAR in terms of absolute error. The comparisons were carried out numerically and graphically, and the configuration files are provided with the source code. Therefore, the main contribution of this work is a tool that supplies the lack of open-source software for path synthesis and solves problems from the literature, either with the same quality or even better. Besides, NSDV can be of general interest as a simple alternative to describe and compare paths representing polygons.

As future work, it is possible to consider the inclusion of new optimization algorithms and the support of other kinds of mechanisms. Similarly, the tool could be extended to handle motion and precision point synthesis problems.

Acknowledgments

This work has been supported by the Spanish Ministry of Economy and Competitiveness (RTI2018-095993-B-I00), Junta de Andalucía (P18-RT-1193), and the University of Almería (UAL18-TIC-A020-B).

References

[1] A. De-Juan, R. Sancibrian, F. Viadero, Optimal synthesis of function generation in steering linkages, International Journal of Automotive Technology 13 (7) (2012) 1033–1046.

[2] J. Mendoza, C. Montufar, J. Campos, Analytical synthesis for four-bar mechanisms used in a pseudo-equatorial solar tracker, Ingeniería e Investigación 33 (3) (2013) 55–60.

[3] R. Sodhi, K. Russell, Kinematic synthesis of planar four-bar mechanisms for multi-phase motion generation with tolerances, Mechanics Based Design of Structures and Machines 32 (2) (2004) 215–233.

[4] P. Chabphet, S. Santichatsak, T. N. Thalang, S. Sleesongsom, S. Bureerat, High-lift mechanism motion generation synthesis using a metaheuristic, Proceedings 39 (1) (2019) 5.

[5] T. Terefe, H. Lemu, A. Mariam, Review and synthesis of a walking machine (Robot) leg mechanism, in: MATEC Web of Conferences, Vol. 290, 2019.

[6] J. Sun, W. Liu, J. Chu, Dimensional Synthesis of Open Path Generator of Four-Bar Mechanisms Using the Haar Wavelet, Journal of Mechanical Design, Transactions of the ASME 137 (8) (2015) 1–8.

[7] J. Cabrera, A. Simon, M. Prado, Optimal synthesis of mechanisms with genetic algorithms, Mechanism and Machine Theory 37 (10) (2002) 1165–1177.

[8] J. Cabrera, A. Ortiz, F. Nadal, J. Castillo, An evolutionary algorithm for path synthesis of mechanisms, Mechanism and Machine Theory 46 (2) (2011) 127–141.

[9] S. Acharyya, M. Mandal, Performance of EAs for four-bar linkage synthesis, Mechanism and Machine Theory 44 (9) (2009) 1784–1794.

[10] F. Nadal, J. Cabrera, A. Bataller, J. Castillo, A. Ortiz, Turning functions in optimal synthesis of mechanisms, Journal of Mechanical Design 137 (6) (2015) 1–9.

[11] E. Arkin, L. Chew, D. Huttenlocher, K. Kedem, J. Mitchell, An efficiently computable metric for comparing polygonal shapes, in: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 1990, pp. 129–137.

[12] D. Aiordăchioaie, On estimation of the orientation of mobile robots using turning functions and SONAR information, The Annals of Dunarea de Jos, University of Galati. Fascicle III, Electrotechnics, Electronics, Automatic Control, Informatics 26 (2003) 61–66.

[13] E. McCreath, Partial matching of planar polygons under translation and rotation, in: Proceedings of the 20th Annual Canadian Conference on Computational Geometry, CCCG, 2008, pp. 1–4.

[14] C. Volotão, R. Santos, G. Erthal, L. Dutra, Shape characterization with turning functions, in: Proceedings of the 17th International Conference on Systems, Signals and Image Processing, Vol. 1, 2010, pp. 554–557.

[15] I. Ullah, S. Kota, Optimal synthesis of mechanisms for path generation using fourier descriptors and global search methods, Journal of Mechanical Design, Transactions of the ASME 119 (4) (1997) 504–510.

[16] N. Khan, I. Ullah, M. Al-Grafi, Dimensional synthesis of mechanical linkages using artificial neural networks and Fourier descriptors, Mech. Sci 6 (2015) 29–34.

[17] W. Lee, K. Russell, Developments in quantitative dimensional synthesis (1970present): four-bar path and function generation, Inverse Problems in Science and Engineering 26 (9) (2018) 1280–1304.

[18] S. Sleesongsom, S. Bureerat, Optimal synthesis of four-bar linkage path generation through evolutionary computation with a novel constraint handling technique, Computational Intelligence and Neuroscience 2018.

[19] A. Bataller, J. Cabrera, M. Clavijo, J. Castillo, Evolutionary synthesis of mechanisms applied to the design of an exoskeleton for finger rehabilitation, Mechanism and Machine Theory 105 (2016) 31–43.

[20] A. Sedano, R. Sancibrian, A. De-Juan, F. Viadero, F. Egaa, Hybrid optimization approach for the design of mechanisms using a new error estimator, Mathematical Problems in Engineering 2012.

[21] R. Sancibrian, A. Sedano, E. Sarabia, J. Blanco, Hybridizing differential evolution and local search optimization for dimensional synthesis of linkages, Mechanism and Machine Theory 140 (2019) 389–412.

[22] V. Goulet, W. Li, H. Cheong, F. Iorio, C.-G. Quimper, Four-bar linkage synthesis using non-convex optimization, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2016, pp. 618–635.

[23] R. Singh, H. Chaudhary, A. Singh, A novel gait-based synthesis procedure for the design of 4-bar exoskeleton with natural trajectories, Journal of Orthopedic Translation 12 (2018) 6–15.

[24] Info Key, ASOM: Analysis, synthesis and optimization of multi-bar linkages (2021).

[25] A. Rankers, H. Schrama, SAM - Simulation and analysis of mechanisms, in: Proceedings of the ASME Design Engineering Technical Conference, Vol. 5 B, American Society of Mechanical Engineers Digital Collection, 2002, pp. 1383–1387.

[26] A. Bataller, A. Ortiz, J. Cabrera, F. Nadal, WinMecC: Software for the analysis and synthesis of planar mechanisms, in: Mechanisms and Machine Science, Vol. 43, Kluwer Academic Publishers, 2017, pp. 233–242.

[27] Y. Hongying, T. Dewei, W. Zhixing, Study on a new computer path synthesis method of a four-bar linkage, Mechanism and Machine Theory 42 (4) (2007) 383–392.

[28] A. Purwar, S. Deshpande, Q. Ge, Motiongen: Interactive design and editing of planar four-bar motions for generating pose and geometric constraints, Journal of Mechanisms and Robotics 9 (2).

[29] A. Ramos, F. Ramos, Heliostat blocking and shadowing efficiency in the video-game era, arXiv preprint (2014) 1402.1690.

[30] J. Vince, Mathematics for computer graphics, Vol. 3, Springer, 2010.

[31] R. Rao, V. Savsani, D. Vakharia, Teaching–learning-based optimization: an optimization method for continuous non-linear large scale problems, Information Sciences 183 (1) (2012) 1–15.

[32] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[33] I. Griva, S. Nash, A. Sofer, Linear and nonlinear optimization, Vol. 108, Siam, 2009.

[34] M. Branch, A. Grace, MATLAB: Optimization Toolbox: User's Guide, Math Works, 2020.

[35] S. Salhi, Heuristic Search: The emerging science of problem solving, Springer, 2017.

[36] R. Xue, Z. Wu, A survey of application and classification on Teaching-Learning-Based Optimization algorithm, IEEE Access 8 (2020) 1062–1079.

[37] K. Price, R. Storn, J. Lampinen, Differential evolution: a practical approach to global optimization, Springer Science & Business Media, 2006.

[38] S. Sivanandam, S. Deepa, Genetic algorithms, in: Introduction to Genetic Algorithms, Springer, 2008, pp. 15–37.

[39] J. Lampinen, I. Zelinka, On stagnation of the differential evolution algorithm, in: Proceedings of MENDEL, 2000, pp. 76–83.

[40] A. Kunjur, S. Krishnamurty, Genetic Algorithms in Mechanical Synthesis, Journal of Applied Mechanisms and Robotics 4 (2) (1997) 18–24.