Exploring Inverse Kinematics

UNIVERSITY OF BATH

DEPARTMENT OF COMPUTER SCIENCE: CS50244
NAME: Jacob Haynes
CANDIDATE NUMBER: 02151
PROGRAM: EngD Digital Entertainment
21/11/2017

This paper details an exploratory investigation into inverse kinematics. A three hinge joint, two-dimensional system is modeled with each linkage being a simple rod. A damped pseudoinverse jacobian method is used with additional terms for simulating joint stiffness. The results are animated via interpolation techniques and a slow-in, slow-out function is applied to the animation. The mathematics and theory behind the methodology will be discussed in detail followed by demonstrations of the functions used. The results will be discussed as well as further developments and refinements that can be applied.

1 Introduction

Defined as the mathematical process of finding the movement of an object to a desired position, inverse kinematics is a powerful tool in robotics and animation. Unlike forward kinematics which define relations between joint angles of a skeleton to form a specified pose, inverse kinematics can be used to calculate the joint angles for a desired pose. The start and end positions can then be interpolated to form a smooth animated motion from one position to another.

Rigid skeletons consist of individual objects, links, connected together by joints. In this paper only hinge style joints will be considered but it is possible to also model rotational and translational joints via this method [1]. Each link has a joint location and an end effector, the end effector of the terminal link has an assigned target location which it will aim to move to. Simple skeletons can be solved analytically, however, more complex systems, such as the skeleton shown in figure 1, must be solved via a different approach. This paper will focus on Jacobian methods of computing this solution.

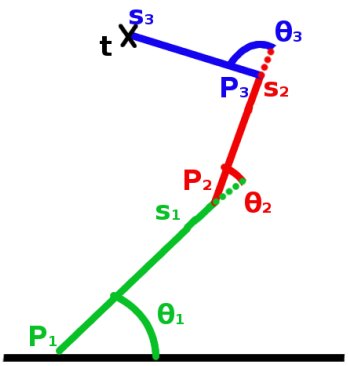


Figure 1: A schematic of a two dimensional, three hinge joint skeleton. The green, red and blue links are named (for the purpose of this paper) the upper arm, lower arm and hand respectively. Each link, of length L_i, has a joint, θ_i, located at point, P_i, and an end effector, s_i. The target position is labeled t

The system described in figure 1 will be modeled and explored in the following text, primarily by pseudoinverse Jacobian method. A derivation of the mathematics as well as a concise review into inverse kinematics methods will then be followed by experimental results of the created model. The paper will conclude with a discussion of the results as well as a discussion of other parameters which could be modeled

such as link flexibility.

2 Theory & Methods

2.1 The Jacobian Matrix

For complex skeletons the motion is incrementally calculated. Computation is performed change each joint angle to slowly move the end effector to the target location. One such way of achieving this is via a matrix of partial derivatives known as the Jacobian, this was first described computationally by Wolovich and Elliott [2]. Mathematically the Jacobian can be formed as follows. Consider i functions of i variables:

$$y_i = f_i(x_1, ..., x_i) \tag{1}$$

the differential of y can be written as

$$dy_i = \frac{\delta f_i}{\delta x_1} dx_1 + ... + \frac{\delta f_i}{\delta x_i} dx_i \tag{2}$$

applying vector notation, the matrix $\frac{\delta F}{\delta X}$ is called the Jacobian and is thus a function of the current variables, x_i, which maps the velocities of X to the desired output velocities of Y. At every time step the current variables, x_i, and their related velocities have changed so the Jacobian must be recalculated for each time step. In terms of the schematic in figure 1, the input variables, x_i, are the joint angles, θ_i, the output variables, y_i are the end effector positions, s_i.

In vector notation the velocities of the joint angles can be represented as a column vector, $\dot{\theta}$, and the end effector velocities as a column vector, V. Each element of the vectors correspond to each link's end effector or joint, therefore in the system presented in figure 1 each will be a three element vector. As such equation 2 can be written as

$$V = J\dot{\theta} \tag{3}$$

where J is the Jacobian matrix. In cartesian coordinates, x, y and z, this becomes

$$J = \begin{bmatrix} \frac{\delta p_x}{\delta \theta_1} & \cdots & \frac{\delta p_x}{\delta \theta_i} \\ \frac{\delta p_y}{\delta \theta_1} & \cdots & \frac{\delta p_y}{\delta \theta_i} \\ \vdots & \ddots & \vdots \\ \frac{\delta \alpha_z}{\delta \theta_1} & \cdots & \frac{\delta \alpha_z}{\delta \theta_i} \end{bmatrix} \tag{4}$$

where p is the change in end effector position of the link associated to joint θ_i and α is the change in orientation. Therefore, each component of the matrix relates a joint to a change in one coordinate of the end effector of that link.

In the example based on figure 1 the Jacobian can be simplified. Orientation can be ignored as in two dimensions all the joints rotate around the z axis, $(0, 0, 1)$. The incremental rotation is then much simpler to identify and can be determined via the cross-product of the joint axis and the vector of the joint position to the terminal end effector for each axis, (x, y, z). The Matlab function listed below is used to find the Jacobian matrix, e is used to find the vector from the terminal end effector and the target, all other symbols are the same as previously indicated.

```
function [J,V] = jacob_IK(s, t, P)
%% jacobian
e = [t;t;t] - s;
end_effector_vector = s - P;
end_effector_vector = [end_effector_vector, zeros(3,1)];
joint_axis = [0,0,1];
V = [e(3,:),0]'; % desired change to end effector
g = zeros(3);
for i = 1:3
    g(i,:) = cross(joint_axis, end_effector_vector(i,:));
end
J = g';
end
```

2.2 The Pseudoinverse Method

The now computed Jacobian can now be used to solve equation 3 as both V and J are known. This requires the inverse of the Jacobian, however in many situations J^{-1} does not exist as the solution is singular. In the case of the system described in this paper, the Jacobian potentially has infinite solutions and is not a square matrix due to the z component row always equals zero. This means the inverse does not exist. Instead the pseudoinverse, J^+, is

used as a matrix multiplied by its own transpose may have an inverse. J^+ is defined as

$$V = J\dot{\theta} \tag{5}$$
$$J^T V = J^T J\dot{\theta} \tag{6}$$
$$(J^T J)^{-1} J^T V = \dot{\theta} \tag{7}$$
$$J^T (JJ^T)^{-1} V = \dot{\theta} \tag{8}$$
$$J^+ V = \dot{\theta}. \tag{9}$$

If the rows of the Jacobian are linearly independent then JJ^T is invertible so J^+ has a solution. This method is widely used but often performs poorly due to instability near singularities, as such this model will go further and employ a damped least squares method.

2.3 Damped Least Squares

To overcome the failings of the Pseudoinverse method near singularities, the problem is reformulated into a damped least squares calculation. The damped least squares method was first formulated by Wampler, [3], and balances the error in the solution against the size of the solution. This is achieved by minimising the sum

$$||J\dot{\theta} - V||^2 + \lambda^2 ||\dot{\theta}||^2 \tag{10}$$

which is equivalent to

$$\left\| \begin{pmatrix} J \\ \lambda I \end{pmatrix} \dot{\theta} - \begin{pmatrix} V \\ 0 \end{pmatrix} \right\|. \tag{11}$$

The corresponding equation is then

$$\begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} J \\ \lambda I \end{pmatrix} \dot{\theta} = \begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} V \\ 0 \end{pmatrix} \tag{12}$$

which when simplified can be written as

$$\dot{\theta} = J^T (JJ^T + \lambda^2 I)^{-1} V. \tag{13}$$

It is possible to do this rearrangement as $J^T J$ is a nxn matrix of a size dependent on the degrees of freedom of the system. It can be shown via single value decomposition that $J^T J + \lambda^2 I$ is nonsingular and the statement. In Matlab code created for this system the function for carrying out the damped least squares method is as follows

```
function theta = psudo_inv_damp(J, damp, V, cur_alpha, theta)
delta_theta = J'*inv(J*J' + eye(3).*damp^2)*V;
for i=1:3
    if delta_theta(i) > cur_alpha
        delta_theta(i) = cur_alpha;
    elseif delta_theta(i) < -cur_alpha
        delta_theta(i) = -cur_alpha;
    end
end
theta = theta + delta_theta;
end
```

a value named "cur_ alpha" is used to limit the maximum angle change per iteration. It was found this provided a much smoother animation, the results of which will be discussed in section 3.

2.4 Controlling Joint Parameters

It is possible to control joint stiffness and bias a joint towards certain angles by including the following term

$$z = \beta_i (\theta_i - \theta_{di})^2 \tag{14}$$

where z is the biasing equation to be used, β_i is the "joint stiffness" parameter for joint i, θ_i is the current joint angle and θ_{di} is the desired bias angle. This equation can be used to form the control expression

$$\dot{\theta} = (J^+ J - I)z. \tag{15}$$

As the form of equation 15 contributes nothing to the end effector velocities, V, it can be simply added to the pseudoinverse function

$$\dot{\theta} = J^+ V + (J^+ J - I)z. \tag{16}$$

A similar approach can be taken with the least squares approach leading to the following equation:

$$\begin{aligned} \dot{\theta} &= J^T (JJ^T + \lambda^2 I)^{-1} V \\ &\quad + (J^T (JJ^T + \lambda^2 I)^{-1} J - I))z. \end{aligned} \tag{17}$$

The results of which are shown in 3. The developed code for this proposed equation is as follows

```
function theta =
    psudo_inv_damp_jointres(J,
    damp, V, alpha, theta, j_gains
    , bias_angle)
delta_theta = J'*inv(J*J' + eye
    (3).*damp^2)*V + (J'*inv(J*J'+
     eye(3).*damp^2)*J - eye(3))*(
    j_gains.*((theta-(bias_angle)
    .^2)));

for i=1:3
    if delta_theta(i) > alpha
        delta_theta(i) = alpha;
    elseif delta_theta(i) < -
        alpha
        delta_theta(i) = -alpha;
    end
end
theta = theta + delta_theta;
end
```

3 Results & Discussion

3.1 Application Overview

The completed application can be seen in figure 2. The left hand side of the GUI is used for changing the various simulation and animation parameters which default to values that appropriately demonstrate the basic functionality. The right hand side of the GUI contains a set of axes for displaying the output animation. The skeleton is shown as a blue, three segmented line, where each of the segments lengths can be controlled. The starting location of the terminal end effector can be set as well as the target location.

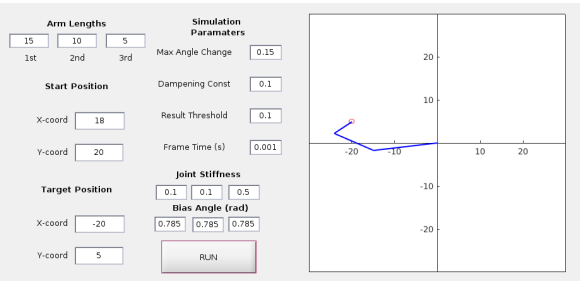


Figure 2: The final application. A larger version of this image can be found in appendix A.

There is also a collection of input fields for changing the maximum angle change for each iteration of the Jacobian, the damping constant, the acceptable result threshold, the time each frame of the animation holds for and parameters for changing joint behaviour.

3.2 Target Location Limits

The target location is indicated by a red circle marker as shown in figure 3. The terminal end effector will then move towards the center of this circle and halt when within the selected threshold distance of the target.

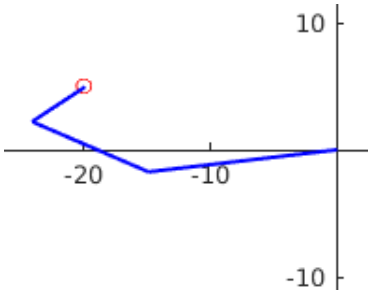


Figure 3: The set target location is indicated by a red circle.

If a target location is selected that is outside of the reach of the arm, which is defined as the sum of the individual link lengths, the following error message is provided to the user:

```
Error using IK_GUI (line 13)
A target location that is out of reach of the arm has been chosen
```

This message is generated by a throw exception function as follows

```
if max_Length < (t_x^2 + t_y^2)
    ^(1/2)
    msgID = 'Invrs_Kin:OutOfRange
        ';
    msg = 'A target location that
        is out of reach of the
        arm has been chosen';
    baseException = MException(
        msgID,msg);
    throw(baseException)
end
```

3.3 Setting Start Locations

The start location of the end effector can also be chosen within the GUI. Should a start location be chosen that is out of range of the arms maximum reach a similar error message to selecting an out of range target is given. The arm is moved to the start location via a non-animated inverse kinematic process. This allows start coordinates to be selected rather than defining starting angles. Calculating the start location via inverse kinematics also means the arm will be in a more natural position rather than predefined angles. On the other hand, by using inverse kinematics to define the start location, the terminal end effector will be a small distance away from the actual specified coordinates. This threshold is set to be a very low value, in this case 0.01, and can be changed from the "I_ Kin_ Start.m" function.

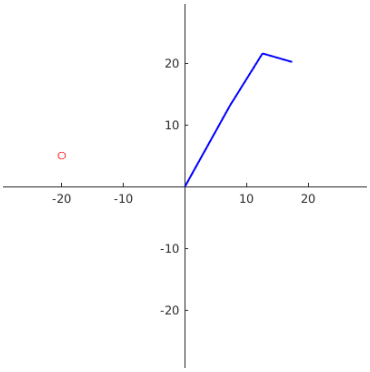


Figure 4: The start location of a default length arm set to coordinates $(18, 20)$.

3.4 Inverse Kinematic Iterations

Each time the Jacobian is calculated, the new angles are computed and the coordinates of the links are updated. These coordinates are saved to a CSV file, an example of which can be found in appendix B, these coordinates are used to update the arms position and the next iteration occurs. This repeats until the terminal end effector is within the threshold distance of the target. The amount of iterations and subsequently the arms path can be drastically altered by changing various parameters. The default parameters for a path between $(18, 20)$ and $(-20, 5)$ can be seen in figure 5.

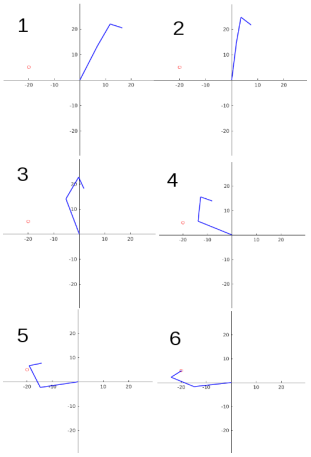


Figure 5: A series of images showing the path of the arm using the default parameters.

3.4.1 Effect of Maximum Angle Change

Changing the maximum angle change parameter changes the maximum amount theta for each joint can change per iteration of the Jacobian. Smaller values of this parameter, in the majority of cases, lead to many more iterations but also smoother animation paths and greater stability once close to the target. Figure 6 shows how the parameter alters the final position of the arm. Large values of this parameter can cause instability and unusual paths for the arm, as shown in figure 7. The large value can cause the arm to overshoot the target causing it to oscillate around the target location until eventually landing within the threshold, this is found to be much less likely for small values of this parameter. An optimal value of 0.15 was chosen as the default as a good balance between stability and iterations for fast computation.

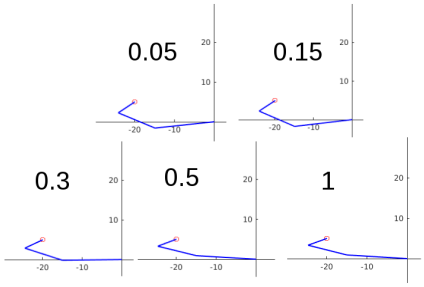


Figure 6: The final positions of the arm for various maximum angle change values. The parameter value is labeled.

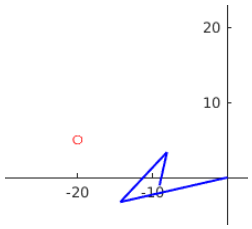


Figure 7: An intermediary position of the arm with a parameter value of 1 showing an unusual path

3.4.2 Effect of Dampening Constant

The damping constant, as discussed in section 2.3, helps to improve the behaviour of the system close to singularities. What was also observed when changing the damping constant is the arms path can be drastically changed. A large constant was found to cause the arm to undergo much more vigorous changes in position when far from the target and much slower changes close to the target.

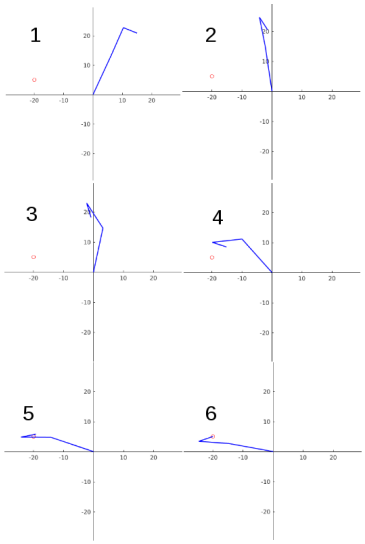


Figure 8: A series of images showing the path of the arm using a damping constant of 0.9.

This can be seen in figure 8, panels 3 and 4, where the arm bends in the opposite direction to the default parameters in figure 5. This is due to the iterative steps being made larger by the addition of the large constant term in equation 13. Too big a value and the arm will begin to overshoot and fail to reach the target position. It was found that a value of 0.1 was sufficient to improve behaviour around the arm reach limits and was not too big to cause unnatural iteration steps.

3.4.3 Effect of Joint Stiffness and Bias Angle

The stiffness parameters and bias angles can be set to manipulate the motion path and the final position as shown in figure 9. The higher the stiffness parameter the more rigidly the angle will attempt to stay close to the selected bias angle.

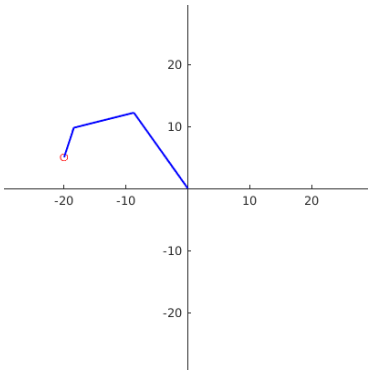


Figure 9: The final position of an arm where a stiffness parameter (0.9, 0.1, 0.1) is set to reduce the first joints movement and a bias angle (0.785, 1.5, 0.785) chosen to bias the angle of the second joint towards a 1.5 radians.

3.5 Slow In Slow Out Functionality

The iterative outputs that form the CSV file in appendix B are used as key frames for creating the animation. The movement between each iterative position is found by interpolating between the initial and final coordinates of each iteration. The number of interpolation steps is determined by the terminal end effectors position. The closer the terminal end effector is to the start and target locations the more interpolation steps. This allows for a slow in, slow out, functionality causing the arm to move slower when starting and approaching the target causing the animation to look as a natural movement.

```
function cur_steps = slow_IO(s, t
    , dist_to_targ, numSteps)
cur_dist_targ = norm(s(3,:)-t);
```

```
if cur_dist_targ > dist_to_targ/2
    cur_steps = numSteps*(
        cur_dist_targ/(
        dist_to_targ/2))*2;
else
    cur_steps = numSteps*((
        dist_to_targ/2)/
        cur_dist_targ);
end
if cur_steps < 30
    cur_steps = 30;
else if cur_steps > 100
        cur_steps = 100;
end
end
```

One issue with this implementation is that the speed is determined by the arms proximity to the start and end locations, this causes noticeable slowing of the animation if the arm happens to move close to and then away from the target.

4 Modeling Flexibility

Adding in the possibility of each link bending effectively offers additional degrees of freedom. The simplest bend to model is to assume the link bends with a circular arc with a constant arc length that is equal to the length of the link, shown in figure 10. The position of any point, M_j, on the flexible body can be described as the superposition of the rigid state position, M_{0j}, and an elastic deformation, u_{ej}, where u_{ej} is a function that acts on point M_{0j} and is formed from Rayleigh-Ritz shape functions [4].

The resulting vector $r_{O_jM_j}$ giving the position of M_j is found by

$$r_{O_jM_j} = r_{O_jM_{0j}} + u_{ej}(M_{0j}). \tag{18}$$

The translational velocities due to the bodies elasticity can be found and these velocities can be used within the Jacobian process as previously used [5]. This approach has been used to model various flexible bodies such as protein chains [6] and snake-like robots [7].

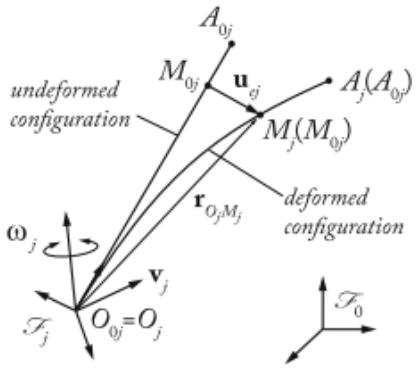


Figure 10: A schematic of a flexible link [5].

5 Conclusions

The application present successfully models a three link, three joint, system which generates a movement path towards a target position via a damped least squares method. Packaged into a simple GUI, all key parameters are easily changeable. The system indicates the target location as well as warning the user when the target location is out of reach of the arm. Basic physical properties are modeled such as joint stiffness as well as biasing a joint towards a certain angle.

The iterative outputs from the damped least squares method are used as keyframes to animate the movement of the arm. A slow in, slow out, functionality is implemented via manipulating the interpolation steps of the key frame animation.

Link flexibility was discussed with a short introduction to the mathematical changes that would need to be made to the model as well as examples of implemented rod flexibility models. Due to the increased mathematical complexity it was decided to not implement this functionality.

References

[1] Jadran Lenarčič and Philippe Wenger. *Advances in Robot Kinematics: Analysis and Design*. Springer, Dordrecht, 2008.

[2] W. A. Wolovich and H. Elliott. A computational technique for inverse kinematics. In *The 23rd IEEE Conference on Decision and Control*, pages 1359–1363, Dec 1984.

[3] C. W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):93–101, Jan 1986.

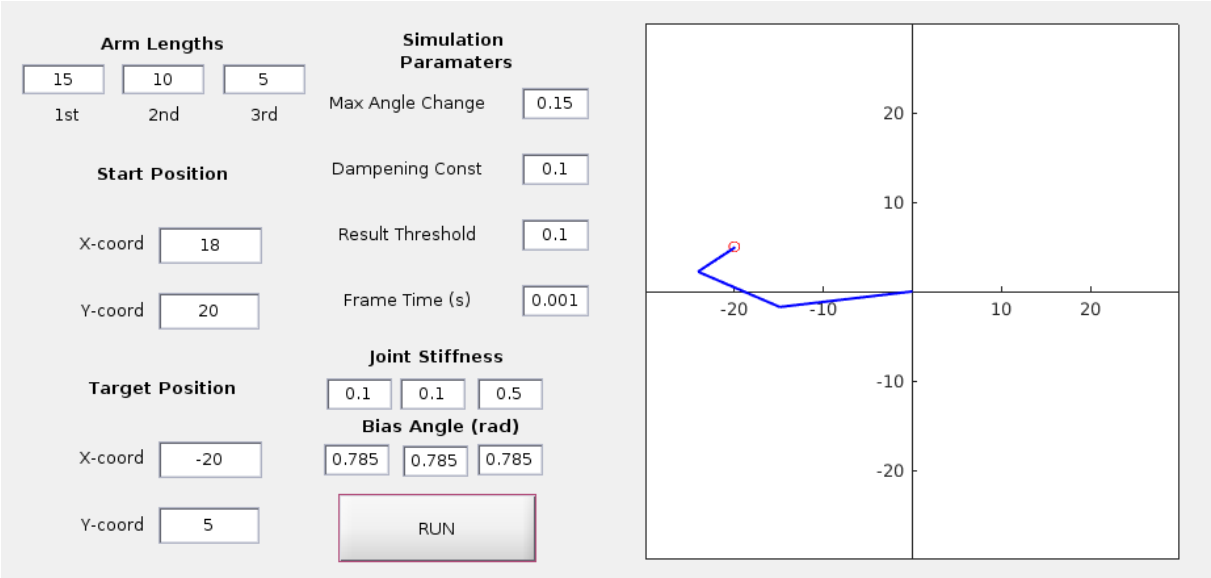
[4] Robert D. Blevins. *Formulas for natural frequency*. Krieger Reprint, Malabar, 2001.

[5] Sébastien Briot and Wisama Khalil. *Dynamics of Parallel Robots*. Springer, Switzerland, 2015.

[6] Steven Hayward and Akio Kitao. Monte carlo sampling with linear inverse kinematics for simulation of protein flexible regions. *Journal of Chemical Theory and Computation*, 11(8):3895–3905, 2015. PMID: 26574470.

[7] Z. Zhang, G. Yang, and S. H. Yeo. Inverse kinematics of modular cable-driven snake-like robots with flexible backbones. In *2011 IEEE 5th International Conference on Robotics, Automation and Mechatronics (RAM)*, pages 41–46, Sept 2011.

A Application GUI



B CSV Coordinate File

```
0
,
0,0
7.6383,12.91
7.6383,12.91
13.158,21.248
13.158,21.248
18,20
,
0,0
5.6233,13.906
5.6233,13.906
```

```
9.8352,22.976
9.8352,22.976
14.436,21.018
,
0,0
3.4821,14.59
3.4821,14.59
6.2913,24.188
6.2913,24.188
10.548,21.564
,
0,0
1.2626,14.947
1.2626,14.947
2.6061,24.856
2.6061,24.856
6.423,21.626
,
0,0
-0.98515,14.968
-0.98515,14.968
1.2928,24.705
1.2928,24.705
4.5842,20.941
,
0,0
-3.2108,14.652
-3.2108,14.652
0.49667,23.94
0.49667,23.94
3.1886,19.726
,
0,0
-5.3644,14.008
-5.3644,14.008
-0.31065,22.637
-0.31065,22.637
1.7214,18.069
,
0,0
-7.3975,13.049
-7.3975,13.049
-1.111,20.826
-1.111,20.826
1.5547,16.596
,
0,0
-9.2644,11.797
-9.2644,11.797
-3.4027,19.899
-3.4027,19.899
-0.13479,16.115
,
0,0
-10.923,10.28
-10.923,10.28
-6.3382,19.167
-6.3382,19.167
-2.5414,15.914
,
0,0
-12.337,8.5323
-12.337,8.5323
-9.1313,18.005
-9.1313,18.005
-4.891,15.355
,
0,0
-13.473,6.5929
-13.473,6.5929
-11.719,16.438
-11.719,16.438
-7.1307,14.452
,
0,0
-14.307,4.5054
-14.307,4.5054
-14.044,14.502
-14.044,14.502
-9.2103,13.224
,
0,0
-14.82,2.3168
-14.82,2.3168
-16.054,12.24
-16.054,12.24
-11.083,11.699
,
0,0
-15,0.076086
-15,0.076086
-17.703,9.7039
-17.703,9.7039
-12.707,9.9114
,
0,0
-14.843,-2.1663
-14.843,-2.1663
-18.954,6.9495
-18.954,6.9495
-14.039,7.8656
,
0,0
-14.707,-2.9507
-14.707,-2.9507
-20.134,5.4484
-20.134,5.4484
-15.37,6.9671
,
0,0
-14.774,-2.593
-14.774,-2.593
-21.396,4.9008
-21.396,4.9008
-16.824,6.9247
,
0,0
-14.822,-2.3013
-14.822,-2.3013
-22.489,4.1189
-22.489,4.1189
-18.107,6.5269
,
0,0
-14.863,-2.0254
-14.863,-2.0254
-23.403,3.1769
-23.403,3.1769
-19.163,5.8264
,
0,0
-14.899,-1.7376
-14.899,-1.7376
-24.089,2.2061
-24.089,2.2061
-19.91,4.9517
```