Analysis of Generalized Revolute Kinematic Chains

Joseph Adamson
Department of ECE
Baskin School of Engineering, UCSC
Santa Cruz, California
jmadamso@ucsc.edu

***Abstract*—This paper provides an exploration of the different properties induced in a kinematic chain by varying the DH parameters and providing various degrees of freedom. An overview of the process is provided, Simulation results, which we generated using the versatile Robotics Toolbox for MATLAB, are provided and analyzed.**

I. Introduction and Motivation

Kinematic chains consist, roughly speaking, of a sequence of links that are connected with joints. The most elementary types of joint are *revolute* and *prismatic*, which allow link movement in rotational or linear relationships, respectively. When actuated, these joints allow the chain,which may be considered as a robotic arm, to move around a workspace, often in conjunction with a manipulator at the wrist to achieve some task.

These chains have been studied to a high degree, with analysis seemingly as old as the field of robotics itself. The coursework of CMPE215 introduces standard techniques for analyzing them, most notably including the math used to describe end-effector orientation. The 6 degree-of-freedom orientation can be described using a 4x4 matrix representing translation and rotation, known as the *Homogeneous Transform*.

The formulation of these kinematic chains in the language of linear algebra provides a very powerful set of tools to map between the spaces we care about; specifically, we can easily transform between *Joint Space*, which describes the arm as a vector of its joint positions, and *Cartesian Space*, which describes the arm as a vector of its orientation in 3D space.

The notion of a degree of freedom (DOF) describes how many axes something is able to move along, and consists in euclidean space of three translational displacements and three angles, resulting in 6 DOF's required to fully orient something in space.

Since linear algebra can easily be as arbitrary as it is concrete, the research presented here is motivated by the generalization of kinematic chains. Specifically, we explore chains containing only revolute joints and vary the parameter α across chains with varying amount of links, and thus with varying degrees of freedom. Keeping the analysis fairly abstract aligns with the author's goal of turning a curious and exploratory eye to the concepts developed during this coursework.

II. Theoretical Analysis

A. Denavit-Hartenburg Parameters

The following parameters, known as DH parameters and referenced from [1] and [2], can be used to fully describe a kinematic link and may be seen in the figure below:

a_{i-1} : Link Length to previous link
α_{i-1} : Link Twist from previous link
d_i : Link Offset between links
θ_i : Link Angle

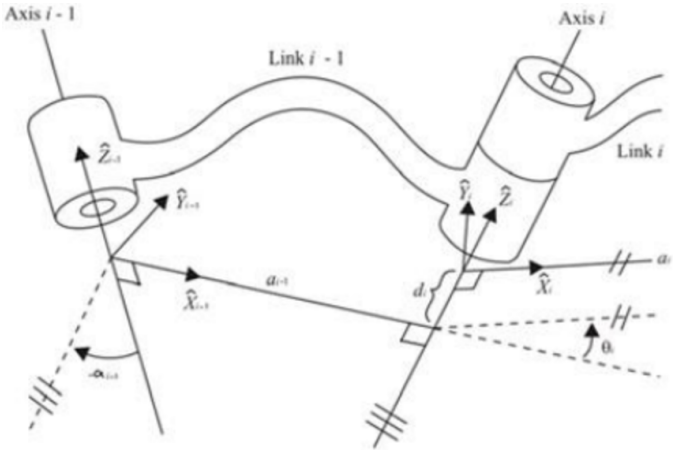

Fig. 1. DH Parameters

An in-depth analysis of these parameters is beyond the scope of this paper; see [1] for more information. In summary, these parameters can define any link. For a revolute joint, θ is the joint variable that we can actuate, while a prismatic joint allows d to vary.

For our analysis, the following parameters are used, and every link will be considered identical with the exception of the actuated joint variable θ. The twist parameter α is a subject of study and will be adjusted in the specified range during analysis.

d	a	α	l
.3	1	$[-\pi/6, -\pi/2]$	1

B. Forward Kinematics

Using the DH parameters, one may construct a transformation matrix in order to describe the orientation of the wrist

when the positions of every joint is known. In other words, the forward kinematics map a kinematic chain from joint space into Cartesian space and answer the question: "If my joints are at these angles, where is my wrist?".

This mapping will depend on the different DH parameters, which geometrically describe the position of each link. These calculations result in a nested sequence of homogeneous transforms, resulting in a final 4x4 forward kinematics matrix.

A model of the forward kinematics for a 4-link revolute kinematic chain may be found in appendix A.

C. Inverse Kinematics

The inverse kinematics calculations work in the opposite direction, and may be formulated as a matrix which 'undoes' the forward calculations; in other words, it maps from cartesian to joint space and answers the question "If I want my wrist in the following position, where do I position my joints?"

Inverse kinematics are generally uglier, and demonstrate that a kinematic chain may have more than one way to reach a desired point. Their math generally involves arctangents and other inverse trig functions, which have multiple solutions and also give rise to singularities. In other words, a kinematic chain will have regions that the math goes to $\pm\infty$, which can generally be considered as a bad thing.

This research was driven in part by the author's interest in overdetermined systems, in which the project task requires far fewer degrees of freedom than the robot has; in other words, these systems will have infinite solutions.

Though it is not discussed in detail here, the subject of inverse kinematics gives rise to a multitude of optimisation possibilities since the systems tend to have enough extra dimensionality in which to impose constraints and perform other optimal control techniques.

III. Simulation Method

The focus of this project is largely to explore the conceptual effect of varying parameters. This is reflected in the approach taken for simulation, which emphasized generalization/scalability of parameters and quick changes using variables. In this manner, several configurations were explored, with results given in section IV. As described above, the analysis is limited to revolute-only chains.

A. Assumptions/Simplifications

To aid the exploratory process, we need to make a few simplifications:

- Self-Intersection is ignored
- Joints are limited to ± 179 degrees to avoid singularities
- Assume kinematic calculations only (no overshoot or other dynamics behavior)

B. Robot Toolbox

Peter Corke's Robotics Toolbox for MATLAB is an extremely versatile tool for performing simulations. It provides an API for instantiating robot links according to the DH parameters, as well as a host of other analysis tools including forward/inverse kinematics, trajectory generation, and plotting/animation capabilities. The toolbox was used in conjunction with native MATLAB functionality in each of the simulations described below.

Importantly, the toolbox provides several methods for calculating inverse kinematics, using various types of optimization. The simulations provided here are using the version which attempts to minimise the error between the calculated and desired values through the use of the MATLAB minimization tool *fmincon*.

Further detail can be found in the documentation of [3].

C. Method

The simulation was scripted to iteratively determine the performance of a given configuration as follows:

1) Generate 10 random setpoints. The setpoint axes are allowed to take on any rotation value between 0 and 2π and allowed to translate anywhere within a specified radius.
2) Perform inverse kinematics for each setpoint using optimisation
3) Calculate the forward kinematics of the resultant angles (thus calculating the position reached by the configuration)
4) Calculate and sum the error for each setpoint
5) repeat 25 times and average the error

The radius was chosen to be 1/3 the length of the arm when fully extended, so as to provide enough room for the arm to 'curl around' to reach the target.

We define error as the difference in euclidean norms between the desired and actual orientations transforms:

$$err = ||T_{des}||_2 - ||T_{act}||_2$$

This method of calculating the error is arbitrary, and doesn't use concrete units; it is simply a quick way to analyze the numerical differences describing where the arm was actually able to reach. The author notes that this methodology is fairly inapplicable to the design of real-world robots, but functions as a sort of academia-focused sandbox.

IV. Simulation Results

A. Initial Observations

Initial research involved instantiating various arm configurations and watching their behavior. Using only negative values for α causes the arm to behave rather like a human elbow, as shown in figure 2.

It was noted during initial simulations that the arm, which is curling counterclockwise akin to a human right arm, performs better when working in the left half of the workspace. This is fairly intuitive, given that human arms, which are angled quite similarly, don't function well when not handling an object directly in front of the person they are attached to.

This gives rise to the observation of a 'preferred' workspace in which the arm is better able to reach arbitrary orientations.

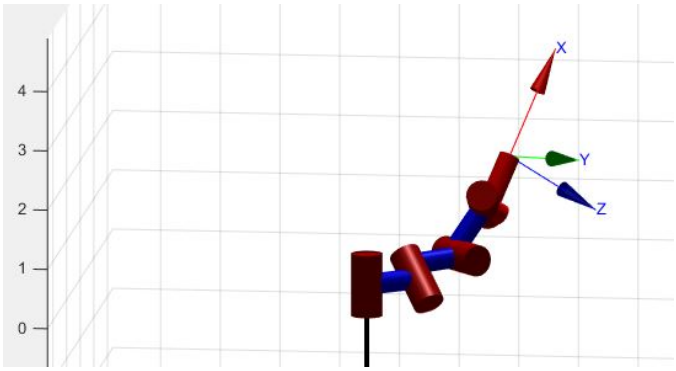

Fig. 2. Arm with $\alpha = -pi/4$, 4 links

Another observation made during initial simulations was that the arm can easily 'fold in' on itself when the joints are allowed to rotate more than a half circle. During these severe singularities, the math is unable to re-extract proper joint angles since they have collapsed in the matrices and the robot sort of curled into a ball and looked very sad. Imposing the joint limits described above avoids this issue.

B. Configuration Performance

The following table summarizes the results found according to the method described in III-C. The configurations chosen for analysis reflect three different α values in the 6-DOF case, as well as under-, over-, and very-over-actuated cases using a few different α values.

α	links	Avg. Error
$-\pi/6$	6	5.53
$-\pi/4$	6	4.28
$-\pi/2$	6	2.78
$-\pi/4$	4	3.88
$-\pi/2$	4	2.96
$-\pi/4$	8	3.98
$-\pi/2$	8	2.20
$-\pi/2$	20	∞^*

*this one crashed matlab

It can be seen that the error decreases when the angle between links is larger, and performance is generally better with more links. Intuitively, this makes sense given that a larger alpha angle allows each joint to cover more area; the arm is able to make wider arcs and curve its trajectory more to reach the desired orientation.

A good example of the robot nearly reaching it's goal can be seen in figure 3. The green line indicates the path it took from a resting position.

When a kinematic chain of this nature has more degrees of freedom, the performance is also increased because in effect there are more twists available. Since the kinematics consist of a coordinated effort to reach the goal, adding additional links is quite literally adding extra dimensions in which the arm can move.

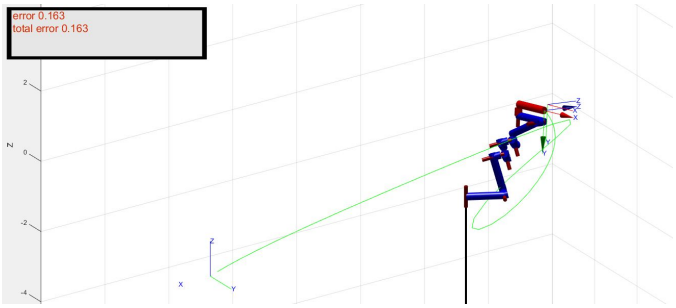

Fig. 3. Setpoint Attempt with $\alpha = \pi/2$ and 8 links

Though the math involved in this is beyond the scope of this paper, it is suffice to say that when the system becomes overactuated (ie, more than 6 DOF) it can generally be considered that there are infinite solutions to how such an arm may reach its goal. However, the results of these simulations indicate that it's not enough to just add more joints; one may safely conclude that kinematic chains of the type explored here are sub-optimal for real-world tasks, as the imposed restrictions (revolute-only and non-orthogonal links) create a limited dextrous workspace.

V. Future Work

Future work will be done at the author's discretion as the goals for this coursework have been met as it is. Regardless, the following tasks remain topics of interest:

- Perform separate simulations to analyze preferred quadrants as described above
- Generate reachable workspace plots
- Build one!

References

[1] J. Craig, Introduction to Robotics: Mechanics and Control. Upper Saddle River, N.J.: Pearson/Prentice Hall, 2005.

[2] Michael Wehner, CMPE215 Lecture Slides; Accessed via Canvas
I have literally no idea how to cite this

[3] Peter Corke, Robotics Toolbox, Matlab Toolbox. https://petercorke.com/wordpress/toolboxes/robotics-toolbox

APPENDIX A - FORWARD KINEMATICS FOR 4 LINKS

$$ {}^{4}_{0}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$\mathrm{r}_{11} = -c\theta_4(s\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2)) - s\theta_4(c\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) - s\alpha^2(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + c\alpha c\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2))$

$\mathrm{r}_{12} = s\alpha(c\alpha s\alpha(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + s\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) + c\theta_3 s\alpha(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) + c\alpha s\theta_4(s\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2)) - c\alpha c\theta_4(c\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) - s\alpha^2(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + c\alpha c\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2))$

$\mathrm{r}_{13} = c\alpha(c\alpha s\alpha(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + s\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) + c\theta_3 s\alpha(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) - s\alpha s\theta_4(s\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2)) + c\theta_4 s\alpha(c\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) - s\alpha^2(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + c\alpha c\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2))$

$\mathrm{r}_{14} = ac\theta_1 + d(c\alpha s\alpha(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + s\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) + c\theta_3 s\alpha(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) - as\theta_4(c\alpha s\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) - s\alpha^2(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) + c\alpha c\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) + ac\theta_1 c\theta_2 + ds\alpha s\theta_1 - as\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - ac\theta_4(s\theta_3(c\alpha^2 c\theta_2 s\theta_1 - s\alpha^2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2)) + ac\theta_3(c\theta_1 c\theta_2 - c\alpha s\theta_1 s\theta_2) + ds\alpha(c\alpha s\theta_1 + c\theta_1 s\theta_2 + c\alpha c\theta_2 s\theta_1) - ac\alpha s\theta_1 s\theta_2$

$\mathrm{r}_{21} = -c\theta_4(s\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2) - c\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) - s\theta_4(s\alpha^2(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) + c\alpha c\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2))$

$\mathrm{r}_{22} = s\alpha(s\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\alpha s\alpha(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\theta_3 s\alpha(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2)) + c\alpha s\theta_4(s\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2) - c\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) - c\alpha c\theta_4(s\alpha^2(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) + c\alpha c\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2))$

$\mathrm{r}_{23} = c\alpha(s\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\alpha s\alpha(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\theta_3 s\alpha(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2)) - s\alpha s\theta_4(s\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2) - c\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) + c\theta_4 s\alpha(s\alpha^2(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) + c\alpha c\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2))$

$\mathrm{r}_{24} = as\theta_1 + d(s\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) - c\alpha s\alpha(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\theta_3 s\alpha(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2)) - as\theta_4(s\alpha^2(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + c\alpha s\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) + c\alpha c\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2)) - dc\theta_1 s\alpha + ac\theta_2 s\theta_1 - as\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2) - ac\theta_4(s\theta_3(c\theta_1 s\alpha^2 - c\alpha^2 c\theta_1 c\theta_2 + c\alpha s\theta_1 s\theta_2) - c\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2)) - ds\alpha(c\alpha c\theta_1 - s\theta_1 s\theta_2 + c\alpha c\theta_1 c\theta_2) + ac\theta_3(c\theta_2 s\theta_1 + c\alpha c\theta_1 s\theta_2) + ac\alpha c\theta_1 s\theta_2$

$\mathrm{r}_{31} = c\theta_4 s\alpha(c\alpha s\theta_3 + c\theta_3 s\theta_2 + c\alpha c\theta_2 s\theta_3) + s\alpha s\theta_4(c\alpha^2 + c\alpha^2 c\theta_3 - c\theta_2 s\alpha^2 + c\alpha^2 c\theta_2 c\theta_3 - c\alpha s\theta_2 s\theta_3)$

$\mathrm{r}_{32} = s\alpha(c\alpha(c\alpha^2 - c\theta_2 s\alpha^2) - c\theta_3 s\alpha(c\alpha s\alpha + c\alpha c\theta_2 s\alpha) + s\alpha^2 s\theta_2 s\theta_3) + c\alpha c\theta_4(s\alpha(c\alpha^2 - c\theta_2 s\alpha^2) + c\alpha c\theta_3(c\alpha s\alpha + c\alpha c\theta_2 s\alpha) - c\alpha s\alpha s\theta_2 s\theta_3) - c\alpha s\theta_4(s\theta_3(c\alpha s\alpha + c\alpha c\theta_2 s\alpha) + c\theta_3 s\alpha s\theta_2)$

$\mathrm{r}_{33} = c\alpha(c\alpha^3 + s\alpha^2 s\theta_2 s\theta_3 + c\alpha c\theta_2(c\alpha^2 - 1) + c\alpha c\theta_3(c\alpha^2 - 1) - c\alpha c\theta_2 c\theta_3 s\alpha^2) - c\theta_4 s\alpha^2(c\alpha^2 + c\alpha^2 c\theta_3 - c\theta_2 s\alpha^2 + c\alpha^2 c\theta_2 c\theta_3 - c\alpha s\theta_2 s\theta_3) + s\alpha^2 s\theta_4(c\alpha s\theta_3 + c\theta_3 s\theta_2 + c\alpha c\theta_2 s\theta_3)$

$\mathrm{r}_{34} = d + dc\alpha + d(c\alpha^3 + s\alpha^2 s\theta_2 s\theta_3 + c\alpha c\theta_2(c\alpha^2 - 1) + c\alpha c\theta_3(c\alpha^2 - 1) - c\alpha c\theta_2 c\theta_3 s\alpha^2) + d(c\alpha^2 - c\theta_2 s\alpha^2) + as\alpha s\theta_2 + ac\theta_3 s\alpha s\theta_2 + ac\theta_4 s\alpha(c\alpha s\theta_3 + c\theta_3 s\theta_2 + c\alpha c\theta_2 s\theta_3) + as\alpha s\theta_4(c\alpha^2 + c\alpha^2 c\theta_3 - c\theta_2 s\alpha^2 + c\alpha^2 c\theta_2 c\theta_3 - c\alpha s\theta_2 s\theta_3) + ac\alpha s\alpha s\theta_3(c\theta_2 + 1)$

Generated using symbolic capabilities of the Robotics Toolbox. The author notes that this appendix is presented as humorous overkill.