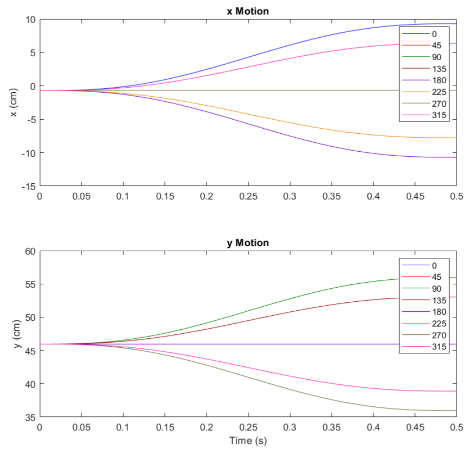
ES 249 Mini-Project

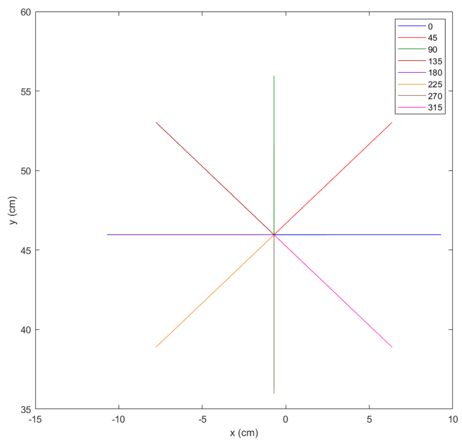
Andrew T. Sullivan

17 May 2019

1 Question 1: Torques Required on 2-Joint Planar Arm for Minimum-Jerk Movements

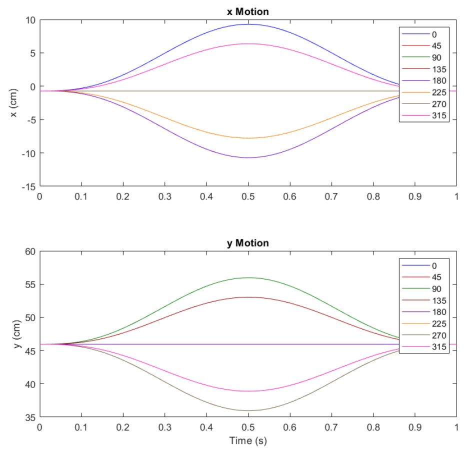


(a) x and y for different directions.

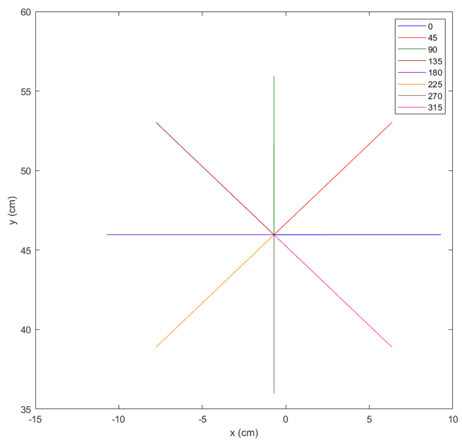


(b) x vs. y for different direction.

Figure 1: Simulation of Dynamics for Minimum Jerk Trajectories.



(a) x and y for different directions.



(b) x vs. y for different direction.

Figure 2: Simulation of Out-and-Back Dynamics for Minimum Jerk Trajectories.

We begin by considering unconstrained trajectory optimization using a mimimum cumlative squared jerk (CSJ) criterion. This has previously been discussed for motion in one dimension, and we now simply extend it to two. Given that the two coordinates, x and y, can be treated independently, the solution to this problem is rather trivial as we have already solved the case for one dimensional CSJ. For two dimensions, our cost function takes the form $C = \arg\min_{x(t),y(t)} \int_{t_0}^{t_f} (\dddot{x}(t))^2 + (\dddot{y}(t))^2 dt$. Clearly this integral can be separated into two integrals of the form $\int_{t_0}^{t_f} (\dddot{x}(t))^2 dt$, which have been previously shown to yield solutions that are fifth-order polynomials. These solutions have also been obtained using Matlab's lsqnonlin function, which is used for trajectory planning in this exercise as well. The calculus of variations solution is reproduced in Appendix A for reference. We consider initial and boundary conditions identical to those used in this derivation, i.e. that both coordinates start from a value of 0, and that initial and final velocities and accelerations for both variables are zero. We specify some final value x_f and y_f for each variable, yielding a net motion of distance 10 cm with the individual components specified by one of eight angles between 0 and 315 degrees in increments of 45 degrees. For any case, the solution to the optimal trajectory is a fifth-order polynomial for both spatial coordinates of the form:

$$x(t) = 6t_f^{-5}x_f t^5 - 15t_f^{-4}x_f t^4 + 10t_f^{-3}x_f t^3$$
$$y(t) = 6t_f^{-5}y_f t^5 - 15t_f^{-4}y_f t^4 + 10t_f^{-3}y_f t^3$$

The first figure displays the solutions corresponding to these polynomials obtained using Matlab's lsqnonlin function by passing in an initial vector of 1000 zeros augmented with three copies of the initial (0) and final (x/y_f) conditions on either side into the diff operator, which finds the difference between adjacent points in the vector. Here, we specify a value of 3 for the second argument, corresponding to the third difference, an approximation of the jerk. We can then see that the optimal trajectory to minimize the cumulative squared jerk in two dimensions is a straight line with individual spatial profiles following the common sigmoidal shape of the fifth-order polynomial. Note that this figure does not directly display these solutions, but rather the final output of the dynamics simulation discussed later which perfectly reproduces this trajectory without any external perturbations.

By a similar explanation, we can see that the out-and-back motion in two dimensions is simply the previously derived one-dimensional trajectory (also a fifth-order polynomial) for both variables. Appendix B contains the derivation for the one-dimensional out-and-back solution, in which the same boundary conditions are considered (i.e. x, $\dot{x}$, and $\ddot{x}$ are initially zero; the velocity is zero at the "endpoint," at a position of x_f) except we do not explicitly constrain the acceleration to be zero at the endpoint. Here, the endpoint instead corresponds to the midpoint of the entire trajectory, and the second half is obtained by time-reversal of the first half and negation of the velocity values due to the change in trajectory direction. The cumulative squared jerk can be minimized with respect to the midpoint acceleration, a_{min}, to obtain the equation $a_{min} = -\frac{20x_f}{3t_f^2}$. The overall trajectory is then given by the fifth-order polynomial $x(t) = a_{min}(0.5t_f^{-3}t^5 - t_f^{-2}t^4 + 0.5t_f^{-1}t^3) + x_f(6t_f^{-5}t^5 - 15t_f^{-4}t^4 + 10t_f^{-3}t^3)$. We can obtain the individual points in this trajectory using a similar approach to the previous scenario. We construct vectors of length 1000 for both spatial variables and augment each with only two copies of the initial and final positions, allowing the acceleration to vary at the midpoint. After the lsqnonlin function minimizes the cumulative squared jerk of the outward trajectory, we time-reverse both spatial vectors and concatenate the initial and time-reversed trajectories (removing the redundant value at the midpoint) to obtain our path.

Once we have our trajectories in terms of the end-effector coordinates, x and y, which are shifted by the values corresponding to our initial conditions, we can use inverse kinematic relations to obtain the corresponding shoulder and elbow angles, θ_1 and θ_2, which are equivalent to this end-effector point with the given arm lengths $l_1 = 32cm$ and $l_2 = 33cm$. The initial end-effector coordinate is given using the simple forward-kinematic relationships, $x = l_1 cos(\theta_1) + l_2 cos(\theta_2)$ and $y = l_1 sin(\theta_1) + l_2 sin(\theta_2)$. Using initial angles of $\pi/4$ and $3\pi/4$, we obtain an initial point (-$\frac{\sqrt{2}}{2}$, 45.9619) which is added onto all the points in our obtained trajectories. We can then use the inverse kinematic relations from each (x,y) pair to obtain the corresponding angle pair (θ_1, θ_2) at each of the time values t, where the spacing in the vectors is set such that the final position is reached at time $T_f = 0.5$. The inverse kinematic relations are given for the configuration of interest using previously derived equations (see Appendix C), using the definition $l_3 = \sqrt{x^2 + y^2}$:

$$\theta_1 = \arctan\frac{y}{x} - \arccos\frac{l_2^2 - x^2 - y^2 - l_1^2}{-2l_1 l_3}$$
$$\theta_2 = \pi + \arctan\frac{y}{x} - \arccos\frac{l_2^2 - x^2 - y^2 - l_1^2}{-2l_1 l_3} - \arccos\frac{x^2 + y^2 - l_1^2 - l_2^2}{-2l_1 l_2}$$

Using these relations, we can obtain the corresponding arm configuration for each of the pairs throughout the trajectory. We then model the dynamics using the provided equation, $T = M(\theta)\ddot{\theta} + C(\theta)\dot{\theta}^2$, where the inertial matrix $M(\theta) =$

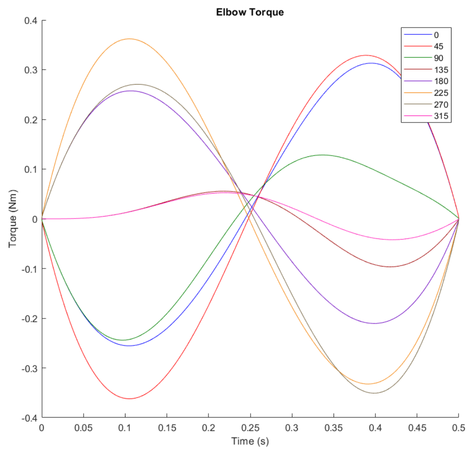
$$\begin{bmatrix} a & c\cos\theta_2 - \theta_1 \\ c\cos\theta_2 - \theta_1 & b \end{bmatrix}$$

And the Coriolis matrix $C(\theta) =$

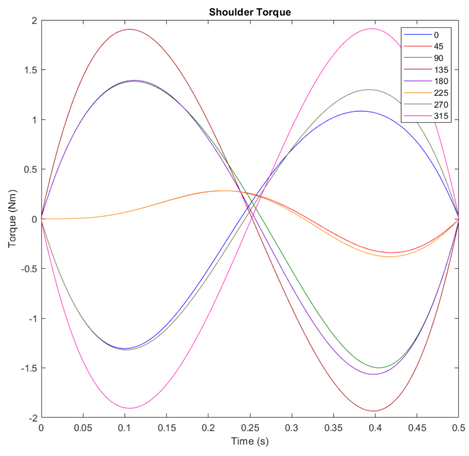
$$\begin{bmatrix} 0 & -c\sin\theta_2 - \theta_1 \\ c\sin\theta_2 - \theta_1 & 0 \end{bmatrix}$$

The coefficients are provided as $[a, b, c] = [0.265, 0.052, 0.0844] N/m^2$. It should be noted that these coefficients do not properly correspond to units that would yield torque (i.e. Nm, which would require units of Nms^2 due to the multiplication by the angular acceleration or square of the angular velocity). However, the dynamics that we obtain from these values properly reproduce the expected trajectory after converting the link lengths and final distance to meters. Our overall workflow is summarized as follows: we begin by specifying all the desired directions and start a loop that performs the same calculations for each direction. First, the change in x and y are obtained from the direction of interest through simple trigonometry. The lengths of each segment are specified along with the initial angular coordinates. A forward kinematics function is called to obtain the corresponding end-effector coordinates from these initial values, and the total time is specified. We then extract the trajectory of interest by using either a cumulative squared jerk or out-and-back cumulative squared jerk function which returns the optimal trajectories for inputs Δx, Δy, and T_f. We then use an inverse kinematics function to convert each of the coordinate pairs in this trajectory to angular pairs for our joints and verify that the original trajectory can be properly reconstructed. We then initialize our dynamics by specifying the coefficients for the inertial and Coriolis matrices, and find the angular velocity and angular acceleration using the diff function on the θ vectors. We append zeros to the end to enforce boundary conditions and ensure that all the vectors are the same length. We then call the dynamics function, which takes in the angle, angular velocity, and angular acceleration matrices and the coefficients and returns the joint torques applied to each joint at each time point using the above formula. We then simulate the dynamics using a function that takes in the applied joint torques, lengths of each segment of the arm, initial conditions for the angles, inertial/Coriolis coefficients, force-field coefficients (see the next question), and stiffness and viscosity coefficients (see the third question). The function returns the calculated angles at each point, their first and second derivatives, and the corresponding end-effector coordinates and their first and second derivatives. All of these matrices are initialized as zeros, and the initial conditions of the two angles are input into the angle matrix. These initial conditions are used to initialize the end-effector coordinates as well using forward kinematics. For each time point/ entry in the matrix, an iteration through a loop is conducted. First, the forces resulting from the force-field are evaluated (discussed further in question 2). The M and C matrices are evaluated using the current angles, and the dynamics equation is solved for the angular acceleration as $\ddot{\theta} = M^{-1}(\tau - C\dot{\theta}^2)$. From this angular acceleration, the kinematic equations are solved in each dimension (i.e. $\dot{\theta}_{i+1} = \dot{\theta}_i + \ddot{\theta}_i dt$ and $\theta_{i+1} = \theta_i + \dot{\theta}_i dt + \frac{1}{2}\ddot{\theta}_i dt^2$). From the updated angles, the forward kinematics function is used to find the updated end-effector coordinates, and the velocity and acceleration are then obtained as $\dot{x}_{i+1} = \frac{x_{i+1} - x_i}{dt}$ and $\ddot{x}_{i+1} = \frac{\dot{x}_{i+1} - \dot{x}_i}{dt}$. This simulation has been verified to properly reconstruct the trajectories without any external perturbations, and is subsequently modified to account for force-fields, stiffness, and viscosity.

The above figures display the trajectories for both outward and out-and-back dynamics in the eight specified directions. As expected from the previous discussion, the optimal trajectories in all directions are straight lines and individual spatial coordinates follow the familiar fifth-order polynomial sigmoidal trajectory.



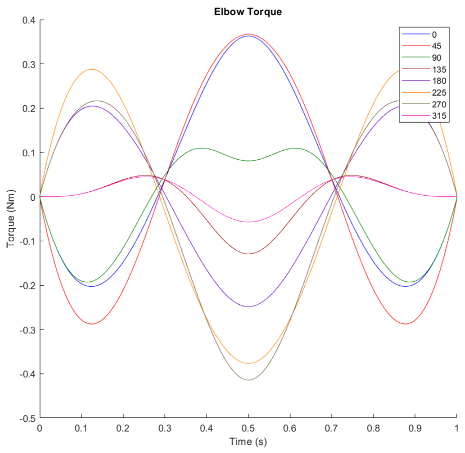
(a) Elbow Torque.



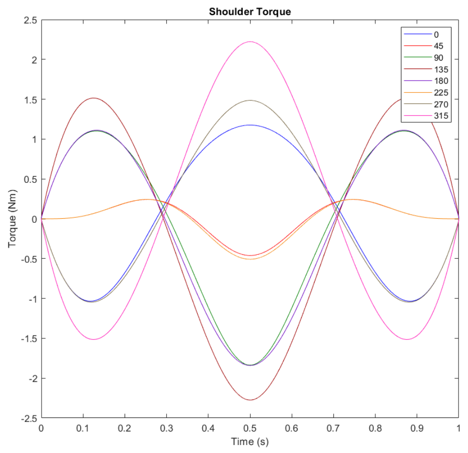
(b) Shoulder Torque.

Figure 3: Torques for Minimum Jerk Trajectories.

The next figure displays the computed torques arising from the specified dynamics. We can see that, like the acceleration, the torques are not constrained to be zero at the midpoint of the out-and-back case. Additionally, it appears that the torques are largely complementary, that is that for directions where one joint has a large torque profile, it seems as though the other joint typically has a smaller peak-to-peak magnitude.



(a) Elbow.



(b) Shoulder.

Figure 4: Torques for Out-and-Back Trajectories.

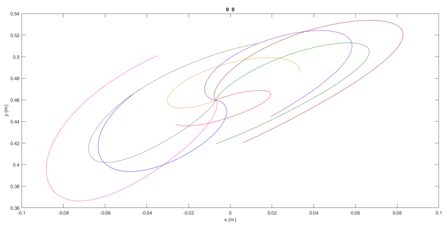
2 Question 2: Addition of a Velocity-Dependent Force-Field

2.1 a)

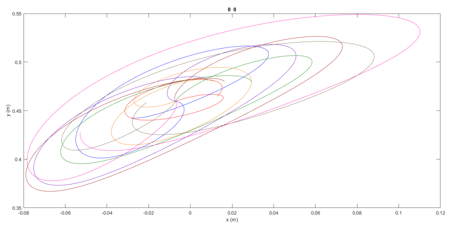
We next introduce the effect of a perturbative force-field with coefficients [0 15; -15 0]. This force field is introduced in a velocity-dependent manner, such that the net force $F_x = 15v_y$ and $F_y = -15v_x$. These forces are evaluated at each iteration of the loop by multiplying the current velocity (obtained during the previous iteration) by the coefficient matrix. The forces are converted to torques by considering the Jacobian matrix, $J =$

$$\begin{bmatrix} -l_1 sin(\theta_1) & -l_2 sin(\theta_2) \\ l_1 cos(\theta_1) & l_2 cos(\theta_2) \end{bmatrix}$$

This matrix can be seen to be equal to the perpendicular distances for each of the force components from the two joints (considering the force acting on link one as the reaction force at the joint between links 1 and 2), and thus the torques produced by the end-effector force are given by $\tau = J^T F$, where T denotes the transpose. Once the torque vector has been evaluated, it is added on to the joint torques computed previously for the trajectory, and thus the angular acceleration is now obtained as $\ddot{\theta}_i = M^{-1}(\tau_{joint} - C(\theta)\dot{\theta}^2 + \tau_{FF})$. We can see from the figure that the force-field causes the trajectory to veer off to the side and curve clockwise at the end in all cases, and the effects appear to be more pronounced for trajectories closer to $3\pi/4$ or $7\pi/4$, especially for the out-and-back case. For all cases, the force-field causes the trajectory to curve clockwise and then return to a position close to the starting value of one coordinate. For the out-and-back case, this process appears to be repeated, causing a spiral trajectory.



(a) Outward.

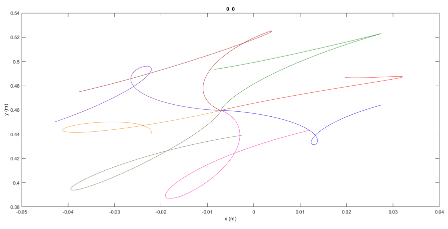


(b) Out-and-Back.

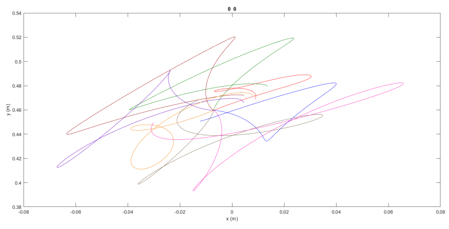
Figure 5: Effects of Velocity-Dependent Perturbation.

2.2 b)

We next consider adaptation to this velocity-dependent force-field by introducing a preprogrammed torque term that seeks to counteract the force-field. This term is introduced by taking the velocity profile of each spatial coordinate

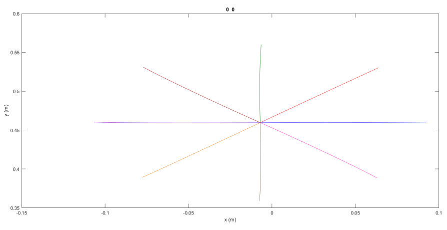


(a) Outward.

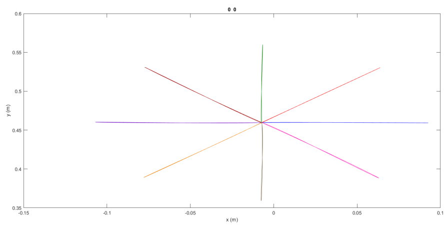


(b) Out-and-Back.

Figure 6: 50% Adaptation to Velocity-Dependent Perturbation.



(a) Outward.



(b) Out-and-Back.

Figure 7: 100% Adaptation to Velocity-Dependent Perturbation.

(obtained as $diff(x)/dt$ or $diff(y)/dt$, obtained from the returned trajectories from the initial cumulative squared jerk optimization function) and scaling it by the force-field coefficient matrix and some other adaptation factor, either 0.5 or 1.0. This adaptation force time series is fed into the simulation function and is subtracted from the computed force-field at each point and the difference between the two forces is then converted to a torque as described above. We can see that a 50% adaptation is a noticeable improvement, especially for the out-and-back case, though the trajectories still deviate substantially from their intended target. Now, these deviations appear closer to linear rather than cyclical. As expected, a 100% adaptation completely removes the influence of the force-field (up to some very small deviations due to numerical integration error).

3 Question 3: Addition of Stiffness and Viscosity

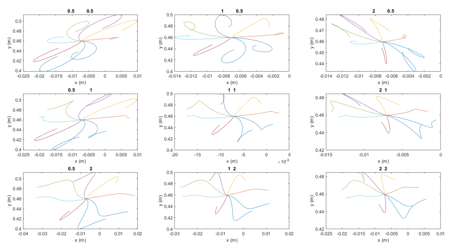
Finally, we consider the influence of stiffness and viscosity on the previous trajectories without any force-field adaptation. Stiffness is introduced as an angle-dependent parameter with coefficient matrix (with units of Nm/rad) K =

$$\begin{bmatrix} 15 & 6 \\ 6 & 16 \end{bmatrix}$$

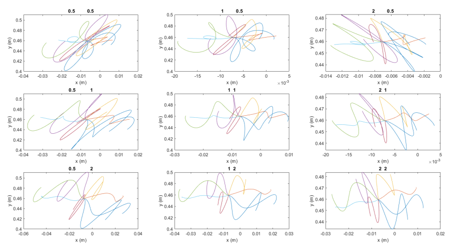
Similarly, viscosity is introduced as an angular velocity-dependent parameter with coefficient matrix (with units of $Nm/rad-s$) B =

$$\begin{bmatrix} 2.25 & 0.9 \\ 0.9 & 2.4 \end{bmatrix}$$

These contributions are introduced in an analogous way to the velocity-dependent force-field. While in the initial implementation of stiffness and viscosity in the dynamics simulation, for each iteration, the current angle vector, θ, was multiplied by the K matrix, and the current angular velocity vector, $\dot{\theta}$, was multiplied by the B matrix, it was found that the large magnitude of the absolute angles in conjunction with the large coefficients of the stiffness matrix produced trajectories that were wholly dominated by stiffness for all directions. Instead, an implementation was constructed based on the equilibrium point hypothesis, which states that a trajectory is implemented as a smoothly changing equillibrium point along the optimal path, and stiffness acts within this contex as a function of the difference in angle between the current true angle and the ideal angle along the optimal trajectory. Therefore, the stiffness matrix was multiplied by the difference between these two by adding in the vector of ideal shoulder and elbow angles as an input to the simulate dynamics function. At each iteration of the loop, the current angle is subtracted from the same angle vector from the ideal path and multiplied by the stiffness matrix, and the current angular velocity is multiplied by the viscosity matrix. The total joint torque which would be required to account for



(a) Outward.



(b) Out-and-Back.

Figure 8: Addition of Stiffness and Viscosity.

stiffness and viscosity is given by $\tau = M\ddot{\theta} + C\dot{\theta}^2 + K(\theta - \theta_{ideal}) + B\dot{\theta}$, and thus the angular acceleration is given as $\ddot{\theta} = M^{-1}(\tau_{joint} - C\dot{\theta}^2 - K(\theta - \theta_{ideal}) - B\dot{\theta} + \tau_{FF})$ for each iteration. We consider nine different combinations of stiffness and viscosity corresponding to independently adjusting a coefficient in front of the K and B matrices between values of 0.5, 1.0, and 2.0. We plot the results for both outward and out-and-back simulations. The title of each plot specifies the coefficients used for the B and K matrices in that order.

Clearly, adding these parameters substantially changes the trajectories. Large stiffness coefficients even appear to flip the trajectory from being biased to upward and rightward directions to upward and leftward directions. Intermediate stiffness and viscosity coefficients within the tested range appear to produce the most optimal trajectories, best following the desired direction without a substantial amount of perpendicular deviation. Even so, the value of the total displacement is an order of magnitude smaller than intended in this case, with values only changing by 1-4 cm instead of the desired 10 cm. The out-and-back trends are even more chaotic, though we can see qualitative similarity in the effect of stiffness (i.e. the leftward shift) and increasing linearity with larger stiffness and viscosity, with concomitant decrease in total displacement.

As a final note, similar curl force fields with velocity dependence have been used in the literature, and the results are rather different, even without stiffness and viscosity contributions. It appears that we would expect to see a path perturbed counterclockwise but without the drastic backward displacement we observe here for the outward only case. In other words, for a direct upward trajectory, the path is perturbed to the left and curves back toward the final position, rather than turning all the way back around and curving back toward the start point. It is possible that this is a consequence of the magnitude of the coefficients for the mass and Coriolis matrices, given that the units for these values do not yield those of torque when inserted into the dynamical equation. Here, these coefficients, and thus the matrices themselves, have units of N/m^2. Both matrices are multiplied by values with units of s^{-2}, either squared angular velocity or angular acceleration. This yields units of N/m^2s^2, rather than Nm, the units of torque. Thus, it is possible that some other factor or unit conversion is necessary here, as a preliminary test obtained by scaling these coefficients up by a factor of 10 yields trajectories qualitatively similar to those in the literature, for example the trials shown in Darainy *et al.* (Journal of Neurophysiology, 2009), Herzfeld *et al.* (Science, 2014), Li *et al.* (Neuron, 2001), or Huang *et al.* (Journal of Neuroscience, 2012). In all of these studies, a curl force field was used and the perturbation induces a leftward shift in the trajectories which curves back toward the target at the end. It is also possible, of course, that this is due to an alternative source of error, either in the form of incorrect model parameters, an error in simulation, or an incomplete model.

4 Appendix A: Derivation for Minimum Jerk Trajectory

As mentioned in the question, use of the calculus of variations to find an optimal function that minimizes the cumulative squared jerk, the third time derivative of position, reveals a fifth order polynomial for position:

$$x(t) = c_5t^5 + c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0$$

This method is based on constructing an arbitrary perturbation of $x(t) \to x(t) + \varepsilon\eta(t)$, and differentiating the cost function, $H(x(t))$ with respect to the perturbation parameter ε. Evaluating this derivative for the cost function $H(x(t)+\varepsilon\eta(t)) = \int(\dddot{x}(t)+\varepsilon\dddot{\eta}(t))^2dt$ at $\varepsilon = 0$, using integration by parts three times so the integral is in terms of $\eta(t)$ rather than its derivatives (with boundary conditions such that $\eta(t)$ and all its derivatives are zero at the endpoints (t_0 and t_f)), setting this integral equal to zero, and using the fundamental theorem of the calculus of variations, i.e. that if $\int_a^b f(x)h(x)dx = 0$ for all continuous and bounded functions $h(x)$, then $f(x)$ is zero over the interval from a to b. Since $\eta(t)$ can be any arbitrary function, this implies that $x^{(VI)}(t) = 0$, and so $x(t)$ is a fifth-order polynomial. Using proper initial and boundary conditions, we can solve for the six coefficients in the polynomial.

Here, we use the boundary conditions $x(0) = v(0) = a(0) = v(t_f) = a(t_f) = 0$ and $x(t_f) = x_f$ for some smooth motion from an initial position of 0 at time zero, and a final position x_f after time t_f. We require the object to start and end at rest, hence the bounds on velocity, and remain at rest after, forcing acceleration to zero as well. In order to solve for the coefficients, we use these six boundary conditions to construct a matrix equation and invert this equation to isolate the coefficients. The velocity is given as the derivative of position, and acceleration is the derivative of velocity, so:

$$v(t) = 5c_5t^4 + 4c_4t^3 + 3c_3t^2 + 2c_2t + c_1$$
$$a(t) = 20c_5t^3 + 12c_4t^2 + 6c_3t + 2c_2$$

Plugging in the boundary conditions, we can factor out the coefficients and place them in a vector such that $\mathbf{T}\vec{C} = \vec{X}$, or:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ t_f^5 & t_f^4 & t_f^3 & t_f^2 & t_f & 1 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 & 2t_f & 1 & 0 \\ 20t_f^3 & 12t_f^2 & 6t_f & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_f \\ 0 \\ 0 \end{bmatrix}$$

Clearly, the first three rows of the matrix imply that the three lower-index coefficients are zero, i.e. $c_0 = c_1 = c_2 = 0$, so we can reduce the 6x6 matrix to a 3x3 matrix:

$$\begin{bmatrix} t_f^5 & t_f^4 & t_f^3 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 \\ 20t_f^3 & 12t_f^2 & 6t_f \end{bmatrix} \begin{bmatrix} c_5 \\ c_4 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_f \\ 0 \\ 0 \end{bmatrix}$$

We can obtain the coefficients by inverting this 3x3 matrix and multiplying by the vector on the right-hand side. To invert a 3x3 matrix, we first construct its matrix of minors, obtain its cofactor matrix by negating all values in even positions on odd rows and odd positions in even rows, transpose the cofactor matrix to obtain the adjoint, and divide this by the determinant. The matrix of minors here is obtained by finding the 2x2 determinant of the values that do not share a row or column with the position of interest. For example, the first entry at row 1, column 1, has the value $24t_f^4 - 36t_f^4 = -12t_f^4$. The matrix of minors is then:

$$\begin{bmatrix} -12t_f^4 & -30t_f^5 & -20t_f^6 \\ -6t_f^5 & -14t_f^6 & -8t_f^7 \\ -t_f^6 & -2t_f^7 & -t_f^8 \end{bmatrix}$$

The cofactor matrix is obtained by negating positions 2, 4, 6, and 8:

$$\begin{bmatrix} -12t_f^4 & 30t_f^5 & -20t_f^6 \\ 6t_f^5 & -14t_f^6 & 8t_f^7 \\ -t_f^6 & 2t_f^7 & -t_f^8 \end{bmatrix}$$

The adjoint matrix is its transpose:

$$\begin{bmatrix} -12t_f^4 & 6t_f^5 & -t_f^6 \\ 30t_f^5 & -14t_f^6 & 2t_f^7 \\ -20t_f^6 & 8t_f^7 & -t_f^8 \end{bmatrix}$$

Finally, the determinant of the initial matrix is $\det T = t_f^5(24t_f^4 - 36t_f^4) - t_f^4(30t_f^5 - 60t_f^5) + t_f^3(60t_f^6 - 80t_f^6) = -2t_f^9$, so the inverse of the matrix is:

$$\begin{bmatrix} 6t_f^{-5} & -3t_f^{-4} & 0.5t_f^{-3} \\ -15t_f^{-4} & 7t_f^{-3} & -t_f^{-2} \\ 10t_f^{-3} & -4t_f^{-2} & 0.5t_f^{-1} \end{bmatrix}$$

Thus, our complete polynomial is:

$$x(t) = 6t_f^{-5}x_f t^5 - 15t_f^{-4}x_f t^4 + 10t_f^{-3}x_f t^3$$

5 Appendix B: Derivation for Out-and-Back Minimum Jerk Trajectory

We can modify the above scenario for a case where we desire an "out and back" motion that returns to the initial starting point at time t_f. This can be best accomplished by decomposing the problem into two segments that are antisymmetric. The first segment is an outward motion from 0 to x_f from time 0 to $\frac{t_f}{2}$. The second segment follows the reverse trajectory and can be constructed by time-reversing the first segment. Clearly, the initial setup of this problem is the same as the case above with t_f set to $\frac{t_f}{2}$. Therefore, we can use the same functional form of the position, since our goal is still to minimize the cumulative squared jerk, but our boundary conditions will change. Thus, we have:

$$x(t) = c_5t^5 + c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0$$

with the constraints that $t_0 = 0, x(0) = v(0) = a(0) = 0$, $x(\frac{t_f}{2}) = x_f$, and $v(\frac{t_f}{2}) = 0$. For simplicity, we also set t_f in the above solutions to be equal to $\frac{t_f}{2}$, and then replace this later. Due to the antisymmetry, we still require the velocity at the midpoint to be zero, as a non-zero velocity would imply that the arm travelled too far beforehand (hence a negative velocity) or will travel too far afterwards (a positive velocity), and is thus not the ideal motion, especially with the time-reversal. The boundary condition on acceleration is less clear, however. While a positive acceleration is unlikely to give a more optimal solution, since this would drive the velocity to increase in the positive direction, a negative acceleration could potentially be more optimal than requiring the acceleration to be zero at the midpoint. Thus, our main task here is to minimize the cumulative squared jerk by finding the optimal acceleration at the midpoint. As before, we have the same expressions for our velocity and acceleration, and the first three equations still imply that the lower-index coefficients are zero due to the initial conditions. We then end up with an almost identical system of three equations:

$$\begin{bmatrix} t_f^5 & t_f^4 & t_f^3 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 \\ 20t_f^3 & 12t_f^2 & 6t_f \end{bmatrix} \begin{bmatrix} c_5 \\ c_4 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_f \\ 0 \\ a_{min} \end{bmatrix}$$

As our matrix is the same, we can use the same inverse we found earlier, so the coefficients we obtain are from the product:

$$\begin{bmatrix} 6t_f^{-5} & -3t_f^{-4} & 0.5t_f^{-3} \\ -15t_f^{-4} & 7t_f^{-3} & -t_f^{-2} \\ 10t_f^{-3} & -4t_f^{-2} & 0.5t_f^{-1} \end{bmatrix} \begin{bmatrix} x_f \\ 0 \\ a_{min} \end{bmatrix} = \begin{bmatrix} c_5 \\ c_4 \\ c_3 \end{bmatrix} = \begin{bmatrix} 6t_f^{-5}x_f + 0.5t_f^{-3}a_{min} \\ -15t_f^{-4}x_f - t_f^{-2}a_{min} \\ 10t_f^{-3}x_f + 0.5t_f^{-1}a_{min} \end{bmatrix}$$

We can then represent our position function as:

$$x(t) = (6t_f^{-5}x_f + 0.5t_f^{-3}a_{min})t^5 + (-15t_f^{-4}x_f - t_f^{-2}a_{min})t^4 + (10t_f^{-3}x_f + 0.5t_f^{-1}a_{min})t^3$$
$$x(t) = a_{min}(0.5t_f^{-3}t^5 - t_f^{-2}t^4 + 0.5t_f^{-1}t^3) + x_f(6t_f^{-5}t^5 - 15t_f^{-4}t^4 + 10t_f^{-3}t^3)$$

In order to minimize our cumulative squared jerk, we use the same cost function as for the above solution, that is $H(x(t)) = \int_0^{t_f} (\dddot{x}(t))^2 dt$. Using the above form of $x(t)$, we can construct the jerk as:

$$J(t) = \dddot{x}(t) = a_{min}(30t_f^{-3}t^2 - 24t_f^{-2}t + 3t_f^{-1}) + x_f(360t_f^{-5}t^2 - 360t_f^{-4}t + 60t_f^{-3})$$

Our problem is then to solve the equation:

$$\frac{d}{da_{min}}\int_0^{t_f}(a_{min}(30t_f^{-3}t^2-24t_f^{-2}t+3t_f^{-1})+x_f(360t_f^{-5}t^2-360t_f^{-4}t+60t_f^{-3}))^2dt=0$$

$$\int_0^{t_f}\frac{d}{da_{min}}(a_{min}(30t_f^{-3}t^2-24t_f^{-2}t+3t_f^{-1})+x_f(360t_f^{-5}t^2-360t_f^{-4}t+60t_f^{-3}))^2dt=0$$

$$2\int_0^{t_f}(a_{min}(30t_f^{-3}t^2-24t_f^{-2}t+3t_f^{-1})+x_f(360t_f^{-5}t^2-360t_f^{-4}t+60t_f^{-3}))(30t_f^{-3}t^2-24t_f^{-2}t+3t_f^{-1})dt=0$$

$$\int_0^{t_f}a_{min}(900t_f^{-6}t^4-2*720t_f^{-5}t^3+2*90t_f^{-4}t^2+576t_f^{-4}t^2-2*72t_f^{-3}t+9t_f^{-2})+$$
$$x_f(10800t_f^{-8}t^4-8640t_f^{-7}t^3+1080t_f^{-6}t^2-10800t_f^{-7}t^3+8640t_f^{-6}t^2-1080t_f^{-5}t+1800t_f^{-6}t^2-1440t_f^{-5}t+180t_f^{-4})dt=0$$

Evaluating the integral with respect to t yields the equation (evaluated between 0 and t_f):

$$a_{min}(180t_f^{-6}t^5-360t_f^{-5}t^4+60t_f^{-4}t^3+192t_f^{-4}t^3-72t_f^{-3}t^2+9t_f^{-2}t)+$$
$$x_f(2160t_f^{-8}t^5-2160t_f^{-7}t^4+360t_f^{-6}t^3-2700t_f^{-7}t^4+2880t_f^{-6}t^3-540t_f^{-5}t^2+600t_f^{-6}t^3-720t_f^{-5}t^2+180t_f^{-4}t)=0$$

Clearly, evaluating the above at 0 leads to 0, as every term contains t to some power. We can then plug in t_f, simplify, move the x_f term to the right, and divide through by the coefficients of a_{min} to obtain an expression for our acceleration.

$$a_{min}=-\frac{x_f(2160t_f^{-3}-2160t_f^{-3}+360t_f^{-3}-2700t_f^{-3}+2880t_f^{-3}-540t_f^{-3}+600t_f^{-3}-720t_f^{-3}+180t_f^{-3})}{(180t_f^{-1}-360t_f^{-1}+60t_f^{-1}+192t_f^{-1}-72t_f^{-1}+9t_f^{-1})}$$

$$a_{min}=-\frac{60x_ft_f^{-3}}{9t_f^{-1}}=-\frac{20x_f}{3t_f^2}$$

6 Appendix C: Inverse Kinematics Derivation

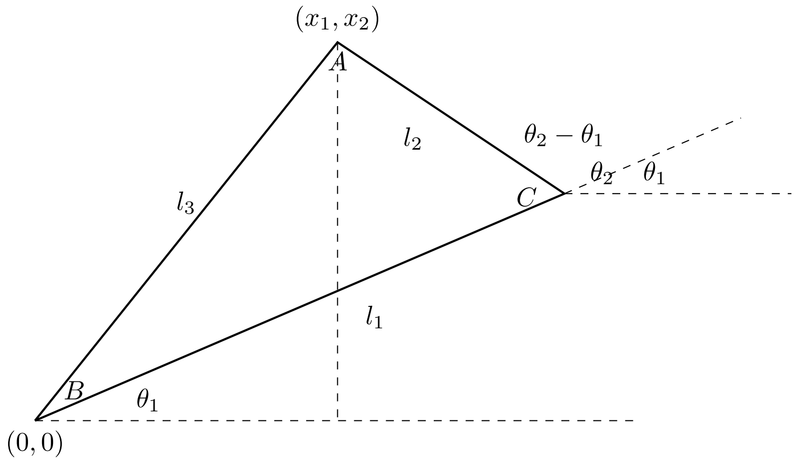


Figure 9: Inverse Kinematics Setup

In the above diagram, the length of the longest side is given by $l_3=\sqrt{x_1^2+x_2^2}$. Using the law of cosines, angle C can be found to be:

$$C=\arccos\frac{x_1^2+x_2^2-l_1^2-l_2^2}{-2l_1l_2}$$

As shown in the diagram, in the absolute configuration, the angle between the edge of length l_2 and the diagonal construction line extending from angle C is the difference between θ_2 and θ_1. This difference is also equal to $\pi - C$ radians, or:

$$\theta_2 = \theta_1 + \pi - \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1l_2}$$

Using the right triangle construction line, it can be seen that $B + \theta_1 = \arctan \frac{x_2}{x_1}$, and again, through the law of cosines, $B = \arccos \frac{l_2 - x_1^2 - x_2^2 - l_1^2}{-2l_1l_3}$. Thus,

$$\theta_1 = \arctan \frac{x_2}{x_1} - \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1l_3}$$

$$\theta_2 = \pi + \arctan \frac{x_2}{x_1} - \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1l_3} - \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1l_2}$$

7 Appendix D: Code

The first code presented is the overall workflow script, which loops through directions and coefficients, obtaining the optimized trajectories, corresponding angular coordinates, joint torques, and simulation results and plotting whichever sections are uncommented.

```
dirs = 0:45:315;
figure;
labels = zeros(length(dirs),1);
coeffs = [0.5 0.5; 1.0 0.5; 2.0 0.5; 0.5 1.0; 1.0 1.0; 2.0 1.0; 0.5 2.0; 1.0
   2.0; 2.0 2.0];
for cond = 1:9
    coeff = coeffs(cond,:);
for k = 1:length(dirs)
%Initialize angles and positions, specify total change in x and y, and
%define length of time
Dx = 10*cos(dirs(k)*pi/180);
Dy =10*sin(dirs(k)*pi/180);
tinit1 = 45*pi/180;
tinit2 = 135*pi/180;
l1 = 32;
l2 = 33;
[xinit,yinit] = ForwardKinematics(tinit1,tinit2,l1,l2);
Tf = 0.5;

%Use minimal cumulative squared jerk to find optimal trajectory
[x_,y_] = LCSJ(Dx, Dy, Tf);
x = x_+xinit;
y = y_+yinit;
dt = Tf/(length(x_)-1);
t = [0:dt:Tf];

%Convert optimal trajectory in x and y to shoulder and elbow angles
[theta1, theta2] = InverseKinematics(x,y,l1,l2);

%Verify conversion
[xtest,ytest] = ForwardKinematics(theta1,theta2,l1,l2);

%Initialize parameters and vectors for Torque Computation
```

```
a = 0.265;
b = 0.052;
c = 0.0844;
dth1 = diff(theta1)/dt;
dth2 = diff(theta2)/dt;
dtheta1 = [0 dth1];
dtheta2 = [0 dth2];
ddth1 = diff(theta1,2)/dt^2;
ddth2 = diff(theta2,2)/dt^2;
ddtheta1 = [0 ddth1 0];
ddtheta2 = [0 ddth2 0];

theta = [theta1;theta2];
dtheta = [dtheta1;dtheta2];
ddtheta = [ddtheta1;ddtheta2];

JT = Dynamics(theta, dtheta, ddtheta, a, b, c);
% figure;
% plot(t,JT(1,:));
% hold on
% %plot(t,JT(2,:));
% xlabel('Time (s)');
% ylabel('Torque (Nm)')

%Simulate dynamics based on calculated torques, initial conditions, and
%coefficients for force-field, stiffness, and viscosity
fc = [0 15; -15 0];
%fc = [0 0;0 0];
B = [2.25 0.9; 0.9 2.4];
K = [15 6; 6 15];

Fadapt = 0*fc*[diff(x);diff(y)]/dt;
[t12, xsim, dt12, ddt12, dx, ddx] = SimulateDynamics(JT, l1, l2, tinit1, tinit2,
    a,  b, c, dt, fc, coeff(1)*B, coeff(2)*K, Fadapt, theta);
%figure;
% subplot(2,1,1)
% plot(t,xsim(1,:));
% hold on
% ylabel('x (cm)');
% subplot(2,1,2)
% plot(t,xsim(2,:));
% xlabel('Time (s)');
% ylabel('y (cm)');
% hold on
%
% figure;
% subplot(2,1,1)
% plot(t,dt12(1,:));
% ylabel('dx/dt (cm/s)');
% subplot(2,1,2)
% plot(t,dt12(2,:));
% xlabel('Time (s)');
% ylabel('dy/dt (cm/s)');
%
% figure;
subplot(3,3,cond);
plot(xsim(1,:),xsim(2,:));
```

```
hold on

end
% legend({'0','45','90','135','180','225','270','315'})
xlabel('x (cm)');
ylabel('y (cm)');
%title(num2str(coeff));

% subplot(2,1,1)
% legend({'0','45','90','135','180','225','270','315'})
% title('x Motion');
%
% subplot(2,1,2)
% legend({'0','45','90','135','180','225','270','315'})
% title('y Motion');

end
```

The following function performs trajectory optimization on both spatial variables using the lsqnonlin function and returns the results.

```
function [x_,y_] = LCSJ(xfin, yfin, Tf)

N=1000;
x0=zeros(1,N);
y0=zeros(1,N);

csjx = @(x,xf) diff([0,0,0,x,xf,xf,xf],3);
csjy = @(y,yf) diff([0,0,0,y,yf,yf,yf],3);

x=lsqnonlin(@(x,xf) csjx(x,xfin), x0);
y=lsqnonlin(@(y,yf) csjy(y,yfin), y0);
x_ = [0,x,xfin];
y_ = [0,y,yfin];
disp(length(x_))
t=[0:(N+1)]/(N+1)*Tf;
dt=Tf/(N+1);
%figure;

% subplot(3,1,1);
% plot(t,x_);
% ylabel('Position');
% hold on;
%
% subplot(3,1,2);
% plot(t(2:end),diff(x_)/dt);
% ylabel('Velocity');
% hold on;
%
% subplot(3,1,3);
% plot(t(2:end-1),diff(x_,2)/dt^2);
% ylabel('Acceleration');
% xlabel('Time (sec)');
% hold on;
%
% hold off;
%
% figure;
% subplot(3,1,1);
% plot(t,y_);
% ylabel('Position');
% hold on;
%
% subplot(3,1,2);
% plot(t(2:end),diff(y_)/dt);
% ylabel('Velocity');
% hold on;
%
% subplot(3,1,3);
% plot(t(2:end-1),diff(y_,2)/dt^2);
% ylabel('Acceleration');
% xlabel('Time (sec)');
% hold on;
%
% hold off;
%
% figure;
% plot(x,y);
```

Like the previous function, LCSJOAB performs trajectory optimization for mimimum cumulative squared jerk, but in this case, the acceleration is not constrained at the endpoint and the trajectory is time-reversed and concatenated to form an out-and-back motion.

```
        function [x_,y_] = LCSJOAB(xfin, yfin, Tf)

N=1000;
x0=zeros(1,N);
y0=zeros(1,N);

csjx = @(x,xf) diff([0,0,0,x,xf,xf],3);
csjy = @(y,yf) diff([0,0,0,y,yf,yf],3);

x=lsqnonlin(@(x,xf) csjx(x,xfin), x0);
y=lsqnonlin(@(y,yf) csjy(y,yfin), y0);
x_ = [0,x,xfin];
y_ = [0,y,yfin];
t=[0:(N+1)]/(N+1)*Tf;
dt=Tf/(N+1);
% figure;
%
t=[t,t(2:end)+t(end)];
x_=[x_,flip(x_(2:end))];
y_=[y_,flip(y_(2:end))];
%
% subplot(3,1,1);
% plot(t,x_);
% ylabel('Position');
% hold on;
%
% subplot(3,1,2);
% plot(t(2:end),diff(x_)/dt);
% ylabel('Velocity');
% hold on;
%
% subplot(3,1,3);
% plot(t(2:end-1),diff(x_,2)/dt^2);
% ylabel('Acceleration');
% xlabel('Time (sec)');
% hold on;
%
% hold off;
%
% figure;
% subplot(3,1,1);
% plot(t,y_);
% ylabel('Position');
% hold on;
%
% subplot(3,1,2);
% plot(t(2:end),diff(y_)/dt);
% ylabel('Velocity');
% hold on;
%
% subplot(3,1,3);
% plot(t(2:end-1),diff(y_,2)/dt^2);
% ylabel('Acceleration');
% xlabel('Time (sec)');
% hold on;
%
% hold off;
%
% figure;
% plot(x,y);
```

This short function performs the forward kinematic calculations for obtaining end-effector coordinates from angles.

```
        function [x,y] = ForwardKinematics(theta1_list,theta2_list,l1,l2)

x=zeros(1,length(theta1_list));
y=x;

for i = 1:length(theta1_list)
    theta1=theta1_list(i);
    theta2=theta2_list(i);

    x(i) = l1*cos(theta1)+l2*cos(theta2);
    y(i) = l1*sin(theta1)+l2*sin(theta2);

end
```

Likewise, the inverse kinematic function obtains joint angles corresponding to specific end-effector coordinates.

```
        function [theta1, theta2] = InverseKinematics(x1_vec,x2_vec,l1,l2)

theta1 = zeros(1,length(x1_vec));
theta2 = zeros(1,length(x1_vec));

for i = 1:length(x1_vec)
    x1 = x1_vec(i);
    x2 = x2_vec(i);
    l3=sqrt(x1^2+x2^2);
    C=(l2^2-x1^2-x2^2-l1^2)/(-2*l1*l3);
    CC=(x1^2+x2^2-l1^2-l2^2)/(-2*l1*l2);
    theta1(i)=atan2(x2,x1)-acos(C);
    theta2(i)=pi+theta1(i)-acos(CC);
end
```

The dynamics function obtains the joint torques at each time point corresponding to the vectors obtained from trajectory optimization and inverse kinematics. The spatial trajectories and their first and second derivative are fed into the function.

```
        function [JT] = Dynamics(theta,dtheta,ddtheta,a,b,c)

JT = zeros(2,length(theta));
for i = 1:length(theta)
        th = theta(:,i);
        dth = dtheta(:,i);
        ddth = ddtheta(:,i);
        M = [a c*cos(th(2)-th(1)); c*cos(th(2)-th(1)) b];
        C = [0 -c*sin(th(2)-th(1)); c*sin(th(2)-th(1)) 0];

        JT(:,i) = M*ddth + C*dth.^2;
end
```

Finally, the simulation function takes in joint torques for each time point, initial conditions and parameters for inertia, Coriolis terms, viscosity, stiffness, applied force-fields, and adaptation forces and evaluates the dynamics by solving the dynamic equation for acceleration and updating angular velocity, angle, position, velocity, and acceleration based on the results.

```
        function [t12, x, dt12, ddt12, dx, ddx] = SimulateDynamics(JT, l1, l2,
           t10, t20, a, b, c, dt, fc, B, K, Fadapt, theta)

%inputs:
% JT = 2 x N vector of joint torques to be applied
% l1, l2 are used to compute x and y from thetas
% t10 and t20 are initial conditions for theta
% a, b, and c are coefficients for the interia matrix M
% B and K are stiffness and viscosity matrices
% fc are the coefficients for the applied force-field

N=length(JT);
t12 = zeros(2,N);
dt12 = t12;
ddt12 = t12;
x=t12;dx=t12;ddx=t12;
f = t12;
t12(1,1) = t10;
t12(2,1) = t20;
[x(1,1),x(2,1)] = ForwardKinematics(t12(1,1),t12(2,1),l1,l2);
dx(1,1)=0;dx(2,1)=0;ddx(1,1)=0;ddx(2,1)=0;

for i = 1:N-1
       J = [-l1*sin(t12(1,i)) -l2*sin(t12(2,i)) ; l1*cos(t12(1,i)) l2*cos(t12(2,
          i))];
        f(:,i) = fc*dx(:,i)-Fadapt(:,i);
        tforce = transpose(J)*f(:,i);
        M = [a c*cos(t12(2,i)-t12(1,i)); c*cos(t12(2,i)-t12(1,i)) b];
        C = [0 -c*sin(t12(2,i)-t12(1,i)); c*sin(t12(2,i)-t12(1,i)) 0];
        ddt12(:,i) = inv(M)*(JT(:,i)-K*(t12(:,i)-theta(:,i))-B*dt12(:,i)+tforce-
           C*dt12(:,i).^2);
        dt12(:,i+1) = dt12(:,i) + ddt12(:,i)*dt;
        t12(:,i+1) = t12(:,i) + dt12(:,i)*dt + 0.5*ddt12(:,i)*dt^2;
        [x(1,i+1),x(2,i+1)] = ForwardKinematics(t12(1,i+1),t12(2,i+1),l1,l2);
        dx(:,i+1) = (x(:,i+1)-x(:,i))/dt;
        ddx(:,i+1) = (dx(:,i+1)-dx(:,i))/dt;
end

% plot(f(1,:))
% hold on
% title('Force');
```