CONTROL, SENSOR, NETWORK AND PERCEPTION LABORATORY

Baxter Left and Right Arm Modeling Using Equations of Motion (Model C)

Technical Report # 2025 - 001

Jack Bannan, Chang-Hee Won
Temple University
CSNAP Lab
Last Modified: 03/07/2025

1. Overview Description

This technical report provides a detailed description of the modeling of the Baxter Robot's right and left arms found in [1]. The document goes over the alternative derivation of the Baxter using the equations of motion. This model is dubbed "Baxter Model C" as it was derived to fix some inconsistencies in the previous model developed in [2]. Here presented is the derivation of the ***left arm*** of the Baxter robot. The right arm has the same derivation, however its numerical computation in the code is slightly different to do different parameters, thus one should make proper edits to the code found in the appendix with the parameters for the right arm in order to do those calculations.

2. Derivation of Continuous Baxter Model C

Model C was developed using nonlinear equation for a robot arm given below in (1) found in [3, pp 191] as:

$$\tau\big(q(t),\dot{q}(t),\ddot{q}(t)\big) = M\big(q(t)\big)\ddot{q}(t) + C\big(q(t),\dot{q}(t)\big)\dot{q}(t) + F\big(\dot{q}(t)\big) + G\big(q(t)\big) + J^T\big(q(t)\big)g \qquad (1)$$

where τ(t) is the vector of joint torques, $q(t)$ is the vector of joint angles, $M\big(q(t)\big)$ is the joint-space inertia matrix, $C\big(q(t),\dot{q}(t)\big)$ is the Coriolis and centripetal coupling matrix, $F\big(\dot{q}(t)\big)$ is the friction force, $G\big(q(t)\big)$is the gravity force, g is the external wrench due to end effector tool, and $J\big(q(t)\big)$ is the manipulator Jacobian [3]. We then make the following assumptions

1. The Baxter robot is in torque control mode (see Chapter 1) and so the control inputs $u(t)$ is the joint torques.

2. The Baxter robot is in torque control mode, which includes online gravity compensation, so we may drop the gravity term, $G\big(q(t)\big)$, from the nonlinear model.

3. The robot is free to move in space, so no external wrench forces, g, or torques are enacted on the arm.

4. Viscous friction in the joint is minimal due to the Series Elastic Actuators and joint velocities are relatively small, so the effects of friction, $F\big(\dot{q}(t)\big)$, may be neglected [4].

Thus, the nonlinear equation of motion for the Baxter robot in torque control mode can be reduced to

$$\tau\big(q(t),\dot{q}(t),\ddot{q}(t)\big) = M\big(q(t)\big)\ddot{q}(t) + C\big(q(t),\dot{q}(t)\big)\dot{q}(t). \qquad (2)$$

We also define the homogenous transformation matrix from base frame, 0, to the n^{th} frame as the following [5, p 78, 111]:

$$T_n^0\big(q(t)\big) = \begin{bmatrix} R_n^0\big(q(t)\big) & o_n^0\big(q(t)\big) \\ 0 & 1 \end{bmatrix}, \qquad (3)$$

where we denote the position and orientation of the end effector with respect to the base frame by $o_n^0(q(t))$, which is the origin of the end effector position with respect to the base frame, and the 3x3 rotation matrix $R_n^0(q(t))$. We also note that the transformation matrix of one frame to itself is a 4x4 identity matrix [5, p 77].

We can calculate the homogeneous transformation matrix by the given product assuming the Baxter robot only has revolute joints [5, p77]:

$$T_n^0(q(t)) = A_1(q_1(t)) \dots A_n(q_n(t)), \tag{4}$$

where each homogenous transformation matrix, $A_i(q_i(t))$, for link $i = 1, \dots, n$ is given as [5, p 79]

$$A_i(q_i(t)) = \begin{bmatrix} R_i^{i-1}(q_i(t)) & o_i^{i-1}(q_i(t)) \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(q_i(t)) & -\sin(q_i(t))\cos(\alpha_i) & \sin(q_i(t))\sin(\alpha_i) & a_i\cos(q_i(t)) \\ \sin(q_i(t)) & \cos(q_i(t))\cos(\alpha_i) & -\cos(q_i(t))\sin(\alpha_i) & a_i\sin(q_i(t)) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5}$$

where $q_i(t), a_i, d_i, \alpha_i$ are the joint angle, link length, link offset, and link twist respectively and are the Denavit-Hartenberg (DH) parameters associated with link *i*.

We define the nxn joint space inertia matrix, $M(q(t))$, as the following [5, pp 180]

$$M(q(t)) = \left[\sum_{i=1}^{n} \left\{ m_i J_{v_{ci}}(q(t))^T J_{v_{ci}}(q(t)) + J_{w_{ci}}(q(t))^T R_i^0(q(t)) I_i R_i^0(q(t))^T J_{w_{ci}}(q(t)) \right\}\right], \tag{6}$$

where $m_i$is the mass of link *i*, $J_{v_{ci}}(q(t))$ is the Jacobian matrix for the linear velocity of the center of mass of link i, $J_{w_{ci}}(q(t))$ is the Jacobian matrix for the angular velocity of the center of mass of link *i*, $R_i^0(q(t))$ is the transformation rotation matrix for link *i*, I_i is the inertia matrix of link *i*, evaluated around coordinate frame parallel to *i*, but whose origin is at the center of mass.

The angular Jacobian of the center of mass for each link, $J_{w_{ci}}(q(t))$ is defined as the following as all the Baxter joints are revolute joints [5, pp 155]:

$$J_{w_{ci}}(q(t)) = [J_{w_{ci1}}(q(t)), \dots, J_{w_{cin}}(q(t))]. \tag{7}$$

Where the *jth* column, *j = 1, ..., n*, of (7) is defined as:

$$J_{w_{cij}}(q(t)) = \begin{matrix} p_j z_{j-1}^0(q(t)), & j \le i, \\ 0, & j > i. \end{matrix} \tag{8}$$

Where i is the revolute joint and 0 refers to the base frame, and $p_i = 1$, for revolute joints. Thus, we can simplify the Jacobian to the following for joint i [5, pp 112, 116]:

$$J_{wc_{ij}}(q(t)) = \begin{matrix} z_{j-1}^{0}\,(q(t)), & j \le i \\ 0, & j > i, \end{matrix} \tag{9}$$

where

$$z_{j-1}^{0}(q(t)) = R_{j-1}^{0}(q(t))K, \tag{10}$$

where K is the unit coordinate vector defined as $K = [0, 0, 1]^T$ and $z_0^0 = K$.

We also define the angular velocity Jacobian at the center of mass, $J_{v_{ci}}(q(t))$, similarly as:

$$J_{v_{ci}}(q(t)) = \left[J_{v_{ci1}}(q(t)), \ldots, J_{v_{cin}}(q(t))\right]. \tag{11}$$

Since all the joints of the Baxter are revolute, the *jth* column, where *j = 1, ..., n,* of the linear Jacobian for each link i at the center of mass is defined as [5, pp 115 - 116]:

$$J_{v_{cij}}(q(t)) = \begin{matrix} z_{j-1}^{0}(q(t)) \times \left(o_{ci}^{0}(q(t)) - o_{j-1}^{0}(q(t))\right), & j \le i, \\ 0, & j > i, \end{matrix} \tag{12}$$

with $\times$ denoting the cross product and $o_{ci}^{0}(q(t))$ defined as the position of center of mass of link *i w.r.t* the base frame.

We define the $(k, j)^{th}$element of the Coriolis and centrifugal coupling matrix, $C(q(t), \dot{q}(t))$, as

$$C(q(t), \dot{q}(t)) = [c_{kj}], \tag{13}$$

$$c_{kj} = \sum_{i=1}^{n} \frac{1}{2}\left\{\frac{\partial m_{kj}}{\partial q_i} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{ij}}{\partial q_k}\right\}\dot{q}, \tag{14}$$

where $m_{i,j}$ are the entries of the *n x n* inertia matrix, *M(q),* and *k, j, i = 1, ..., n.*

Linearization around an operating point $Q_0 = [q_0, \dot{q}_0, \ddot{q}_0]$ is done through Taylor series expansion. The first-order Taylor Series expansion for three variables is given as

$$f(x, y, z) \approx f(x_0, y_0, z_0) + \left.\frac{\partial f}{\partial x}\right|_{x_0, y_0, z_0}(x - x_0) + \left.\frac{\partial f}{\partial y}\right|_{x_0, y_0, z_0}(y - y_0) + \left.\frac{\partial f}{\partial z}\right|_{x_0, y_0, z_0}(z - z_0). \tag{15}$$

Thus, for (2), we have the following first order Taylor series expansion, we also omit the t term in $q(t), \dot{q}(t), \ddot{q}(t)$ for writing simplicity:

$$\tau(q,\dot{q},\ddot{q}) \approx \tau(q_0,\dot{q}_0,\ddot{q}_0) + \left.\frac{\partial\tau}{\partial q}\right|_{Q_0}(q-q_0) + \left.\frac{\partial\tau}{\partial\dot{q}}\right|_{Q_0}(\dot{q}-\dot{q}_0) + \left.\frac{\partial\tau}{\partial\ddot{q}}\right|_{Q_0}(\ddot{q}-\ddot{q}_0). \tag{16}$$

We also define the following:

$$\delta\tau(q,\dot{q},\ddot{q}) = \tau(q,\dot{q},\ddot{q}) - \tau(q_0,\dot{q}_0,\ddot{q}_0) \tag{17}$$

$$\delta q = (q-q_0) \tag{18}$$

$$\delta\dot{q} = (\dot{q}-\dot{q}_0) \tag{19}$$

$$\delta\ddot{q} = (\ddot{q}-\ddot{q}_0). \tag{20}$$

Using algebraic manipulation and substituting (15) – (18) into (14), we obtain (19):

$$\delta\tau(q,\dot{q},\ddot{q}) \approx \left.\frac{\partial\tau}{\partial q}\right|_{Q_0}\delta q + \left.\frac{\partial\tau}{\partial\dot{q}}\right|_{Q_0}\delta\dot{q} + \left.\frac{\partial\tau}{\partial\ddot{q}}\right|_{Q_0}\delta\ddot{q}. \tag{21}$$

We then can write the first term of (19) as the following using product rule:

$$\left.\frac{\partial\tau}{\partial q}\right|_{Q_0}\delta q = \left[\frac{\partial M(q)}{\partial q}\ddot{q} + \frac{\partial\ddot{q}}{\partial q}M(q) + \frac{\partial C(q,\dot{q})}{\partial q}\dot{q} + \frac{\partial\dot{q}}{\partial q}C(q,\dot{q})\right]_{Q_0}\delta q. \tag{22}$$

Since $q,\dot{q},\ddot{q}$ are independent, the second term and fourth term of (20) is zero and thus we can simplify (20) as:

$$\left.\frac{\partial\tau}{\partial q}\right|_{Q_0}\delta q = \left[\frac{\partial M(q)}{\partial q}\ddot{q} + \frac{\partial C(q,\dot{q})}{\partial q}\dot{q}\right]_{Q_0}\delta q. \tag{23}$$

Similarly, we write the second term of (19) using product rule as:

$$\left.\frac{\partial\tau}{\partial\dot{q}}\right|_{Q_0}\delta\dot{q} = \left[\frac{\partial M(q)}{\partial\dot{q}}\ddot{q} + \frac{\partial\ddot{q}}{\partial\dot{q}}M(q) + \frac{\partial C(q,\dot{q})}{\partial\dot{q}}\dot{q} + \frac{\partial\dot{q}}{\partial\dot{q}}C(q,\dot{q})\right]_{Q_0}\delta\dot{q}. \tag{24}$$

We simplify (22) to the following:

$$\left.\frac{\partial\tau}{\partial\dot{q}}\right|_{Q_0}\delta\dot{q} = \left[\frac{\partial C(q,\dot{q})}{\partial\dot{q}}\dot{q} + C(q,\dot{q})\right]_{Q_0}\delta\dot{q}. \tag{25}$$

For the third term of (19) we write it as the following:

$$\left.\frac{\partial\tau}{\partial\ddot{q}}\right|_{Q_0}\delta\ddot{q} = \left[\frac{\partial M(q)}{\partial\ddot{q}}\ddot{q} + \frac{\partial\ddot{q}}{\partial\ddot{q}}M(q) + \frac{\partial C(q,\dot{q})}{\partial\ddot{q}}\dot{q} + \frac{\partial\dot{q}}{\partial\ddot{q}}C(q,\dot{q})\right]_{Q_0}\delta\ddot{q}. \tag{26}$$

We simplify (24) to the following:

$$\left.\frac{\partial \tau}{\partial \ddot{q}}\right|_{Q_0} \delta\ddot{q} \;=\; M(q)|_{Q_0}\delta\ddot{q}. \tag{27}$$

For small deviations around operating point the linearized for the of the manipulator equations of motion can be represented as [6, pp 223-225]

$$\delta\tau(q,\dot{q},\ddot{q}) = D_0\delta\ddot{q} + V_o\delta\dot{q} + P_0\delta q, \tag{28}$$

where $\delta\tau, \delta\ddot{q}, \delta\dot{q}, \delta q \;\in\; R^n$ are small perturbations around the operating point Q_0 and D_0, V_0, P_0 are known as the coefficient joint sensitivity matrices. The coefficients of the tensors of the linearized equation are computed using Taylor series expansion as

$$D_0 = \left.\frac{\partial \tau}{\partial \ddot{q}}\right|_{Q_0} = M(q)|_{Q_0}, \tag{29}$$

$$V_0 = \left.\frac{\partial \tau}{\partial \dot{q}}\right|_{Q_0} = \left[\frac{\partial C(q,\dot{q})}{\partial \dot{q}}\dot{q} + C(q,\dot{q})\right]_{Q_0}, \tag{30}$$

$$P_0 = \left.\frac{\partial \tau}{\partial q}\right|_{Q_0} = \left[\frac{\partial M(q)}{\partial q}\ddot{q} + \frac{\partial C(q,\dot{q})}{\partial q}\dot{q}\right]_{Q_0}. \tag{31}$$

Thus, substituting the results of Taylor (21), (23) and (25) into (19) we obtain the result the Taylor series expansion of (19) in (26).

Since $D_0$is (27) is positive definite, we can write the state space model [6, p 225].
Where,

$$\delta\dot{q} = I_{n\times n}\delta\dot{q}. \tag{32}$$

And using algebraic manipulation write (26) to define $\delta\ddot{q}$ as:

$$\delta\ddot{q} = \; D_0^{-1}(\delta\tau(q,\dot{q},\ddot{q}) - V_o\delta\dot{q} - P_0\delta q). \tag{33}$$

Thus, the model is given as:

$$\begin{bmatrix}\delta\dot{q}\\ \delta\ddot{q}\end{bmatrix} = \begin{bmatrix}0_{n\times n} & I_{n\times n}\\ -D_0^{-1}P_0 & -D_0^{-1}V_0\end{bmatrix}\begin{bmatrix}\delta q\\ \delta\dot{q}\end{bmatrix} + \begin{bmatrix}0_{n\times n}\\ D_0^{-1}\end{bmatrix}\delta\tau \tag{34}$$

Where $0_{n\times n}$ and $I_{n\times n}$ are *nxn* zero and identity matrices respectively.

In state space form:

$$\dot{x} = Ax(t) + Bu(t), \tag{35}$$

where,

$$x = \begin{bmatrix} \delta\dot{q} \\ \delta\ddot{q} \end{bmatrix},$$

$$A = \begin{bmatrix} 0_{n\times n} & I_{n\times n} \\ -D_0^{-1}P_0 & -D_0^{-1}V_0 \end{bmatrix},$$

$$B = \begin{bmatrix} 0_{n\times n} \\ D_0^{-1} \end{bmatrix},$$

$$u(t) = \delta\tau, \tag{36}$$

where $u(t)$ are the joint torques.

The Baxter robot has compliant joints which show additive noise present in joint angle measurements [7]. We account for this noise by adding a zero-mean gaussian white term dw with covariance W to our model with

$$E\{dw(t)\} = 0, \tag{37}$$

$$E\{dw(t)dw^T(t)\} = W(t)dt. \tag{38}$$

This results in the following stochastic state space model:

$$\dot{x} = Ax(t) + Bu(t) + Gdw(t), \tag{39}$$

where *G* is a *14 x 1* noise coefficient matrix.

3. Example Model C with Linearization at Q_0

3.1 Left Arm

With the derivation of the Baxter Model in Section 2, we can now find an A and B matrix for a desired linearization point. As an example, we choose the linearization point defined in Eqs. (40) to (42) and find the A and B matrices of our model. We used the code found in the appendix for our calculations. The linearization point used for the left arm, $Q_{l0} = [q_{l0}, \dot{q}_{l0}, \ddot{q}_{l0}]$, is defined as:

$$q_{l0} = [2.1644, -1.469, 0.7140, -0.4878, 1.2732, -0.2274, -0.3152] \quad (40)$$

$$\dot{q}_{l0} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00] \quad (41)$$

$$\ddot{q}_{l0} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00] \quad (42)$$

Using the code found in Appendix B.2 and the procedure in A.1 and A.2, we can calculate the joint space inertia, $M\big(q(t)\big)$, and Corolis matrices, $C(q, \dot{q})$. The result of the calculation of the joint space inertia matrix, $M\big(q(t)\big)$, and Corolis matrix, $C(q, \dot{q})$ is found using Mathematica. Utilizing the above $M\big(q(t)\big)$ and $C(q, \dot{q})$ in Eqs. (29) – (31) we numerically obtain D_0, V_0, P_0 in the following matrices for the linearization point, Q_0:

$$D_{l0} = \begin{bmatrix} 125.8 & 60.32 & 37.40 & 19.85 & 12.82 & -5.074 & 0.0872 \\ 60.32 & 44.41 & 18.01 & 19.84 & 7.387 & 0.1179 & 0.0651 \\ 37.40 & 18.01 & 23.97 & 6.680 & 7.781 & -1.534 & -0.0202 \\ 19.85 & 19.84 & 6.680 & 15.58 & 3.929 & 3.008 & -0.0357 \\ 12.82 & 7.387 & 7.781 & 3.929 & 5.655 & 0.2345 & -0.0015 \\ -5.074 & 0.1179 & -1.534 & 3.008 & 0.2345 & 5.345 & 0.0164 \\ -0.0872 & -0.0651 & -0.0202 & -0.0357 & -0.0015 & 0.0164 & 0.0060 \end{bmatrix} \quad (43)$$

$$P_{l0} = 0_{7x7} \quad (45)$$

$$V_{l0} = 0_{7x7} \quad (46)$$

We can see from (44) that the matrix is symmetric, which is expected because the mass matrix is symmetric [5]. We can see from the numerical calculations that (45) and (46) are zero matrices. This makes sense as both (41) and (42) are zero vectors. Using (34-36) we find the following values for A:

$$A_l = \begin{bmatrix} 0_{7\times7} & I_{7\times7} \\ 0_{7\times7} & 0_{7\times7} \end{bmatrix} \quad (47)$$

And similarly for B:

$$B_l = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.049 & -0.0108 & -0.0100 & 0.0184 & -0.0320 & 0.0023 & 3.732 * 10^{-4} \\ -0.0108 & 0.1268 & -8.0768 * 10^{-4} & -0.1446 & 0.0294 & 0.0017 & -0.0061 \\ -0.0100 & -8.078 * 10^{-4} & 0.0526 & -0.0410 & -0.0023 & -0.0639 & 0.0648 \\ 0.0184 & -0.1446 & -0.0410 & 0.4373 & -0.0316 & -0.1647 & -0.0557 \\ -0.0320 & 0.0294 & -0.0023 & -0.0316 & 0.1471 & -0.0675 & -0.1351 \\ 0.0023 & 0.0017 & -0.0639 & -0.1647 & -0.0675 & 0.7070 & -2.3869 \\ 3.732 * 10^{-4} & -0.0061 & 0.0648 & -0.0557 & -0.1351 & -2.386 & 223.1852 \end{bmatrix} \quad (48)$$

3.1 Right Arm

We also repeat this process for the right arm at a different linearization point given as: $Q_{r0} = [q_{r0}, \dot{q}_{r0}, \ddot{q}_{r0}]$, is defined as:

$$q_{r0} = [-0.2534, -0.0233, 0.2193, 1.389, -0.3570, 1.1938, 1.2655] \quad (49)$$

$$\dot{q}_{r0} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00] \quad (50)$$

$$\ddot{q}_{r0} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00] \quad (51)$$

Using the same procedure as the left arm we obtain the following matrices:

$$D_{r0} = \begin{bmatrix} 65.53 & 16.18 & 50.44 & 16.30 & 14.41 & 7.031 & 0.0430 \\ 16.18 & 16.46 & 16.15 & 8.694 & 6.787 & 3.560 & 0.0250 \\ 50.44 & 16.15 & 50.45 & 16.44 & 14.26 & 7.065 & 0.0439 \\ 16.30 & 8.694 & 16.44 & 10.15 & 5.386 & 3.958 & 0.0308 \\ 14.41 & 6.787 & 14.26 & 5.386 & 11.01 & 2.743 & 0.01158 \\ 7.031 & 3.560 & 7.065 & 3.958 & 2.743 & 2.701 & 0.02358 \\ 0.0430 & 0.0250 & 0.0439 & 0.0308 & 0.01158 & 0.02358 & 0.0046 \end{bmatrix} \quad (52)$$

$$P_{r0} = 0_{7x7} \quad (53)$$

$$V_{r0} = 0_{7x7} \quad (54)$$

With the corresponding A and B matrices as:

$$A_r = \begin{bmatrix} 0_{7\times7} & I_{7\times7} \\ 0_{7\times7} & 0_{7\times7} \end{bmatrix} \quad (55)$$

And similarly for B:

$$B_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0662 & -6.1714 * 10^{-4} & -0.0664 & 0.0025 & -0.0016 & -9.616 * 10^{-5} & 0.0062 \\ -6.1714 * 10^{-4} & 0.1188 & -0.0029 & -0.0808 & -0.0293 & 8.367 * 10^{-4} & -0.0010 \\ -0.0664 & -0.0029 & 0.1166 & -0.0596 & -0.0306 & -0.0103 & 0.0526 \\ 0.0025 & -0.0808 & -0.0596 & 0.3685 & 0.0191 & -0.3038 & 0.0252 \\ -0.0016 & -0.0293 & -0.0306 & 0.0191 & 0.1578 & -0.0681 & 0.2931 \\ -9.616 * 10^{-5} & 8.367 * 10^{-4} & -0.0103 & -0.3038 & -0.0681 & 0.9328 & -2.506 \\ 0.0062 & -0.0010 & 0.0526 & 0.0252 & 0.2931 & -2.506 & 231.237 \end{bmatrix} \quad (56)$$

3.3 Right Arm – Linearization Point 2

We also repeat this process for the right arm at a different linearization point given as: $Q_{r0} = [q_{r0}, \dot{q}_{r0}, \ddot{q}_{r0}]$, is defined as:

$$q_{r0} = [-0.2534, -0.0233, 0.2193, 1.389, -0.3570, 1.1938, 1.2655] \quad (57)$$

$$\dot{q}_{r0} = [-0.005497, 0.001178, 0.002356, 0.02945, -0.007853, 0.005105, 0.003141] \quad (58)$$

$$\ddot{q}_{r0} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00] \quad (59)$$

$$D_{r0} = \begin{bmatrix} 65.53 & 16.18 & 50.44 & 16.30 & 14.41 & 7.031 & 0.0430 \\ 16.18 & 16.46 & 16.15 & 8.694 & 6.787 & 3.560 & 0.0250 \\ 50.44 & 16.15 & 50.45 & 16.44 & 14.26 & 7.065 & 0.0439 \\ 16.30 & 8.694 & 16.44 & 10.15 & 5.386 & 3.958 & 0.0308 \\ 14.41 & 6.787 & 14.26 & 5.386 & 11.01 & 2.743 & 0.01158 \\ 7.031 & 3.560 & 7.065 & 3.958 & 2.743 & 2.701 & 0.02358 \\ 0.0430 & 0.0250 & 0.0439 & 0.0308 & 0.01158 & 0.02358 & 0.0046 \end{bmatrix} \quad (60)$$

$$V_{r0} = \begin{bmatrix} 1.006 & 1.0043 & 1.120 & 1.071 & -0.2436 & 0.4075 & -0.0017 \\ 0.3056 & 0.3032 & 0.3615 & 0.3523 & -0.0536 & 0.1921 & -0.0007 \\ 0.8833 & 0.9369 & 0.9951 & 1.007 & -0.2879 & 0.3622 & -0.0016 \\ 0.1874 & 0.1687 & 0.2301 & 0.2424 & -0.1673 & 0.1128 & -0.0006 \\ 0.3615 & 0.4741 & 0.4131 & 0.4847 & 0.04352 & 0.2493 & -0.0007 \\ 0.0716 & 0.08933 & 0.1095 & 0.1405 & -0.0616 & 0.0745 & 0.0004 \\ 0.0002 & -0.00018 & 0.00001 & -0.0003 & -0.001 & -0.00028 & -1.678 * 10^{-5} \end{bmatrix} \quad (61)$$

$$P_{r0} = \begin{bmatrix} 1.006 & 1.0043 & 1.120 & 1.071 & -0.2436 & 0.4075 & -0.0017 \\ 0.3056 & 0.3032 & 0.3615 & 0.3523 & -0.0536 & 0.1921 & -0.0007 \\ 0.8833 & 0.9369 & 0.9951 & 1.007 & -0.2879 & 0.3622 & -0.0016 \\ 0.1874 & 0.1687 & 0.2301 & 0.2424 & -0.1673 & 0.1128 & -0.0006 \\ 0.3615 & 0.4741 & 0.4131 & 0.4847 & 0.04352 & 0.2493 & -0.0007 \\ 0.0716 & 0.08933 & 0.1095 & 0.1405 & -0.0616 & 0.0745 & 0.0004 \\ 0.0002 & -0.00018 & 0.00001 & -0.0003 & -0.001 & -0.00028 & -1.678 * 10^{-5} \end{bmatrix} \quad (62)$$

Using the same procedure above we obtain the following A and B matrices for this linearization point:

$$A_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -0.0077 & -0.0038 & -0.0078 & -0.0037 & -0.0025 & -0.0027 & 1.0 * 10^{-5} & -0.0077 & -0.0038 & -0.0078 & -0.0037 & -0.0025 & -0.0027 & 1.053 * 10^{-5} \\ -0.0074 & -0.0052 & -0.0087 & -0.0046 & -0.0068 & -0.0052 & 1.703 * 10^{-5} & -0.0074 & -0.0052 & -0.0087 & -0.0046 & -0.0068 & -0.0052 & 1.703 * 10^{-5} \\ -0.0123 & -0.0162 & -0.0131 & -0.0146 & 0.0080 & 5.231 * 10^{-4} & 5.57 * 10^{-6} & -0.0123 & -0.0162 & -0.0131 & -0.0146 & 0.0080 & 5.213 * 10^{-4} & 5.573 * 10^{-6} \\ 0.0206 & 0.0337 & 0.0263 & 0.0299 & 0.0213 & 0.0124 & -4.73 * 10^{-5} & 0.0206 & 0.0337 & 0.0263 & 0.0299 & 0.0213 & 0.0124 & -4.731 * 10^{-5} \\ -0.0183 & -0.0328 & -0.0193 & -0.0286 & -0.0183 & -0.0190 & 2.045 * 10^{-5} & -0.0183 & -0.0328 & -0.0193 & -0.0286 & -0.0183 & -0.0190 & 2.045 * 10^{-5} \\ 0.0243 & 0.0093 & 0.0060 & -0.0151 & 0.0041 & -0.0154 & 1.266 * 10^{-4} & 0.0243 & 0.0093 & 0.0060 & -0.0151 & 0.0041 & -0.0154 & 1.266 * 10^{-4} \\ -0.0386 & 0.0677 & 0.0863 & 0.2239 & 0.0890 & 0.1555 & 0.0031 & -0.0386 & 0.0677 & 0.0863 & 0.2239 & 0.0890 & 0.1555 & 0.0031 \end{bmatrix} \quad (63)$$

$$B_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0662 & -6.1714 * 10^{-4} & -0.0664 & 0.0025 & -0.0016 & -9.616 * 10^{-5} & 0.0062 \\ -6.1714 * 10^{-4} & 0.1188 & -0.0029 & -0.0808 & -0.0293 & 8.367 * 10^{-4} & -0.0010 \\ -0.0664 & -0.0029 & 0.1166 & -0.0596 & -0.0306 & -0.0103 & 0.0526 \\ 0.0025 & -0.0808 & -0.0596 & 0.3685 & 0.0191 & -0.3038 & 0.0252 \\ -0.0016 & -0.0293 & -0.0306 & 0.0191 & 0.1578 & -0.0681 & 0.2931 \\ -9.616 * 10^{-5} & 8.367 * 10^{-4} & -0.0103 & -0.3038 & -0.0681 & 0.9328 & -2.506 \\ 0.0062 & -0.0010 & 0.0526 & 0.0252 & 0.2931 & -2.506 & 231.237 \end{bmatrix} \quad (64)$$

4. Model C Stackelberg Model

A Stackelberg model that augments the right and left arm was created for usage in Stackelberg simulations, the model is given below.

4.1 Stackelberg Model C

We consider the two-player linear stochastic system found in [2] as:

$$dx(t) = Ax(t) + B_1u_1(t) + B_2u_1(t) + Fdw(t), \quad (65)$$

$$x(t_0) = x_0, \quad (66)$$

$$y = Cx(t). \quad (67)$$

We define A as an augmented matrix of A_r and A_l defined as:

$$A = \begin{bmatrix} A_r & 0_{14\times14} \\ 0_{14\times14} & A_l \end{bmatrix}. \tag{68}$$

Furthermore, we define B_1 and B_2 as augmented matrices of A_r and A_l respectively as:

$$B_1 = \begin{bmatrix} B_r \\ 0_{14\times7} \end{bmatrix} \tag{69}$$

$$B_2 = \begin{bmatrix} 0_{14\times7} \\ B_l \end{bmatrix} \tag{70}$$

We also assume that the leader and follow control, u_1 and u_2, are the joint torques associated with the right and left arm respectively as:

$$u_1 = \delta\tau_r \tag{71}$$

$$u_2 = \delta\tau_l \tag{72}$$

Also, F is defined as a one's matrix with dimensions 28 x 1 that is given by:

$$F = \begin{bmatrix} F_r \\ F_l \end{bmatrix} = \begin{bmatrix} 1_{14x1} \\ 1_{14x1} \end{bmatrix} \tag{73}$$

Thus, substituting (60) to (62) into (57) to (59) we obtain

$$\begin{bmatrix} \delta\dot{q}_r \\ \delta\ddot{q}_r \\ \delta\dot{q}_l \\ \delta\ddot{q}_l \end{bmatrix} = \begin{bmatrix} A_r & 0_{14\times14} \\ 0_{14\times14} & A_l \end{bmatrix} \begin{bmatrix} \delta q_r \\ \delta\dot{q}_r \\ \delta q_l \\ \delta\dot{q}_l \end{bmatrix} + \begin{bmatrix} B_r \\ 0_{14\times7} \end{bmatrix} \delta\tau_r + \begin{bmatrix} 0_{14\times7} \\ B_l \end{bmatrix} \delta\tau_l + \begin{bmatrix} F_r \\ F_l \end{bmatrix} dw(t)$$

$$x(t_0) = x_0, \tag{74}$$

$$y = C \begin{bmatrix} \delta q_r \\ \delta\dot{q}_r \\ \delta q_l \\ \delta\dot{q}_l \end{bmatrix} \tag{75}$$

References:

[1] S. J. Lash, "Model Predictive Minimal Cost Variance Control and Dynamic Interrogation For Robotic Manipulation," dissertation, 2023

[2] F. Saleheen et al., "Bimodal Dynamic Imaging System for Tumor Characterization Using Hybrid Hierarchical Statistical Control," dissertation

[3] P. Corke, Robotics, vision and control fundamental algorithms in MATLAB. Springer, 2011.

[4] J. Terry, L. Rupert, and M. Killpack, "Comparison of linearized dynamic robot manipulator models for model predictive control," in IEEE-RAS International Conference on Humanoid Robots, pp. 205–212, 2017.

[5] M. W. Spong and M. Vidyasagar, Robot Modeling and Control 2nd Edition.

[6] C. Balafoutis and R. Patel, Dynamic analysis of robot manipulators: a Cartesian tensor approach. Springer Science Media, 1991

[7] I. R.L. Williams, "Baxter humanoid robot kinematics." Internet Publication, April 2017.

[8] RethinkRobotics, "Baxter_common/baxter_description/URDF/baxter.urdf at master · rethinkrobotics/baxter_common," GitHub, https://github.com/RethinkRobotics/baxter_common/blob/master/baxter_description/urdf/baxter.urdf.

[9] Robotics system toolbox, https://www.mathworks.com/help/robotics/index.html

[10] P. Corke, https://www.petercorke.com/RTB/r9/html/Link.html.

Appendix A Detailed Derivations of Joint Space Inertia and Corolis Matrix, D0, P0, and V0

A.1 Baxter Physical Parameters

To calculate the Inertia and Corolis matrix, $M\big(q(t)\big)$ and $C\big(q(t), \dot{q}(t)\big)$ respectively now denoted as *M* and *C* for writing simplicity, we need the physical parameters of the Baxter. Specifically, we need the following:

1. **The numerical values of the DH Parameters**: $\boldsymbol{q_i(t), a_i, d_i, \alpha_i}$ which are the joint angle, link length, link offset, and link twist for link *i* respectively. The joint angles, $q_i(t)$, for each link are given as a symbolic variable and thus do not have a numerical value until the final step when we substitute in the desired linearization point into the calculated D_0, V_0, P_0. DH parameters are given in Table A.1. This is needed to compute the homogenous transformation matrix $A_i(q_i(t))$: Eqn (5). This is also given in [2] and [8]. This $A_i(q_i(t))$ is used to calculate $T_n^0\big(q(t)\big)$ in Eqs (3) and (4). The numerical values of the link length, link offset, and link twist are given in Table A.1.

Table A.1. DH Parameters [2, 8]

Link i	Joint angle q	Link offset d (m)	Link length a (m)	Link Twist α (rad)
1	q_1	0.2703	0.069	$-\frac{\pi}{2}$
2	q_2	0	0	$+\frac{\pi}{2}$
3	q_3	0.3644	0.069	$-\frac{\pi}{2}$
4	q_4	0	0	$+\frac{\pi}{2}$
5	q_5	0.3743	0.01	$-\frac{\pi}{2}$
6	q_6	0	0	$+\frac{\pi}{2}$
7	q_7	0.2295	0	0

2. **The Inertia Matrix for each link w.r.t the link's Center of Mass I_i.**

3. **The Mass of each link: m_i .**

4. **The Position of the center of mass of each link w.r.t to the base frame: $o_{ci}^0(q(t))$.** The position of the center of mass w.r.t to the base frame can be found by:

$$o_{ci}^0(q(t)) = T_n^0(q(t)) * r \,. \tag{50}$$

Where r is the position of the center of mass of the link w.r.t to link *i*. r is provided by the Baxter.

To obtain the physical parameters above, we will be using the Robotic Systems Toolbox in MATLAB [9] which creates the model of the Baxter robot for simulations. It also contains the physical parameters of the robot.

Procedure to Obtain Baxter Physical Parameters ($a_i, d_i, \alpha_i, I_i, m_i, r$):

1. Install the Robotic Systems Toolbox [9] using the MATLAB add-on installer
2. Run the MATLAB code *ModelCDerivationBannan_v3.m* in Appendix B.1, which is the code that corresponds to this procedure A.1
3. You should find two variables in the workspace named 'left' and 'right', these correspond to the right and left arm *SerialLink* objects of each arm respectively. Double click on one of the right or left arm in the workspace. Then you should get the following image below in Fig A.1.

1x1 SerialLink

Property	Value
name	'Baxter LEFT'
gravity	[0;0;9.8100]
base	*1x1 SE3*
tool	*1x1 SE3*
manufacturer	'Rethink Robotics'
comment	''
plotopt	*[ ]*
fast	0
interface	*[ ]*
ikineType	*[ ]*
trail	*[ ]*
movie	*[ ]*
delay	*[ ]*
loop	*[ ]*
model3d	*[ ]*
faces	*[ ]*
points	*[ ]*
plotopt3d	*[ ]*
n	7
links	*1x7 Link*
T	*[ ]*
mdh	0
config	'RRRRRRR'
offset	[0,1.5708,0,0,0,0,0]
qlim	*7x2 double*
d	[0.2700,0,0.3640,...
a	[0.0690,0,0.0690,...
alpha	[-1.5708,1.5708,-...
theta	*[ ]*

Fig A.1. Serial Link Parameters

4. Click on the "links" variable and it should show you the seven links of the Baxter. This can be shown below in Fig A.2. Each one of these links contains the physical parameters for that link.

left.links

	1	2	3	4	5	6	7
1	*1x1 Link*	*1x1 Link*	*1x1 Link*	*1x1 Link*	*1x1 Link*	*1x1 Link*	*1x1 Link*
2							

Fig A.2. Left Arm Links

5. Click on the first link (i = 1) and it will show you the parameters for the first link in the left arm as in Fig A.3.

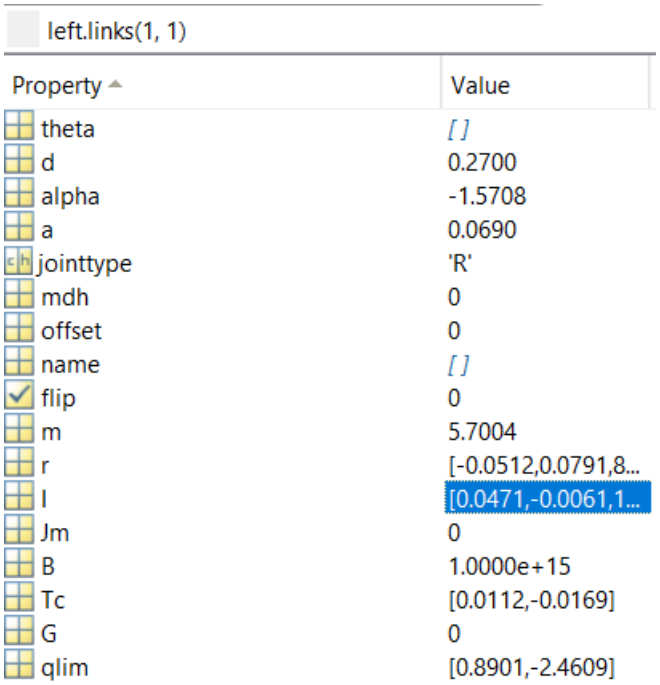

Fig A.3. Link Arm Parameters for link $i = 1$

The parameters found in Fig A.3. are the physical parameters of the left arm for the first link. Here are the terms listed above and their corresponding parameters, if you need a more detailed explanation please visit Peter Corke's website here: Link Parameters [10].

1. **d**: link offset
2. **alpha**: link twist
3. **a**: link length
4. **m**: mass of link
5. **r**: position of COM of link w.r.t to link frame
6. **I**: 3x3 Link Inertia matrix w.r.t to COM

Thus, from these six values, we obtain the physical parameters of Baxter for link 1.

6. Repeat Step 5 for each of the links in the arm: $i = 2,..,7$
7. Manually copy over the values given in steps 5-6 and paste them into directly into *Model C Caluclations.nb* as done in this document.

A.2 Joint Space Inertia, Corolis, and D_0 Calculations

Now that we have the physical Baxter parameters from section A.1, we need to calculate the M, C and D_0, this is done in the code found in Appendix B.2. B.2 is Wolfram Mathematica code and thus requires the user to explicitly tell Mathematica what the computer is looking at. We had selected to use Mathematica over MATLAB for the calculations of the matrices because Mathematica's symbolic computations are faster than MATLAB. It is not like MATLAB where the computations are done as if they were an array. Mathematica instead tries its best to interpret what you have given it. Thus, there was some debugging that was required to get the code to run by defining the parameters as array with proper dimensions. There is a function *dhTransformation* which computes the homogenous transformation matrices found in (5) that is called for each link. B.2 saves the *M*, *C*, and D_0 matrices as "M.mx", "C0.mx", "D0.mx" respectively as a binary file using *dumpsave* for faster computational use. Thus, because they are saved as a binary file, you cannot directly view these matrices in a human readable

format. It is still possible to do that, however the computation time and size of the resulting matrices may take a long time to view and save. Note the save location will likely have to be changed for your desired save location when you run the code. If the code runs and you do not see the save file, it is likely in the directory that Mathematica saves its files by default.

Procedure for Joint Space Inertia, Corolis, and D_0 Calculations:

1. Before running B.2, make sure to run B.1 and copy the relevant physical parameters either manually or by loading the data into Mathematica. Code B.2 already has the parameters manually defined at the beginning of the script, thus you can run A.2.1 by itself.
2. Change the save output directory of the *M, C,* and D_0 to the directory you want to save the output files to. This requires changing the DumpSave lines. For example, for the M matrix dumpsave line:

   DumpSave["~/Desktop/Research Folder/Baxter Research/M.mx", {M}]

   to

   DumpSave["*YourSaveLocation*/M.mx", {M}]

3. Run *Model C Calculations.nb* (Code B.2) in Mathematica. This will take 5 minutes to run or longer.
4. The output of the file should save the symbolic *M, C,* and D_0 matrices as "M.mx", "C0.mx", "D0.mx" respectively as a binary file in the directory that you are running the code. Thus, it returns the calculations for (6), (13), and (27).

A.3 V_0, P_0 Calculations

Originally, we had tried to compute *M, C,* and D_0, V_0, P_0 in one script, however this ended up causing long computational time and causing the kernel doing the calculations to crash. Thus, we split the calculations for V_0 and P_0 into their own files. D_0 was already found in B.2 because it is just the mass matrix, *M*, evaluated at the linearization point. The file that takes the longest to compute is P_0 and was giving us the same issues as before, so it was separated into calculations for its two terms to ease computation. The file that calculates V_0 is given in B.3, the first term and second terms of P_0 in B.4 and B.5 respectively. Note you will have to change the file location of the GET commands in Appendix B.3 to B.5 as well when doing the calculations when retrieving the previously calculated M and C matrices.

Code Procedure for V_0 and P_0 Calculations:

1. Change the save locations of the V0 dumpsave line to where you want the output to be saved
2. Run Mathematica Appendix B.3 Code, *V0_Calc.nb.* This will calculate V_0 (30) and should create V0.mx saved to the directory you specified in step 1. This script may take a couple hours to run.
3. Change the save locations of the P01 dumpsave line to where you want the output to be saved
4. Run Mathematica Appendix B.4 Code, *P0_Calc.nb.* This should calculate the first term of P0 (31) create P01.mx saved to the directory you specified in step 1. This script may take a couple of hours to run.
5. Change the save locations of the P02 dumpsave line to where you want the output to be saved

6. Run Mathematica Appendix B.5 Code, *P02_Calc.nb*. This should calculate the second term of P0 (31) create P02.mx saved to the directory you specified in step 1. This script may take up a significant amount of time to run, around 5 hours.

A.4 D_0, V_0, P_0 Numerical Calculations

Once all of the matrices for the model have been calculated symbolically, we can replace the symbolic values with the desired numerical values of our linearization point by running the code in A.4.1 Once that is done you should have numerical values for D_0, V_0, P_0 which then can be copied into MATLAB and following (34) can be used to find the state space model.

Code Procedure for V_0 and P_0 Calculations:

1. Change the save locations of the D0N, V0N, and P0N dumpsave line to where you want the output to be saved
2. Change the values of linearization point, Q_0, as desired
3. Run Mathematica A.4.1 Code, *NumericalModel.nb*. This should return D_0, V_0, P_0 evaluated at the given linearization point, Q_0, as matrices D0N.mx, V0N.mx, and P0N.mx.

Once D0N, V0N, and P0N are found, we can use (34) to find the A and B matrices of the model and thus have found the dynamics model of the Baxter arm.

Appendix B Code for Mass and Corolis Matrix, D0, P0, and V0

B.1 MATLAB Code to Find CSNAP Baxter Physical Parameters

```
%% Baxter Model Physical Parameters Software
% Author: Jack Bannan
% Email: john.bannan@temple.edu
% Temple CSNAP
% Last Modified: 2/26/2025
% Input: None
% Output: left and right Serial Link Object to workspace which contain the
% baxter parameters for the right and left arm
clear
close all
clc
format shortE

%% Calc Mass Matrix
% Load Baxter Robot
[baxter] = make_csnap_baxter(); % Creates Baxter Model
left = baxter.left % Isolate Left Arm
right = baxter.right % Isolate Right Arm

%% Functions
% Creates Baxter Model
function [baxter] = make_csnap_baxter()
%make_csnap_baxter Generates the Baxter model from csnap components
%   Generates the Baxter dual-arm model as a single struct using the
%       link objects from Peter Corke's Robotics toolbox. The Baxter model
```

```
%       packaged with that toolbox has incorrect base rotations so arm
%       configurations are incorrect. This model fixes the base rotations
%       and packages both arms together into a single struct object.


%     th   d   a   alpha

    viscous_friction = 1e15;  %Define arbitrary joint friction (viscous) for sim
stability.
    coloumb_friction = [11.2e-3, -16.9e-3];

    links = [
            Revolute(...
                'd', 0.27, ...
                'a', 0.069, ...
                'alpha', -pi/2, ...
                'r', [-0.051170 0.079080 0.000859], ...
                'm', 5.700440, ...
                'I', [0.047091 0.035959 0.037669 -0.006149 -0.000781 0.000128], ...
                'B', viscous_friction, ...
                'Tc', coloumb_friction, ...
                'qlim', deg2rad([51 -141])...
            )
            Revolute( ...
                'd', 0, ...
                'a', 0, ...
                'alpha', pi/2, ...
                'offset', pi/2, ...
                'r', [ 0.002690 -0.005290 0.068449], ...
                'm', 3.226980, ...
                'I', [0.027885 0.020787 0.011752 -0.000188 0.002077 -0.000300], ...
                'B', viscous_friction, ...
                'Tc', coloumb_friction, ...
                'qlim', deg2rad([60 -123]) ...
            )
            Revolute( ...
                'd', 0.102+0.262, ...
                'a', 0.069, ...
                'alpha', -pi/2, ...
                'r', [ -0.071760 0.081490 0.001319], ...
                'm', 4.312720, ...
                'I', [0.026617 0.012480 0.028443 -0.003921 -0.001084 0.000293], ...
                'B', viscous_friction, ...
                'Tc', coloumb_friction, ...
                'qlim', deg2rad([173 -173]) ...
            )
            Revolute( ...
                'd', 0, ...
                'a', 0, ...
                'alpha', pi/2, ...
                'r', [ 0.001590 -0.011170 0.026179], ...
                'm', 2.072060, ...
                'I', [0.013182 0.009268 0.007116 -0.000197 0.000745 0.000360], ...
                'B', viscous_friction, ...
                'Tc', coloumb_friction, ...
```

```
            'qlim', deg2rad([150 -3]) ...
        )
        Revolute( ...
            'd', 0.103+0.271, ...
            'a', 0.010, ...
            'alpha', -pi/2, ...
            'r', [ -0.011679 0.13111 0.004599], ...
            'm', 2.246650, ...
            'I', [0.016677 0.003746 0.016754 -0.000186 0.000647 0.000184], ...
            'B', viscous_friction, ...
            'Tc', coloumb_friction, ...
            'qlim', deg2rad([175 -175]) ...
        )
        Revolute( ...
            'd', 0, ...
            'a', 0, ...
            'alpha', pi/2, ...
            'r', [ 0.006970 0.005999 0.060480], ...
            'm', 1.609790, ...
            'I', [0.007005 0.005527 0.003876 0.000153 -0.000211 -0.000444], ...
            'B', viscous_friction, ...
            'Tc', coloumb_friction, ...
            'qlim', deg2rad([120 -90]) ...
        )
        Revolute( ...
            'd', 0.28, ...
            'a', 0, ...
            'alpha', 0, ...
            'r', [ 0.005137 0.000957 -0.066823], ...
            'm', 0.542180, ...
            'I', [0.000816 0.000873 0.003876 0.000128 0.000106 0.000189], ...
            'B', viscous_friction, ...
            'Tc', coloumb_friction, ...
            'qlim', deg2rad([175 -175]) ...
        )
    ];

    baxter.left =  SerialLink(links, 'name', 'Baxter LEFT', 'manufacturer', 'Rethink
Robotics');
    baxter.right = SerialLink(links, 'name', 'Baxter RIGHT', 'manufacturer', 'Rethink
Robotics');

    baxter.left.base = transl(0.064614, 0.25858, 0.119)*rpy2tr(pi/4, 0, 0, 'xyz');
    baxter.right.base = transl(0.063534, -0.25966, 0.119)*rpy2tr(-pi/4, 0, 0, 'xyz');
    baxter.note="CSNAP Baxter Model (corrected base rotations)";
end
```

B.2 $M(q(t))$ and $C(q(t), \dot{q}(t))$ Mathematica Calculation Code (Model B Calculations.nb)

(* Baxter Symbolic Model Script

Author: Jack Bannan

Temple CSNAP

```
Last Modified: 2/26/2025*)
(*Input: None*)
(*Output: Mass Matrix, M, Inertia Matrix I, and D0 as binary .mx \
files*)

(*Starts Timer for Script Run Time Calculation*)
startTime = SessionTime[];

(* Import Baxter Parameters From MATLAB Physical Parameters Script*)
(* DH Parameters*)
(* Link Length of each link*)
a = {0.0690000000000000, 0, 0.0690000000000000, 0, 0.0100000000000000,
   0, 0};

(* Link offset of each link*)
d = {0.270000000000000, 0, 0.364000000000000, 0, 0.374000000000000, 0,
   0.280000000000000};

(* Link Twist of each link*)
alpha = {-1.57079632679490, 1.57079632679490, -1.57079632679490,
  1.57079632679490, -1.57079632679490, 1.57079632679490, 0};

(* Other Robot Parameters*)
(* Mass of Each Link*)
m = {5.70044000000000, 3.22698000000000, 4.31272000000000,
  2.07206000000000, 2.24665000000000, 1.60979000000000,
  0.542180000000000};

(* Inertia Matrix for Each Link @ Center of Mass*)
(*Ii refers to the inerita matrix I, for link i. e.g. I1 is the \
inetia matrix for link 1*)
(*Link 1 Inertia Matrix*)
I1 = {{0.0470910000000000, -0.00614900000000000,
   0.000128000000000000}, {-0.00614900000000000,
   0.0359590000000000, -0.000781000000000000}, \
```

```
{0.000128000000000000, -0.000781000000000000, 0.0376690000000000}};

(*Link 2 Inertia Matrix*)
I2 = {{0.0278850000000000, -0.000188000000000000, \
-0.000300000000000000},
   {-0.000188000000000000, 0.0207870000000000, 0.00207700000000000},
   {-0.000300000000000000, 0.00207700000000000, 0.0117520000000000}};

(*Link 3 Inertia Matrix*)
I3 = {{0.0266170000000000, -0.00392100000000000,
   0.000293000000000000}, {-0.00392100000000000,
   0.0124800000000000, -0.00108400000000000},
   {0.000293000000000000, -0.00108400000000000, 0.0284430000000000}};

(*Link 4 Inertia Matrix*)
I4 = {{0.0131820000000000, -0.000197000000000000,
   0.000360000000000000},
   {-0.000197000000000000, 0.00926800000000000,
   0.000745000000000000},
   {0.000360000000000000, 0.000745000000000000, 0.00711600000000000}};

(*Link 5 Inertia Matrix*)
I5 = {{0.0166770000000000, -0.000186000000000000,
   0.000184000000000000},
   {-0.000186000000000000, 0.00374600000000000,
   0.000647000000000000},
   {0.000184000000000000, 0.000647000000000000, 0.0167540000000000}};

(*Link 6 Inertia Matrix*)
I6 = {{0.00700500000000000,
   0.000153000000000000, -0.000444000000000000},
   {0.000153000000000000,
   0.00552700000000000, -0.000211000000000000},
   {-0.000444000000000000, -0.000211000000000000,
   0.00387600000000000}};
```

```
(*Link 7 Inertia Matrix*)
I7 = {{0.000816000000000000, 0.000128000000000000,
   0.000189000000000000},
  {0.000128000000000000, 0.000873000000000000,
   0.000106000000000000},
  {0.000189000000000000, 0.000106000000000000,
   0.00387600000000000}};
Dimensions[I7];(* This line just shows the dimensions of the matrix*)

(* Concatenates Link Inertia Matrices into one array for easier \
computation*)
IMat = {I1, I2, I3, I4, I5, I6, I7};


(* Center of Mass Location w.r.t Each Link's Frame*)
(*We copied the location of the COM for each link w.r.t to the Link's \
Frame
However, we need to find it w.r.t to the base frame. Thus in order to \
do that we
need to multiply the Transformation matrix of the link frame to the
base frame by the COM w.r.t to the link frame. Thus, since the \
transformation matrix is a 4x4 matrix,
 we must add a 1 to the end of COM vectors to give us a 4x1 matrix.*)
(* Element 1 Added to end for Matrix Multiplication*)
COM1 = {{-0.0511700000000000, 0.0790800000000000,
   0.000859000000000000, 1}};
COM2 = {{0.00269000000000000, -0.00529000000000000,
   0.0684490000000000, 1}};
COM3 = {{-0.0717600000000000, 0.0814900000000000, 0.00131900000000000,
    1}};
COM4 = {{0.00159000000000000, -0.0111700000000000, 0.0261790000000000,
    1}};
COM5 = {{-0.0116790000000000, 0.131110000000000, 0.00459900000000000,
   1}};
```

```
COM6 = {{0.00697000000000000, 0.00599900000000000, 0.0604800000000000,
    1}};
COM7 = {{0.00513700000000000,
   0.000957000000000000, -0.0668230000000000, 1}};
Dimensions[COM1];

(* Symbolic Variables*)
(* Joint Angles*)
q = {q1, q2, q3, q4, q5, q6, q7};
qd = {qd1, qd2, qd3, qd4, qd5, qd6, qd7};
qdd = {qdd1, qdd2, qdd3, qdd4, qdd5, qdd6, qdd7};

(* Calculate Symbolically Transformation Matrices*)
(* This find the homogenous transformation matrices of each link in \
Eqn (5)*)
A1 = Evaluate[dhTransformation[q[[1]], a[[1]], alpha[[1]], d[[1]]]];
A2 = Evaluate[dhTransformation[q[[2]], a[[2]], alpha[[2]], d[[2]]]];
A3 = Evaluate[dhTransformation[q[[3]], a[[3]], alpha[[3]], d[[3]]]];
A4 = Evaluate[dhTransformation[q[[4]], a[[4]], alpha[[4]], d[[4]]]];
A5 = Evaluate[dhTransformation[q[[5]], a[[5]], alpha[[5]], d[[5]]]];
A6 = Evaluate[dhTransformation[q[[6]], a[[6]], alpha[[6]], d[[6]]]];
A7 = Evaluate[dhTransformation[q[[7]], a[[7]], alpha[[7]], d[[7]]]];
Dimensions[A1];

(* Homogenous Transformation Matrices*)
(* This calculates the homogenous transformation matrices for each \
link to the base frame
as in Eqn (4)*)
T00 = IdentityMatrix[4];
T01 = A1;
T02 = A1 . A2;
T03 = A1 . A2 . A3;
T04 = A1 . A2 . A3 . A4;
T05 = A1 . A2 . A3 . A4 . A5;
T06 = A1 . A2 . A3 . A4 . A5 . A6;
```

```
T07 = A1 . A2 . A3 . A4 . A5 . A6 . A7;
Dimensions[T07];

(* Position Vectors*)
(* These are the position vectors found in Eqn (3)*)
O00 = T00[[1 ;; 3, {4}]];
O01 = T01[[1 ;; 3, {4}]];
O02 = T02[[1 ;; 3, {4}]];
O03 = T03[[1 ;; 3, {4}]];
O04 = T04[[1 ;; 3, {4}]];
O05 = T05[[1 ;; 3, {4}]];
O06 = T06[[1 ;; 3, {4}]];
O07 = T07[[1 ;; 3, {4}]];
Dimensions[O07];

(* Rotation Matrices*)
(* These are the rotation matrices found in Eqn (3)*)
R00 = IdentityMatrix[3];
R01 = T01[[1 ;; 3, 1 ;; 3]];
R02 = T02[[1 ;; 3, 1 ;; 3]];
R03 = T03[[1 ;; 3, 1 ;; 3]];
R04 = T04[[1 ;; 3, 1 ;; 3]];
R05 = T05[[1 ;; 3, 1 ;; 3]];
R06 = T06[[1 ;; 3, 1 ;; 3]];
R07 = T07[[1 ;; 3, 1 ;; 3]];
Dimensions[R00];

(*Append rotation matrices together for easier computation*)
R = {R01, R02, R03, R04, R05, R06, R07};

(*Center of Mass of Each Link in Terms of Base Frame*)
(* Transpose COM*)
COM1T = Transpose[COM1];
COM2T = Transpose[COM2];
COM3T = Transpose[COM3];
```

```
COM4T = Transpose[COM4];
COM5T = Transpose[COM5];
COM6T = Transpose[COM6];
COM7T = Transpose[COM7];
Dimensions[COM1T];

(*Multiplies Homogenous Transformation Matrices by COM to get
 the COM position in terms of base frame, \!\(
\*SubsuperscriptBox[\(o\), \(c i\), \(0\)](\(q(t)\))\) in Eqn (12)*)
C1 = T01 . COM1T;
C2 = T02 . COM2T;
C3 = T03 . COM3T;
C4 = T04 . COM4T;
C5 = T05 . COM5T;
C6 = T06 . COM6T;
C7 = T07 . COM7T;
Dimensions[C7];

(*Determine Jacobians for Each Link*)
(* Angular Jacobians, Eqns (7-10)*)
(* Define K*)
K = {{0}, {0}, {1}};
Dimensions[K];

(*Link 1*)
(*Eqn 9*)
Jw1 = ArrayReshape[
   Transpose[{R00 . K, ConstantArray[0, {3, 1}],
     ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}],
     ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}],
     ConstantArray[0, {3, 1}]}], {3, 7}];
Dimensions[Jw1];



(*Link 2*)
```

```
(*Eqn 9*)
Jw2 = ArrayReshape[
  Transpose[{R00 . K, R01 . K, ConstantArray[0, {3, 1}],
    ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}],
    ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}]}], {3, 7}];

(*Link 3*)
(*Eqn 9*)
Jw3 = ArrayReshape[
  Transpose[{R00 . K, R01 . K, R02 . K, ConstantArray[0, {3, 1}],
    ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}],
    ConstantArray[0, {3, 1}]}], {3, 7}];

(*Link 4*)
(*Eqn 9*)
Jw4 = ArrayReshape[
  Transpose[{R00 . K, R01 . K, R02 . K, R03 . K,
    ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}],
    ConstantArray[0, {3, 1}]}], {3, 7}];

(*Link 5*)
(*Eqn 9*)
Jw5 = ArrayReshape[
  Transpose[{R00 . K, R01 . K, R02 . K, R03 . K, R04 . K,
    ConstantArray[0, {3, 1}], ConstantArray[0, {3, 1}]}], {3, 7}];

(*Link 6*)
(*Eqn 9*)
Jw6 = ArrayReshape[
  Transpose[{R00 . K, R01 . K, R02 . K, R03 . K, R04 . K, R05 . K,
    ConstantArray[0, {3, 1}]}], {3, 7}];

(*Link 7*)
(*Eqn 9*)
Jw7 = ArrayReshape[
```

```
  Transpose[{R00 . K, R01 . K, R02 . K, R03 . K, R04 . K, R05 . K,
    R07 . K}], {3, 7}];

(*Eqn 7*)
Jw = {Jw1, Jw2, Jw3, Jw4, Jw5, Jw6, Jw7};
(*Checks Dimensions of jacobians*)
Dimensions[Jw1];
Dimensions[R00 . K];
Dimensions[ConstantArray[0, {3, 1}]];
Jw[[1]];
Dimensions[Jw[[1]]];

(* Linear Jacobians, Eqns(11, 12)*)
(*Link 1*)
(*Eqn 12*)
(*Reshapes Array to Proper Dimensions*)
Jv11 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C1[[1 ;; 3]] - O00]]}];
Jv12 = ConstantArray[0, {3, 1}];
Jv13 = ConstantArray[0, {3, 1}];
Jv14 = ConstantArray[0, {3, 1}];
Jv15 = ConstantArray[0, {3, 1}];
Jv16 = ConstantArray[0, {3, 1}];
Jv17 = ConstantArray[0, {3, 1}];
Jv1 = ArrayReshape[
  Transpose[{Jv11, Jv12, Jv13, Jv14, Jv15, Jv16, Jv17}], {3, 7}];

Dimensions[O00];
Dimensions[C1[[1 ;; 3]]];
Dimensions[Flatten[R00 . K]];
Dimensions[Jv11];
Dimensions[Jv12];
Dimensions[Jv1];

(*Link 2*)
```

```
(*Eqn 12*)
Jv21 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C2[[1 ;; 3]] - O00]]}];
Jv22 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C2[[1 ;; 3]] - O01]]}];
Jv23 = ConstantArray[0, {3, 1}];
Jv24 = ConstantArray[0, {3, 1}];
Jv25 = ConstantArray[0, {3, 1}];
Jv26 = ConstantArray[0, {3, 1}];
Jv27 = ConstantArray[0, {3, 1}];
Jv2 = ArrayReshape[
  Transpose[{Jv21, Jv22, Jv23, Jv24, Jv25, Jv26, Jv27}], {3, 7}];

(*Link 3*)
(*Eqn 12*)
Jv31 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C3[[1 ;; 3]] - O00]]}];
Jv32 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C3[[1 ;; 3]] - O01]]}];
Jv33 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C3[[1 ;; 3]] - O02]]}];
Jv34 = ConstantArray[0, {3, 1}];
Jv35 = ConstantArray[0, {3, 1}];
Jv36 = ConstantArray[0, {3, 1}];
Jv37 = ConstantArray[0, {3, 1}];
Jv3 = ArrayReshape[
  Transpose[{Jv31, Jv32, Jv33, Jv34, Jv35, Jv36, Jv37}], {3, 7}];

(*Link 4*)
(*Eqn 12*)
Jv41 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C4[[1 ;; 3]] - O00]]}];
Jv42 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C4[[1 ;; 3]] - O01]]}];
Jv43 = Transpose[{Cross[Flatten[R00 . K],
```

```
    Flatten[C4[[1 ;; 3]] - O02]]}];
Jv44 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C4[[1 ;; 3]] - O03]]}];
Jv45 = ConstantArray[0, {3, 1}];
Jv46 = ConstantArray[0, {3, 1}];
Jv47 = ConstantArray[0, {3, 1}];
Jv4 = ArrayReshape[
  Transpose[{Jv41, Jv42, Jv43, Jv44, Jv45, Jv46, Jv47}], {3, 7}];

(*Link 5*)
(*Eqn 12*)
Jv51 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C5[[1 ;; 3]] - O00]]}];
Jv52 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C5[[1 ;; 3]] - O01]]}];
Jv53 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C5[[1 ;; 3]] - O02]]}];
Jv54 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C5[[1 ;; 3]] - O03]]}];
Jv55 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C5[[1 ;; 3]] - O04]]}];
Jv56 = ConstantArray[0, {3, 1}];
Jv57 = ConstantArray[0, {3, 1}];
Jv5 = ArrayReshape[
  Transpose[{Jv51, Jv52, Jv53, Jv54, Jv55, Jv56, Jv57}], {3, 7}];

(*Link 6*)
(*Eqn 12*)
Jv61 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C6[[1 ;; 3]] - O00]]}];
Jv62 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C6[[1 ;; 3]] - O01]]}];
Jv63 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C6[[1 ;; 3]] - O02]]}];
Jv64 = Transpose[{Cross[Flatten[R00 . K],
```

```
    Flatten[C6[[1 ;; 3]] - O03]]}];
Jv65 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C6[[1 ;; 3]] - O04]]}];
Jv66 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C6[[1 ;; 3]] - O05]]}];
Jv67 = ConstantArray[0, {3, 1}];
Jv6 = ArrayReshape[
  Transpose[{Jv61, Jv62, Jv63, Jv64, Jv65, Jv66, Jv67}], {3, 7}];

(*Link 7*)
(*Eqn 12*)
Jv71 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O00]]}];
Jv72 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O01]]}];
Jv73 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O02]]}];
Jv74 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O03]]}];
Jv75 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O04]]}];
Jv76 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O05]]}];
Jv77 = Transpose[{Cross[Flatten[R00 . K],
    Flatten[C7[[1 ;; 3]] - O06]]}];
Jv7 = ArrayReshape[
  Transpose[{Jv71, Jv72, Jv73, Jv74, Jv75, Jv76, Jv77}], {3, 7}];

Jv = {Jv1, Jv2, Jv3, Jv4, Jv5, Jv6, Jv7};

(*Compute Mass matrix, Eqn(6)*)
(*Initialize Mass Matrix*)
n = 7; M = ConstantArray[0, {n, n}]; (*Assuming a 7x7 matrix*)
(*Loop from 1 to n*)
M = Total[
```

```
  Table[m[[i]]*Transpose[Jv[[i]]] . Jv[[i]] +
    Transpose[Jw[[i]]] . (R[[i]] . IMat[[i]] . Transpose[R[[i]]]) .
     Jw[[i]], {i, 1, n}]]
Dimensions[M]

(*Compute Inertia matrix, Eqn(13, 14)*)
C0 = ConstantArray[0, {n, n}];

(*Compute Partial Derivatives and Christoffel Symbols, Eqn (14)*)
C0 = Table[
  Sum[(*Compute Partial Derivatives*)dMkjdqi = D[M[[k, j]], q[[i]]];
   dMkidqj = D[M[[k, i]], q[[j]]];
   dMijdqk = D[M[[i, j]], q[[k]]];
   (*Christoffel symbol c_ijk*)
   cijk = 1/2 *(dMkjdqi + dMkidqj - dMijdqk);
   (*Accumulate into C[k,j]*)
   cijk*qd[[i]], {i, 1, n}], {k, 1, n}, {j, 1, n}]
Dimensions[C0]

(* Sets D0 as M*)
D0 = M

(*Saves M, C0, and D0 to thier respective files
Note: Please change the save location to YOUR OWN save location*)
DumpSave["~/Desktop/Research Folder/Baxter Research/M.mx", {M}]
DumpSave["~/Desktop/Research Folder/Baxter Research/C0.mx", {C0}]
DumpSave[
 "~/Desktop/Research Folder/Baxter Research/D0.mx", {D0}]

(*Calculates Run Time of Script*)
endTime = SessionTime[];
executionTime = endTime - startTime

(*Functions*)
(*Defines the homogeneous transformation matrix A, Eqn (5)*)
```

```
dhTransformation[theta_, d_, a_,
  alpha_] := {{Cos[theta], -Sin[theta]*Cos[alpha],
    Sin[theta]*Sin[alpha], a* Cos[theta]}, {Sin[theta],
    Cos[theta]*Cos[alpha], -Cos[theta]*Sin[alpha], a*Sin[theta]}, {0,
    Sin[alpha], Cos[alpha], d}, {0, 0, 0, 1}}
```

B.3 V0 Symbolic Calculation Mathematica Code (V0_Calc.nb)

```
(* V0 Calculations*)
(* Jack Bannan, Temple CSNAP Lab*)
(* This script Calculates V0 from the Mass and Coriolis Matrices*)
(* Input: M.mx and C0.mx Matrices*)
(*Output: V0.mx matrix*)
(* Start Script Timer*)
startTime = SessionTime[];

(* Symbolic Variables*)
(* Joint Angles*)
q = {q1, q2, q3, q4, q5, q6, q7};
qd = {qd1, qd2, qd3, qd4, qd5, qd6, qd7};
qdd = {qdd1, qdd2, qdd3, qdd4, qdd5, qdd6, qdd7};

(* Load M and C Matrices*)
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\M.mx"]
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\C0.mx"];

(*V0 Calculation, Eqn(30)*)
V0 = D[C0, {qd}] . qd + C0;
```

```
DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\V0.mx", {V0}];

(*Calculate Script Execution Time*)
endTime = SessionTime[];
executionTime = endTime - startTime
```

B.4 P0 First Term Symbolic Calculation Mathematica Code (P0_Calc.nb)

```
(* P0 Calculations*)
(* Jack Bannan, Temple CSNAP Lab*)
(* This script calculates the first term of P0*)
(* Input: M.mx and C0.mx Matrices*)
(*Output: P01.mx matrix*)
(* Start Script Timer*)
startTime = SessionTime[];

(* Load M and C Matrices*)
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\M.mx"]
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\C0.mx"];

(* Symbolic Variables*)
(* Joint Angles*)
q = {q1, q2, q3, q4, q5, q6, q7};
qd = {qd1, qd2, qd3, qd4, qd5, qd6, qd7};
qdd = {qdd1, qdd2, qdd3, qdd4, qdd5, qdd6, qdd7};
```

```
(*P0 Calculation, Eqn(31 first term only)*)
P01 = D[M, {q}] . qdd

DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\P01.mx", {P01}];

endTime = SessionTime[];
executionTime = endTime – startTime
```

B.5 P0 Second Term Symbolic Calculation Mathematica Code (P02_Calc.nb)

```
(* P0_2 Calculations*)
(* Jack Bannan, Temple CSNAP Lab*)
(* This script calculates the second term of P0*)
(* Input: M.mx and C0.mx Matrices*)
(*Output: P02.mx matrix*)
(* Start Script Timer*)
startTime = SessionTime[];

(* Load M and C Matrices*)
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\C0.mx"];

(* Symbolic Variables*)
(* Joint Angles*)
q = {q1, q2, q3, q4, q5, q6, q7};
qd = {qd1, qd2, qd3, qd4, qd5, qd6, qd7};
qdd = {qdd1, qdd2, qdd3, qdd4, qdd5, qdd6, qdd7};
```

```
(*P0 Calculation, Eqn(31 second term only)*)
P02 = D[C0, {q}] . qd

(*Save Matrix to Binary File*)
DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\P02.mx", {P02}];

endTime = SessionTime[];
executionTime = endTime – startTime
```

B.6 Numerical Calculation Mathematica Code (NumericalModel.nb)

```
(* Numerical Implementation*)
(* Jack Bannan
CSNAP LAB *)
(*Input: M.mx, C0.mx, D0.mx, V0.mx, P01.mx, P02.mx, Numerical values \
(Linerization point) for q, qd, qdd*)
(*Output: Numerical Values of D0, V0 and P0 saved as D0N.mx, V0N.mx, \
and P0N.mx respectively*)

(*Numerical Evaluation*)
(* Set q, qd, qdd symbolic values to numerical values*)
values = {q1 -> -1.1157, q2 -> -1.0789, q3 -> -0.1432, q4 -> 1.5472,
   q5 -> 0.0754, q6 -> 1.1080, q7 -> 0.4904, qd1 -> 0, qd2 -> -0,
   qd3 -> 0, qd4 -> 0, qd5 -> 0, qd6 -> 0, qd7 -> 0, qdd1 -> 0,
   qdd2 -> 0, qdd3 -> 0, qdd4 -> 0, qdd5 -> 0, qdd6 -> 0, qdd7 -> 0};

(* Get Matrices*)
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
```

```
Research\\M.mx"];
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\C0.mx"];
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\D0.mx"];
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\V0.mx"];
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\P01.mx"];
Get["C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\P02.mx"];

(* Calculates and Displays Numeric Matrices for D0, V0, and P0*)
D0N = N[D0 /. values]
V0N = N[V0 /. values]
P0N = N[P01 /. values] + N[P02 /. values]

(*Saves D0, V0, P0 Numerical Calculations*)
DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\D0N.mx", {D0N}];
DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\V0N.mx", {V0N}];
DumpSave[
  "C:\\Users\\tuq53094\\Desktop\\Research Folder\\Baxter \
Research\\P0N.mx", {P0N}];
```