SYDE 556 Project - Roll,Pitch and Yaw control in a Lamprey

Gogulashanth Sathiyaseelan (20554735)

April 2019

1 Outline

This project is an extension of the Lamprey locomotion model developed in the textbook. The extension involves:

1. High-level pitch and yaw acceleration inputs (θ, ϕ, respectively) and roll angular velocity (denoted as A). These parameters are in addition to the ω, angular velocity of forward motion input.
2. Extension of mechanical Lamprey model from a 2D segmented model to a 3D segmented model (see below).
3. Extension of the muscular tensions model to account for roll,pitch and yaw.
4. Extension of neuronal activity dynamics to account for roll,pitch and yaw.

The modeling approach is similar to the one described in the text (i.e. Top down approach).

2 System description

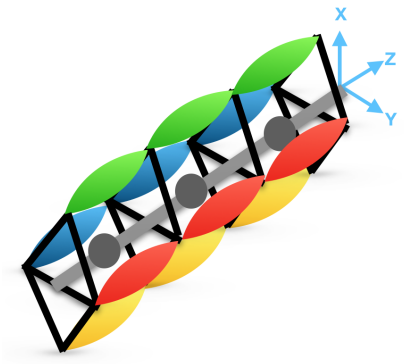
The following parameters were used in the creation of this model:

1. The spinal cord is continuous, but made up of about 100 segments
2. Connectivity is mostly local, but spans several segments
3. Connectivity is asymmetric
4. Individual neurons code muscle tension over small regions
5. During swimming, neural firing on one side of the lamprey oscillates in counter-phase with a similar population on the other side (i.e., biphasic oscillation is observed)
6. The length of the lamprey is equal to about 1 period of the swimming wave
7. The inter-segmental phase lag is independent of swimming frequency (i.e., 6. is true regardless of swimming speed)
8. The lamprey can swim between about .25 and 10Hz forwards,and can swim backwards
9. The lamprey has muscle fibers at four different positions (top-left, top-right, bottom-left, bottom-right) at each segment of the spinal cord

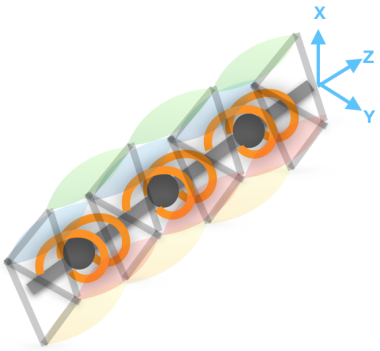
3 Muscle Tensions

In order to start building the model, we need to derive an equation for the muscle tensions along the length of the spinal cord as a function of time. This must be derived for roll, pitch and yaw motion.

Figure 1a shows the coordinate system definition. As seen from the figure, the different colored ellipsoids represent different sets of muscle fibers. The green and yellow pair of muscles are responsible for forward motion and yawing about Y axis. The blue and red pair of muscles are responsible for pitching about X axis. The grey line in the middle represents the spinal cord segment with the dark grey circle representing the joints between the segments.



(a) Coordinate system definition for the lamprey



(b) location of roll muscle fibers in the lamprey (in orange)

Figure 1: Lamprey coordinate system and muscle fibers

Figure 1b shows the set of muscles (in orange) responsible for rolling about Z axis. Note that this set of muscle fibers are wrapped around the spinal cord segment in a helix manner thus causing a torque about the spinal cord as they extend and contract. Note that only 3 segments are shown for illustration purposes. In reality, the model will have 100 segments to mimic the physiology found in Lampreys.

3.1 Muscle tensions for forward motion

The same framework discussed in the text will be used to derive the muscle tensions needed for forward motion.

Figure 2 shows the definition of individual muscle tensions. The coordinate frame is also shown for reference. Note, that the view is from the top of the Lamprey which is why only the green and yellow pairs of muscle fibers are shown. The muscle fibers for roll motion will be visible from this view but is omitted to keep the figure clean.

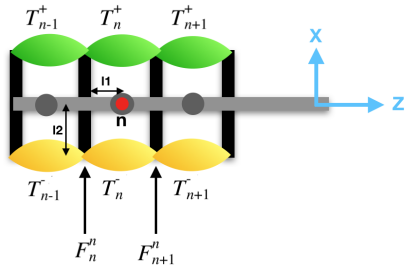


Figure 2: Free body diagram of muscle tensions for forward motion

The normal force experienced by the muscle fibers as the Lamprey moves forward is still the same. Please refer to the textbook [1] for the derivation of the following equation:

$$F^z(z,t) = -\eta A^2 \omega k cos^2(\omega t - kz) \tag{1}$$

where

1. $F^z(z,t)$: the component of the normal force acting on the Lamprey that is responsible for its forward motion
2. η : the viscosity of the water
3. ω : the frequency of oscillatory motion
4. $k = \frac{2\pi}{L}$
5. L : the length of the Lamprey
6. A : the amplitude of oscillations

Writing the summation of moments about point n, we have:

$$l_1 F^n_{n+1} = l_2(T^+_n - T^-_n - T^+_{n+1} + T^-_{n+1}) \tag{2}$$
$$l_1 F_n = l_2(T^+_{n-1} + T^-_n - T^+_n - T^-_{n-1}) \tag{3}$$

Letting $\gamma = \frac{l_1}{l_2}$ and after following the same steps as in the textbook [1], we get:

$$T(z,t) = \kappa(sin(\omega t - 2\pi z/L) - sin(\omega t)) \tag{4}$$

where $\kappa = \frac{\gamma\eta\omega A}{k}$ and $T_n = T^+_n - T^-_n$. Note that this is the same equation that was derived in the textbook.

3.2 Muscle tensions for yawing motion

Next, we need a way to describe muscle tensions due to yawing motion. Yawing is parametrized by $\ddot{\phi}$, the desired angular acceleration about the yaw axis, Y

A series of assumptions are made to simplify the derivation:

1. Effects of gravity and other external forces are ignored.
2. Counter torque due to friction from the turn maneuver is assumed to be negligible.
3. The lamprey is modeled as a series of decoupled rigid bodies in close proximity to one another, as opposed to a single connected multi-body.

Side note about assumption 3

Assumption 3 is a highly simplifying assumption. The Lamprey should realistically be modeled as a single connected multibody system. Newton-Euler or Euler-Lagrange method should be used to derive the torques at each of the 100 joints. However, this method will result in a highly complex set of equations that describe the dynammics. Hence, this assumption is made to simplify the derivations.

Using the above assumptions, the equations of motion for each segment, n can be written as follows:

$$\sum M_n = I\ddot{\phi} \tag{5}$$
$$l_2(T^-_n - T^+_n) = I\ddot{\phi} \tag{6}$$
$$-l_2(T_n) = I\ddot{\phi} \tag{7}$$
$$T_n = -\frac{I\ddot{\phi}}{l_2} \tag{8}$$

where $T_n = T^+_n - T^-_n$ as before.

Note that the yawing motion and forward motion are controlled by the same set of muscle fibers. Hence, the tensions can be superimposed to give:

$$T(z,t) = \kappa(sin(\omega t - 2\pi z/L) - sin(\omega t)) - \frac{I\ddot{\phi}}{l_2} \tag{9}$$
$$T(z,t) = \kappa\left(sin(\omega t - 2\pi z/L) - sin(\omega t) - \frac{I\ddot{\phi}}{\kappa l_2}\right) \tag{10}$$

Before we proceed further, lets verify that assumption 3 results in a reasonable, expected motion. From the equations of motion describing the yawing motion, we can derive an equation for the relative angle between the segments by integrating the equation twice to give us:

$$\int \dot{\phi} dt = \int -\frac{l_2}{I} T_n t + A dt \tag{11}$$
$$\phi = -\frac{l_2}{2I} T_n t^2 + At + B \tag{12}$$

Then, we can use the initial conditions, $\phi(0) = 0$ and $\dot{\phi}(0) = 0$ which say that the initial pitching angle and pitching velocity is 0 (signifying that the Lamprey is initially moving forward in a straight line (or is stationary). This results in the constants $A = B = 0$. Note that the ϕ represents the relative angle between the segments as shown in figure 3.

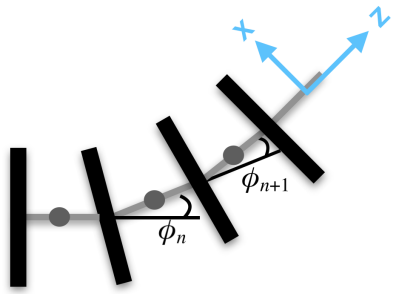
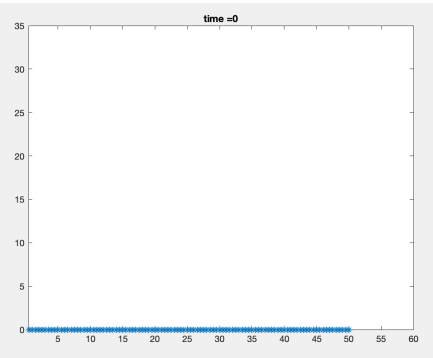
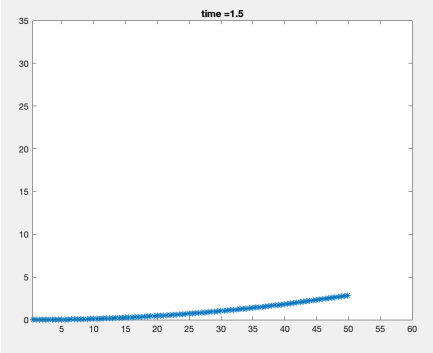


Figure 3: Definition of relative yaw angle between segments

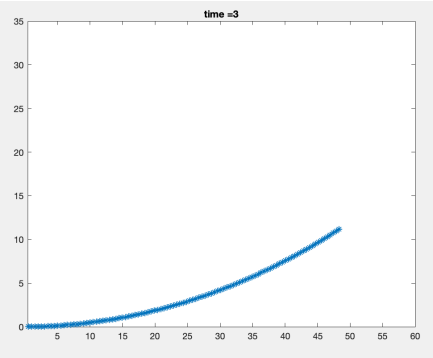
Thus, we can simulate the motion using the equation for ϕ. Figure 4 shows the simulated results. The simulation uses a Lamprey model composed of 100 segments, with a total length of 1m and an angular acceleration of $\ddot{\phi} = 0.001 rad/s^2$. (Please refer to MATLAB code, relative_ang_motion.m in Appendix A-2 for the code that simulates this motion). As seen from the figure, the motion is as expected. (note that this simulation only shows the action of the yawing force. The tensions due to the forward oscillatory motion are not included in the above simulation).



(a) Time $t = 0s$



(b) Time $t = 1.5s$



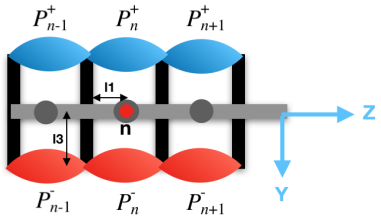
(c) Time $t = 3s$

Figure 4: Simulated yaw motion at different time steps ($\ddot{\phi} = 0.001 rad/s^2$)

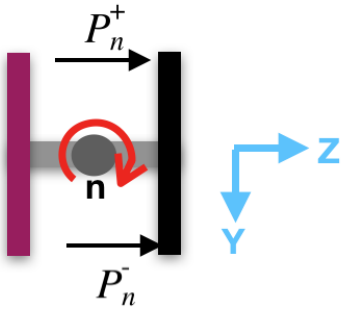
3.3 Muscle tensions for pitching motion

Pitching motion is accomplished by the blue and red pair of muscle fibers about the x axis. This motion is parametrized by the pitching angular acceleration, $\ddot{\theta}$. The equations describing the muscle tensions for the pitching motion can be derived in a similar way to that used to derive the tensions for yawing motion.

Figure 5a shows the mechanical model used to derive the pitching equations. Note that P is the muscle tension.



(a) Mechanical model for pitching muscle tension derivation



(b) Free body diagram of pitching muscle forces

As before, the same set of assumptions that was used for the yawing motion are applied here to derive equations for the pitching motion. These assumptions result in a free body diagram as seen in Figure 5b. Note that the purple link represents a fixed link while the black link is free to move. This is a result of assumption 3 which causes the purple link to be decoupled from the black link. The black link experiences rigid body rotation about the X axis due to the action of P forces.

The muscle tensions can then be derived as follows:

$$\sum M_n = I\ddot{\theta} \tag{13}$$

$$l_3(P_n^- - P_n^+) = I\ddot{\theta} \tag{14}$$

$$-l_3(P_n) = I\ddot{\theta} \tag{15}$$

$$P_n = -\frac{I\ddot{\theta}}{l_3} \tag{16}$$

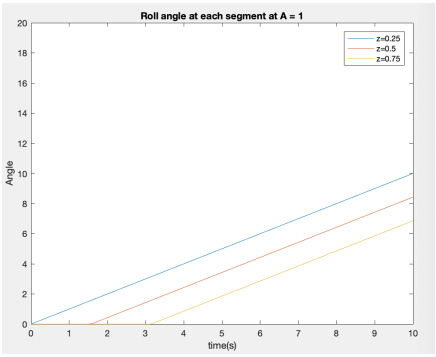
where $P_n = P_n^+ - P_n^-$ can be thought of as total tension in a segment, n.

3.4 Muscle tensions for rolling motion

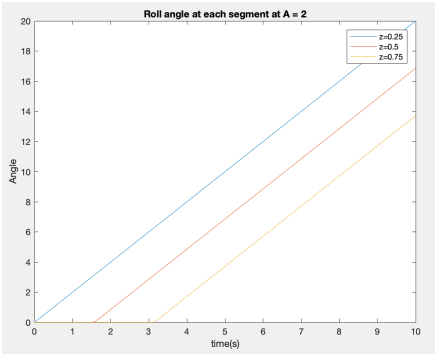
The kinematics underlying the roll motion in Lampreys is heavily researched. The exact mechanism underlying the motion is still unknown and several theories have been suggested. One such theory is the twisting of the body [2]. The idea is that oblique flexion of dorsal and left muscles (or ventral and right) creates a moment about the length of the lamprey, resulting in a helix motion superimposed on the undulatory locomotor motion. Since the kinematics behind this is still being researched, the following model will use a simplified version of the rolling motion where each segment of the spinal cord rolls about its center axis (thus instead of a spiral motion, the Lamprey will roll about its center.) Moreover, the observed roll behavior of a Lamprey shows that there is a time delay as each segment starts rolling one after another. This time delay introduces a phase difference between the roll angles of each segment in the lamprey. An equation describing the relative roll angles of the lamprey is:

$$\theta(z,t) = \left(A(t - \frac{2\pi z}{L} + \frac{2\pi z_1}{L}\right) u\left(t - \frac{2\pi z}{L} + \frac{2\pi z_1}{L}\right) \tag{17}$$

where θ is the angle relative to the starting angle (note that this θ is not to be confused with the $\ddot{\theta}$ describing the pitching angular acceleration) and $u(t)$ is a Heaviside function. Note that the term z_1 in the above equation refers to the z coordinate of the first segment (this was added so that the first segment starts rolling as soon an input is detected). Also, A refers to the angular velocity input. Figure 5 shows a plot of this equation for different A at different points along the segment (Please refer to the MATLAB code roll_angle_simulation.m in Appendix A-3).



(c) Roll angular velocity, $A = 1rad/s$



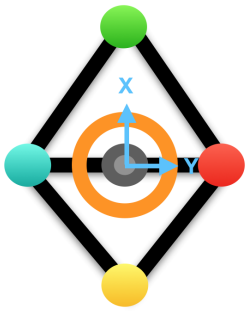
(d) Roll angular velocity, $A = 2rad/s$

Figure 5: Roll angle simulated at different roll angular velocities for a 3 segment model

Differentiating the above equation with respect to time, equating it to the navier stokes equation and after some simplification, we have:

$$\tau = \eta A u\left(t - \frac{2\pi z}{L} + \frac{2\pi z_1}{L}\right) \tag{18}$$

where τ is the torque applied at the segment and η is a constant describing the friction of water. Note that we will use $\eta = 1$ from this point onward. The torque can be converted to muscle tensions by looking at the free body diagram of the roll muscle. Figure 6 shows a frontal view of the lamprey and how the segment turns relative to other segments when provided with a rolling torque.



(a) Front view of the lamprey showing the roll muscle fibers in orange

(b) Relative motion of segment during roll

Figure 6: Rolling motion of lamprey

Figure 7 shows the free body diagram during this motion. In the figure, the orange circular line represents the helix muscle fibers that were presented earlier. The same assumptions that were used for deriving the yawing and pitching equations of motion will be used here. A result of assumption 3 (that the Lamprey is treated as separate rigid bodies in close proximity) is that the torques exerted by the muscle fibers can be thought of as external torques applied locally on each segment of the spinal cord (thus the reaction torque does not transfer over to the other segments)

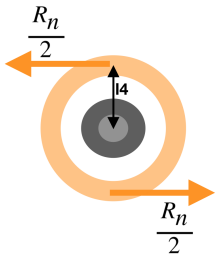


Figure 7: Free body diagram of lamprey during roll

From this, we can derive the final equation describing the muscle tension for roll motion:

$$T_r(t,z) = 1/l_3\tau \tag{19}$$

$$T_r(t,z) = 1/l_3 Au\left(t - \frac{2\pi z}{L} + \frac{2\pi z_1}{L}\right) \tag{20}$$

Note that the t in the above equation refers to the time when the roll motion started. Therefore, $t = t_{sim} - t_{start}$ where t_{sim} is the current time of the simulation and t_{start} is the time that the rolling motion started. Also, if $L = 1$ and $l_3 = 1$, the above equation simplifies to:

$$T_r(t,z) = Au\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right) \tag{21}$$

4 Dynamics

4.1 Forward/Yaw dynamics

A similar approach to the one taken in the textbook is followed here. By choosing a Fourier series as an orthonormal basis to represent the muscle tensions, and calculating the error, we have:

$$error = \left\langle \left(sin(\omega t) - \frac{I\ddot{\phi}}{l_2\kappa} + A_0\right)^2 + 0.5(cos(\omega t) + A_1)^2 + 0.5(sin(\omega t) - A_2)^2 + \sum_{n=3} A_n^2 \right\rangle_t \tag{22}$$

This tells us that for the errors to be 0, we need:

$$A_0 = -sin(\omega t) + \frac{I\ddot{\phi}}{l_2\kappa} \tag{23}$$

$$A_1 = -cos(\omega t) \tag{24}$$

$$A_2 = sin(\omega t) \tag{25}$$

$$A_n = 0 \tag{26}$$

We can differentiate the As to get:

$$\dot{A}_0 = -\omega cos(\omega t) + \frac{I\dddot{\phi}}{l_2\kappa} \tag{27}$$

$$\dot{A}_1 = \omega sin(\omega t) \tag{28}$$

$$\dot{A}_2 = \omega cos(\omega t) \tag{29}$$

$$\dot{A}_n = 0 \tag{30}$$

which can be written in matrix form as follows:

$$\dot{A} = M_d A + M_c \frac{I\dddot{\phi}}{l_2\kappa} \tag{31}$$

where:

$$M_d = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & -\omega & 0 \end{bmatrix}; M_c = \begin{bmatrix} 1 \\ \omega \\ 0 \end{bmatrix}$$

$\Theta = [\tilde{\phi}\Phi]^{-1}$ can be used to project from the orthonormal basis amplitudes to the segment activities. (Please refer to Appendix F2.5 in the textbook [1] for a detailed derivation of these terms). Note that a Gaussian encoding function is also used here similar to the one used in the textbook. Therefore, the final dynamics equation is:

$$\dot{a} = m_d a + m_i u + m_c \frac{I\dddot{\phi}}{l_2\kappa}$$

where a is the segment activity, $m_d = \Theta^{-1} M_d \Theta$, $m_i = \Theta^{-1} M_d \Theta$, $m_c = \Theta^{-1} M_c$ and u is a startup step function which is equal to a until $t = t_s$. Note that the matrix,M_i can be used to control the amplitudes of the oscillations.

4.2 Roll and pitch dynamics

The dynamics of roll and pitch are less complex than that of the forward motion. We will implement neurons that simulate the dynamics of the pitch and roll tensions. In order to convert the muscle tensions into segment activities, we will use a similar methodology to the one used in the forward/yaw dynamics. That is, $a_p = \tilde{\phi} P(z,t)$ where a_p is the pitching muscle activity, $\tilde{\phi}$ is the Gaussian encoding function and P is the pitching muscle tensions.Since the pitching tension, $P_n = -\frac{I\ddot{\theta}}{l_3}$ only depends on the input, $\ddot{\theta}$, the dynamics of P_n also only depend on the input. Therefore:

$$\dot{P_n} = \frac{I\dddot{\phi}}{l_3} \tag{32}$$

$$\text{let } x_1 = P - \frac{I\ddot{\phi}}{l_3} \tag{33}$$

$$\dot{x_1} = 0 \tag{34}$$

Figure 8 shows the system diagram for this motion.

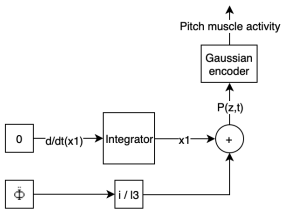


Figure 8: System diagram for pitch dynamics

Similarly, for roll motion, the muscle activities, $T_r = Au\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right)$ can be differentiated with respect to time to determine the dynamics of the system.

$$\dot{T_r} = \dot{A}u\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right) + A\delta\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right) \tag{35}$$

$$\dot{T_r} = \dot{A}u\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right) \qquad \text{Ignoring the } \delta(t) \text{ term} \tag{36}$$

$$\text{Let } x_2 = T - Au\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right) \tag{37}$$

$$\dot{x_2} = 0 \tag{38}$$

Figure 9 shows the system diagram for roll motion. Note that $u*(t) = u\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right)$ in this figure. In essence, this simply introduces a lag in the starting time of the muscle tensions along each segment.

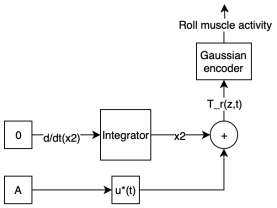


Figure 9: System diagram for roll dynamics

5 Implementation

Refer to the Appendix A-1 for code that implements the dynamics of the system. Figure 10 shows an overview of the model implemented using NEF in nengo. To simplify things, a 3 segment lamprey model was implemented with all constants set to 1.

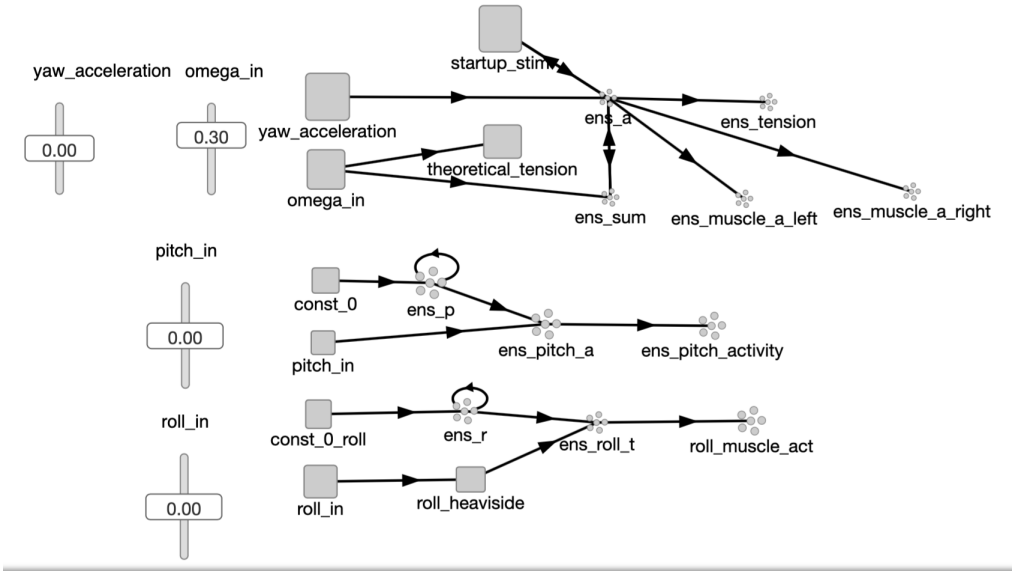


Figure 10: Nengo model

In short, the model consists of four inputs, omega_in, yaw_acceleration, roll_in and pitch_in describing the forward angular velocity, yaw angular acceleration, roll angular velocity an pitch angular acceleration, respectively. The model outputs three different segment activities, ens_pitch_activity, roll_muscle_act and ens_a corresponding to top (pitching), spinal (rolling) and side (yawing/forward) muscle activities, respectively. Additionally, the model has nodes that calculate the theoretical tension for the oscillatory forward motion as well as nodes that send in a startup stimulus signal to ens_a (the forward dynamics model). The model also has ensembles that represent the segment activity in the forward dynamics model in terms of left and right muscles (ens_muscle_a_left and ens_muscle_a_right).

5.1 Yaw and forward motion

Figure 11 shows an isolated view of the forward dynamics model. In this model, ens_a has a recurrent connection with a transformation of $(\tau m_d + I)x$ where x represents the activity. This is essentially implementing the oscillator dynamics. Since ω changes with time, the dynamics matrix, m_d is calculated dynamically with time using ens_sum ensemble of neurons.

The startup_stim node is equal to $(\tau m_I)ens_a$ for a short duration of 0.2s. This is necessary in order to start the oscillations.

ens_tension decodes the muscle activity into segment tensions using the equation, $\hat{T} = \sum a_i(t)\phi_i(z)$.

The neuron ensembles, ens_muscle_a_left and ens_muscle_a_right calculate individual neuron activities (left and right) for the first segment. This is done by representing the segment activity with two ensembles of neurons with encoders +1 and -1.

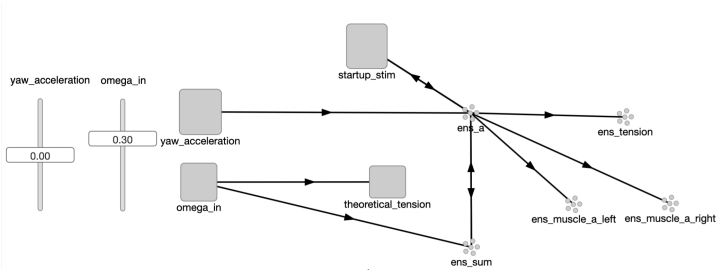


Figure 11: Nengo model for the forward/yaw dynamics

5.2 Pitching motion

Figure 12 shows an isolated view of the pitching dynamics model. The node, const_0 is a constant value of 0 that is fed into the ensemble of neurons, ens_p. ens_p represents $x1$ from the system diagram (see Fig. 8). The recurrent connection to ens_p enables it to implement the dynamic equation, $\dot{x_1} = 0$. The ensemble of neurons, ens_pitch_a sums up the activity from ens_p as well as a transformed pitch input to effectively give the pitching muscle tensions. This is then converted to activity space and represented by the ensemble of neurons, ens_pitch_activity. In order to do this conversion, the same Gaussian function that was used for the forward motion is used as follows:

$$a_p = \left\langle \tilde{\phi}_i(z) P(t,z) \right\rangle$$

where a_p is the pitching muscle activity, $\tilde{\phi}_i(z)$ is the Gaussian function and $P(t,z)$ is the pitching muscle tensions.

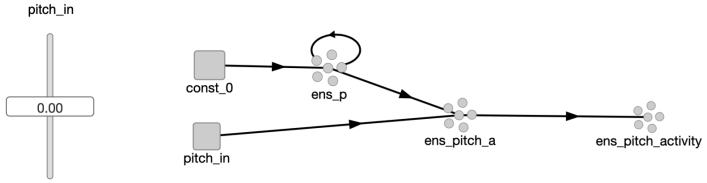


Figure 12: Nengo model for the pitching dynamics

5.3 Rolling motion

Figure 13 shows an isolated view of the rolling dynamics model. Similar to the pitching model, the node, const_0_roll is a constant value of 0 that is fed into the ensemble of neurons, ens_r. ens_r represents $x2$ from the system diagram (see Fig. 9). The recurrent connection to ens_r enables it to implement the dynamic equation, $\dot{x_2} = 0$. The ensemble of neurons, ens_roll_t sums up the activity from ens_r as well as a transformed pitch input to effectively give the rolling muscle tensions. This is then converted to activity space and represented by the ensemble of neurons, roll_muscle_activity. In order to do this conversion, the same Gaussian function that was used for the forward motion is used as follows:

$$a_r = \left\langle \tilde{\phi}_i(z) T_r(t,z) \right\rangle$$

where a_r is the rolling muscle activity, $\tilde{\phi}_i(z)$ is the Gaussian function and $T_r(t,z)$ is the rolling muscle tensions.

Note that the node, roll_heaviside is used to implement the step function (to provide the delay in the muscle tensions), $Au\left(t_{sim} - t_{start} - 2\pi z + 2\pi z_1\right)$.

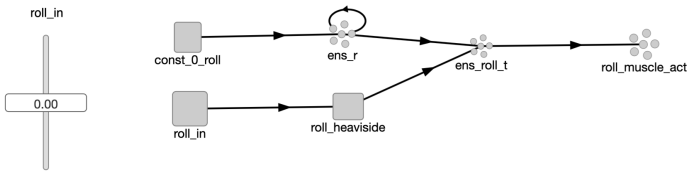
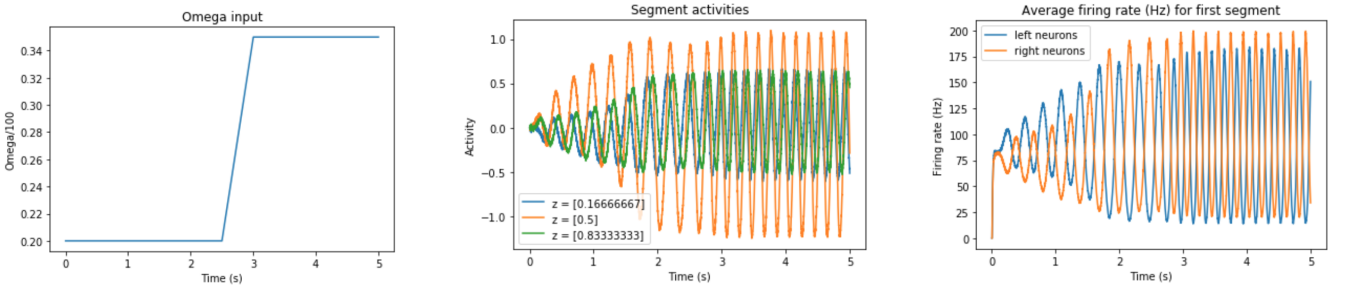


Figure 13: Nengo model for the rolling motion dynamics

6 Discussion

6.1 Varying forward angular velocity, ω

The model was run with different parameters for verification. First, the model was run with no pitch, roll or yaw inputs. The forward angular velocity, ω was increased to see the effect on the segment activities. Figure 14 shows the ω that was input to the model and the corresponding segment activities and the individual neuron activities in the left and right side of the lamprey. As predicted, the neuron firing rates are exactly out of phase with each other (biphasic oscillator behavior). Moreover, it can be seen that the oscillations takes about 2s to stabilize at the beginning. This can be changed by changing the parameters in the startup matrix, M_I. Finally, the frequency of oscillations can be seen to increase when ω is increased.



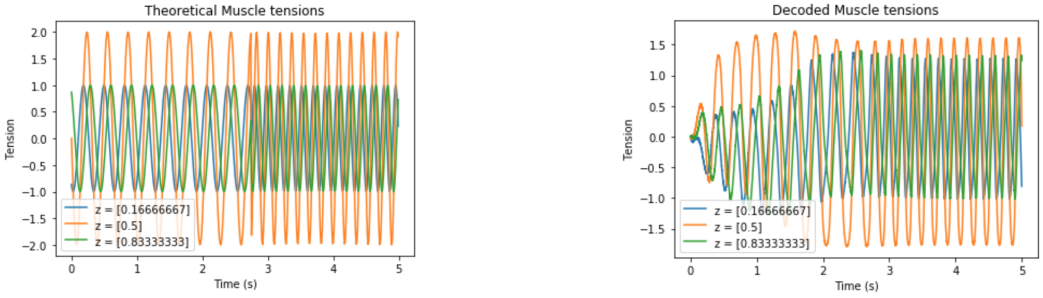
(a) ω input

(b) Segment activity for three segment model

(c) Average left and right side neuron firing rates for the first segment

Figure 14: Response of model to varying ω input

Figure 15 shows a comparison between the theoretical and decoded tensions from the dynamics of the system. (Note that the decoded tensions were calculated by $\hat{T} = \sum a_i(t)\phi_i(z)$ as mentioned in the textbook). As can be seen from the figures, the amplitudes of the oscillations of the tension as predicted by the model seem to be shifted up from the theoretical tension. This is most likely due to the influence of noise on the startup dynamics which determine the amplitude of the oscillations. Increasing the number of neurons and fine tuning the startup parameters will result in better performance.



(a) Theoretical tensions for varying ω

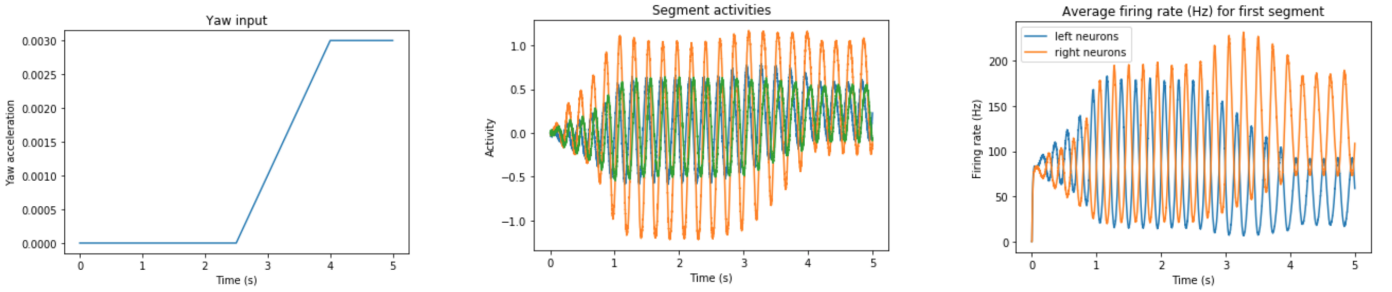
(b) Decoded tensions for varying ω

Figure 15: Comparison of theoretical and decoded tensions

6.2 Varying yaw angular acceleration, $\ddot{\phi}$

Next, the model was run with a varying yaw input. Figure 16 shows the yaw input that was provided to the system and the corresponding response in terms of segment activities as well as the response of the left and right sides of neurons on the lamprey. As seen from the figure, several observations can be made about the behavior of the system:

1. The segment activity oscillations initially stabilize at around 1s.
2. The yaw input causes an asymmetric sinusoidal wave of segment activities that is shifted up (as expected) but with a smaller total amplitude change (unexpected). This is most likely due to saturation effects of the neurons. Increasing the radius of the ensemble and fine tuning the startup matrix to oscillate with a smaller amplitude in the beginning will likely correct this unexpected behavior.
3. The biphasic oscillatory behavior is still seen in the neurons from the left and right side. One side of neurons oscillate at a higher amplitude when a yaw input is provided (as expected).



(a) Yaw input, $\ddot{\phi}$

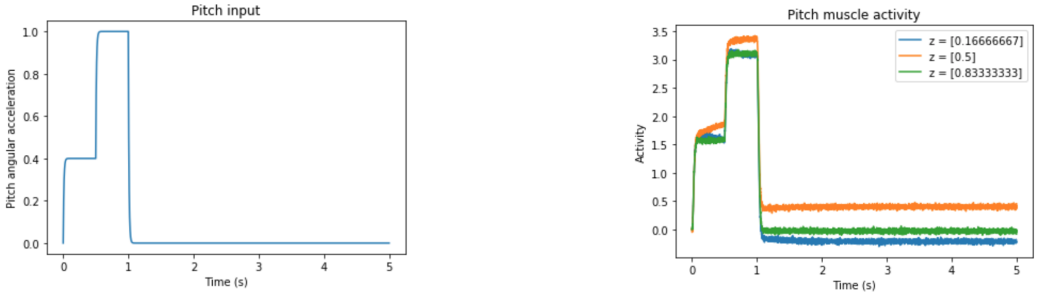
(b) Segment activities for varying $\ddot{\phi}$

(c) Average left and right side neuron firing rates for varying $\ddot{\phi}$

Figure 16: Response of system to a yaw input

6.3 Varying pitch angular acceleration, $\ddot{\theta}$

Next, the model was run with varying pitch input. Figure 17 shows the pitch input that was provided to the system and the corresponding activity in the muscles. Recall that the pitching motion dynamics simply involved implementing $x_1 = 0$ where x_1 was defined previously. This recurrent connection is simply responsible for holding x_1 steady. Hence, the figures are as expected. The only notable observation is that when the pitch input is 0, the muscle activity is not completely 0 as expected. This is because noise in the ensemble of neurons that represent the constant 0 input can cause the integrator (which is basically a single recurrent connection) to stabilize at a specific noise level (see Fig. 8).



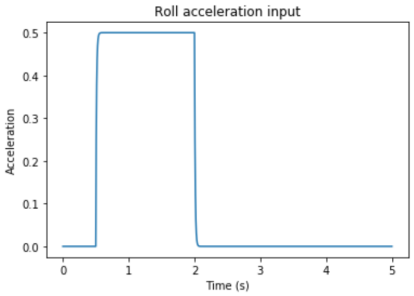
(a) Pitch input, $\ddot{\theta}$

(b) Segment activities for varying $\ddot{\theta}$

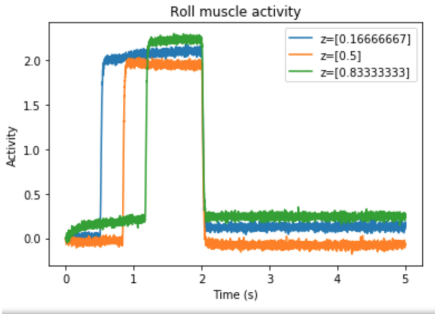
Figure 17: Response of system to a pitch input

6.4 Varying Roll angular velocity, A

Next, the model was run with varying roll input. Figure 18 shows the roll input that was provided to the system and the corresponding activity in the muscles. As seen from the figure, the response is as expected. There is a time delay between the first and last segments in terms of when the activity (hence the torque on the segment) starts. A similar artifact to the pitch response is seen here where the activity is not completely 0 when the roll input is 0. This is again because noise in the ensemble of neurons that represent the constant 0 input can cause the integrator (which is basically a single recurrent connection) to stabilize at a specific noise level (see Fig. 9).



(a) Roll input, A



(b) Segment activities for varying A

Figure 18: Response of system to a roll input

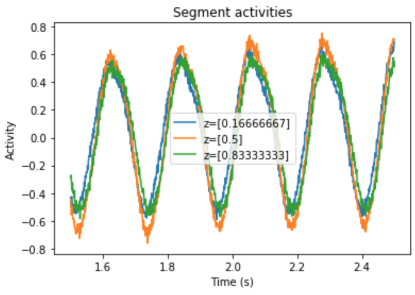
7 Investigating the effect of the encoding function

Next, we would like to investigate the effect of the Gaussian encoding function on the segment activities. As stated in the textbook, the relationship between the encoding function and the segment activity is as follows:

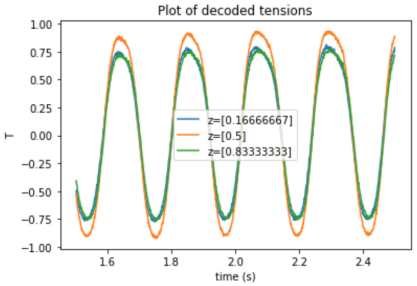
$$a_i(t) = \left\langle \tilde{\phi}_i(z) T(t,z) \right\rangle_z \tag{39}$$

where $\tilde{\phi}_i(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{z-z_{\mu i}}{2\sigma}\right)^2}$. Note that $z_{\mu i}$ refers to the center z value of a specific segment, i.

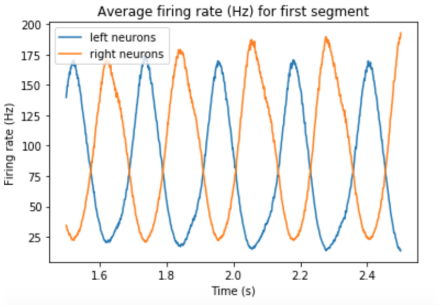
The model was run with different σ to study the effect of the encoding function. Figures 19, 20 and 21 show a plot of the segment activities, the decoded tensions and the average firing rates in the left and right sides of the lamprey for different σs. As seen from Figure 19, a higher σ has the effect of averaging out the individual segment activities causing all of the segment activities to be very similar (as seen in Figure 19a). A lower σ tends to preserve the localized interaction of segment activities (as seen from Figure 20a). However, after a certain point, the σ becomes too low to have any oscillatory behavior (as seen in 21a). This is because the Gaussian encoder is used in the calculation of Θ matrix which helps project the amplitudes from the higher order orthonormal space to activity space. Therefore, a small σ effectively causes the projection operator to be negligible resulting in an almost negligible segment activity (as seen from 21a).



(a) Segment activities

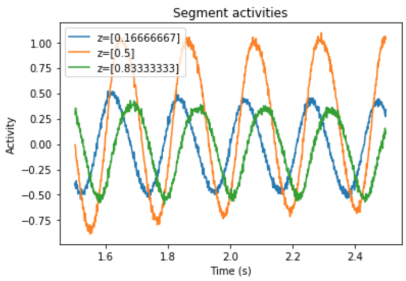


(b) Decoded muscle tensions

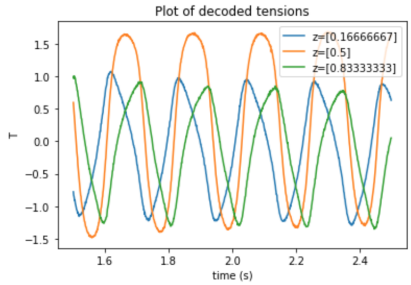


(c) Left and right side average neuron firing rates for one of the segment

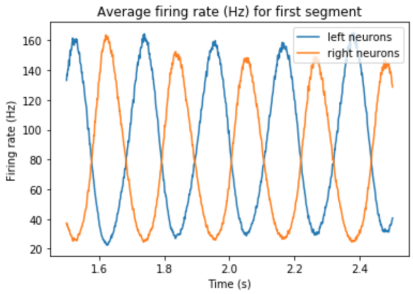
Figure 19: Response of system to $\sigma = 0.5$



(a) Segment activities

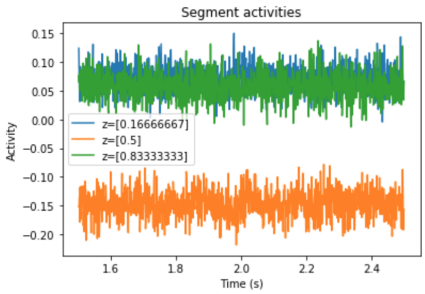


(b) Decoded muscle tensions

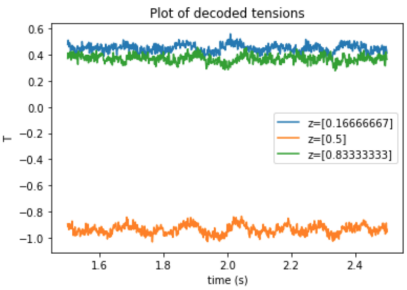


(c) Left and right side average neuron firing rates for one of the segment

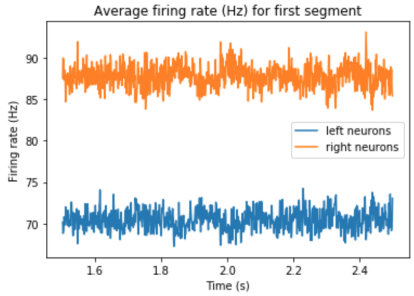
Figure 20: Response of system to $\sigma = 0.1$



(a) Segment activities



(b) Decoded muscle tensions



(c) Left and right side average neuron firing rates for one of the segment

Figure 21: Response of system to $\sigma = 0.06$

8 Recommendations

The results show that the model is not perfect and can be improved. There are several improvements that can be made:

1. The model uses a simplified version of the actual mechanical model with no force interactions between the segment. A more realistic version would use dynamical equations of motion derived using Euler-Lagrange or Newton-Euler method.
2. The model uses three independent sets of muscle fibers to control the pitch, yaw and roll motions. Recent studies have shown, however, that there are four sets of muscle fibers around the lamprey and different patterns of excitation of these muscles produce the desired motion [2].
3. The model had issues with the oscillator amplitudes being stable. A higher number of neurons and a more thorough evaluation of error terms with more damping of higher order terms will result in a more stable oscillator.

Furthermore, a mechanical model that takes in as input the muscle tensions and simulates the motion of the Lamprey can be developed to better evaluate the performance of the model.

9 Conclusion

The model presented above is a very basic version of roll, pitch and yaw control in a lamprey (in addition to simulating the muscle tensions for forward motion). The segment activities derived from the model match the expected pattern.

There were some issues with the model such as not being able to hold a stable amplitude of oscillations and steady state error not being 0. Recommendations were suggested that could help fix these issues.

References

[1] Anderson C. H. Eliasmith, C. *Neural Engineering Computation, representation and dynamics in neurobiological systems.* MIT Press, 2003.

[2] G. N. Orlovsky S. Grillner F. Ullen, T. G. Deliagina. Spatial orientation in the lamprey. i. control of pitch and roll. *Journal of Experimental Biology*, 198:665–673, 1995.

10 Appendix A-1 (main.py)

```
import numpy as np
import nengo
import matplotlib.pyplot as plt
from scipy.integrate import quad
from nengo.utils.ensemble import tuning_curves
import pdb
from nengo.utils.functions import piecewise

#Lamprey specs
l = 1.0    # Length of lamprey in z direction
l3 = 1.0   # Length of lamprey segment in y direction
w = 30     #forward angular velocity omega
n = 3      #Number of segments
I_p = 1    #Moment of inertia about pitch axis.

#Model params
alpha_damp = -0.3    #damping parameter for higher order terms in M_d
T = 5                #Simulation time in seconds
sigma = 0.1         #Standard deviation of gaussian encoding function
num_neurons = 1000   #number of neurons per dimension
tau = 0.01           #Synaptic time constant

#variables needed for roll motion
time_start = 0            #the time at which the rolling input was triggered
roll_triggered = False    #boolean indicating whether rolling motion has been triggered

#Find the z coordinate of the segment centers
segment_centers = np.zeros([n,1])
for i in range(n):
    segment_centers[i] = l*(1+2*i)/(2.0*n)

#Find M_dynamics matrix
M_A = np.array([[0,w,0],[-w,0,0],[0,-w,0]])
M_damp = np.array([[alpha_damp, 0, alpha_damp],[0, 0, 0], [alpha_damp, 0, alpha_damp]])
M_d = M_A + M_damp

#Find M_I matrix
M_I = np.array([[8,0,-8],[0,1,0],[-8,0,8]])

#Calculate theta to convert to activity space
def gauss_encoder(z,segment_center_z,sigma=0.01):
    return 1/(sigma*(2*np.pi)**0.5) * np.exp(-1*np.square(z-segment_center_z)/(2*sigma**2))

def phi(z):
    return [1, np.sin(2*np.pi*z),np.cos(2*np.pi*z)]

encoders = np.zeros([n,1])
phi_mat = np.zeros([1,3])
```

```
theta_inv = np.zeros([n,3])

for i in range(n):
    for j in range(3):
        integrate_func = lambda z: phi(z)[j] * gauss_encoder(z,segment_centers[i],sigma)
        theta_inv[i,j] = quad(integrate_func,0,1)[0]

theta = np.linalg.pinv(theta_inv)

#calculate md,mi matrix
m_d = np.matmul(np.matmul(theta_inv,M_d),theta)
m_i = np.matmul(np.matmul(theta_inv,M_I),theta)

#Create the nengo model
model = nengo.Network('Simple model', seed=3)

A_prime = tau*m_d + np.eye(n)
B_prime = tau*m_i

with model:

    #Function that is used in the recurrent connection of forward dynamics.
    #Dynamically calculates the dynamics matrix and transforms the input, x
    def calc_md(x):
        w = x[0]*100
        activity = np.array(x[1:])

        M_A = np.array([[0,w,0],[-w,0,0],[0,-w,0]])
        M_damp = np.array([[alpha_damp, 0, alpha_damp],[0, 0, 0], [alpha_damp, 0, alpha_damp]])
        M_d = M_A + M_damp
        m_d = np.matmul(np.matmul(theta_inv,M_d),theta)
        A_prime = tau*m_d + np.eye(n)

        return np.matmul(A_prime,activity)

    #Given an input, x = [a,b,c], returns [0,a,b,c]
    def transform_a(x):
        a = np.zeros([n+1,n])
        ind = 0
        for i in range(n+1):
            if i == 0:
                continue
            a[i,ind] = 1
            ind = ind + 1

        return np.matmul(a,x)

    #Given an input, x = [w], returns [w,0,0,0]
    def transform_omega(x):
        a = np.zeros([n+1,1])
        a[0] = 1
        #pdb.set_trace()
        return np.matmul(a,x)

    #Given the current time step and omega, it calculates the theoretical tension
    def calc_tension(t,om_in):
        omega = om_in * 100.0
        T = []
        for i in range(n):
            T = np.append(T,np.sin(omega*t - 2*np.pi*segment_centers[i]) - np.sin(omega*t),axis=
```

```
    return T

#Given, muscle activity, it calculates muscle tension by multiplying and summing up with gau
def calc_decoded_tension(x):
    T = []
    for i in range(n):
        t = 0
        for j in range(n):
            t += gauss_encoder(segment_centers[i],segment_centers[j],sigma) * x[j]
        T = np.append(T,t,axis=0)

    return T

#Provides the model with the startup stimulus (current A) until t = 0.2s
def calc_startup_stim(t,a):
    if t < 0.2:
        return np.matmul(B_prime,a)
    else:
        return np.zeros(n)

#Given a vector of activities for segments, returns the activity for the first segment
def calc_muscle_a(a):
    return a[0]

def yaw_input(t):
    gradient = 0.0007
    f = piecewise({0:0.0,2.5:gradient*t - gradient*2.5,3.5:gradient*3.5 - gradient*2.5})
    return f(t)

def omega_input(t):
    f = piecewise({0:0.2, 2.5:0.3*t-0.55, 3: 0.35})
    w = f(t)*100
    return f(t)

#Converts from orthonormal basis space to activity space
def calc_yaw(x):
    C_prime = np.matmul(theta_inv,[1,w,0])
    return C_prime*x

#Transforms the input pitch acceleration, x to a force
def inertia_calc(x):
    return x*np.ones([n])*I_p/13

#Transforms muscle tensions to muscle activity by multiplying with gaussian function
def calc_muscle_activity(x):
    A = []
    for i in range(n):
        t = 0
        for j in range(n):
            t += gauss_encoder(segment_centers[i],segment_centers[j],sigma) * x[j]
        A = np.append(A,t,axis=0)

    return A

#Yaw+forward motion nodes

#Yaw acceleration inputs
yaw_acceleration = nengo.Node(yaw_input, size_out = 1)

#Startup stimulus for the oscillator
```

```
startup_stim = nengo.Node(calc_startup_stim, size_in = n)

#A node to calculate theoretical tension
theoretical_tension = nengo.Node(calc_tension, size_in=1)

#Forward angular velocity inputs
omega_in = nengo.Node(0.2, size_out=1)

#Yaw + forward ensembles

#Ensemble to represent muscle activities
ens_a = nengo.Ensemble(num_neurons, dimensions=n, radius=3)

#Ensemble to represent muscle tensions
ens_tension = nengo.Ensemble(num_neurons, dimensions=n)

#Ensemble to dynamically calculate the dynamics matrix
ens_sum = nengo.Ensemble(num_neurons*(n+1), dimensions=n+1)

#Ensemble to represent muscle activities in the right and left sides of the lamprey
ens_muscle_a_right = nengo.Ensemble(num_neurons, encoders = np.ones([num_neurons,1]), dimensio
ens_muscle_a_left = nengo.Ensemble(num_neurons, encoders = -1*np.ones([num_neurons,1]), dimen

#Yaw + forward motion connections and probes

nengo.Connection(omega_in, theoretical_tension)
nengo.Connection(yaw_acceleration, ens_a, function=calc_yaw, synapse=tau)
nengo.Connection(omega_in, ens_sum, function=transform_omega, synapse=None)
nengo.Connection(ens_a, startup_stim)
nengo.Connection(startup_stim, ens_a, synapse=tau)
nengo.Connection(ens_a, ens_tension, function=calc_decoded_tension, synapse=tau)
nengo.Connection(ens_a, ens_muscle_a_right, function = calc_muscle_a)
nengo.Connection(ens_a, ens_muscle_a_left, function = calc_muscle_a)
nengo.Connection(ens_a, ens_sum, function=transform_a, synapse=tau/2)
nengo.Connection(ens_sum, ens_a, function=calc_md, synapse=tau/2)

omega_probe = nengo.Probe(omega_in)
yaw_probe = nengo.Probe(yaw_acceleration)
muscle_a_probe_left = nengo.Probe(ens_muscle_a_left.neurons, 'spikes', synapse=tau)
muscle_a_probe_right = nengo.Probe(ens_muscle_a_right.neurons, 'spikes', synapse=tau)

decoded_t_probe = nengo.Probe(ens_tension, synapse=tau)
theoretical_t_probe = nengo.Probe(theoretical_tension, synapse=tau)
a = nengo.Probe(ens_a, synapse=tau)

#Pitch motion nodes and ensembles

#A node representing a constant of 0
const_0 = nengo.Node(np.zeros([n]))

#A node for the pitch input
pitch_in = nengo.Node(piecewise({0:0.4, 0.5:1, 1:0}))

#An ensemble to represent x1
ens_p = nengo.Ensemble(num_neurons, dimensions=n)

#An ensemble to represent pitching muscle tensions
ens_pitch_a = nengo.Ensemble(num_neurons, dimensions=n)

#An ensemble to represent pitching muscle activity
```

```
ens_pitch_activity = nengo.Ensemble(num_neurons, dimensions = n,radius = 5)

#pitching motion conncections and probes
nengo.Connection(const_0,ens_p)
nengo.Connection(ens_p,ens_p,synapse=tau)
nengo.Connection(pitch_in,ens_pitch_a,function = inertia_calc,synapse=tau)
nengo.Connection(ens_p,ens_pitch_a,synapse=tau)
nengo.Connection(ens_pitch_a,ens_pitch_activity,function=calc_muscle_activity,synapse=tau)

p = nengo.Probe(ens_pitch_a, synapse=tau)
pitch_in_probe = nengo.Probe(pitch_in,synapse = tau)
p_activity_probe = nengo.Probe(ens_pitch_activity,synapse=tau)

#Given t and roll angular velocity, it calculates u(t), heaviside function
def roll_heaviside_calc(t, roll_acc):
    global roll_triggered,time_start

    if abs(roll_acc - 0) < 1e-10:
        #reset trigger if input is 0
        roll_triggered = False
        return np.zeros([n])
    elif roll_triggered == False and abs(roll_acc) > 0:
        #set trigger as soon as theres an input
        time_start = t
        roll_triggered = True
        return np.zeros([n])
    elif roll_triggered == True:
        a = np.zeros([n])
        for i in range(n):
            a[i] = roll_acc * np.heaviside(t - time_start - segment_centers[i] / l + segment
        return a

#Roll motion ensembles and nodes

#Node to input roll angular velocity
roll_in = nengo.Node(piecewise({0:0, 0.5:0.5, 2:0}))

#Node that inputs a constant 0
const_0_roll = nengo.Node(np.zeros([n]))

#Node to calculate heaviside function
roll_heaviside = nengo.Node(roll_heaviside_calc,size_in = 1, size_out = n)

#Ensemble that represents x2
ens_r = nengo.Ensemble(num_neurons, dimensions=n)

#Ensemble that represents rolling muscle tensions
ens_roll_t = nengo.Ensemble(num_neurons,dimensions=n)

#Ensemble that represents rolling muscle activity
roll_muscle_act = nengo.Ensemble(num_neurons, dimensions = n,radius=5)

#Rolling motion connections and probes
nengo.Connection(ens_roll_t,roll_muscle_act,function=calc_muscle_activity)
nengo.Connection(roll_in, roll_heaviside)
nengo.Connection(roll_heaviside, ens_roll_t)
nengo.Connection(const_0_roll,ens_r)
nengo.Connection(ens_r,ens_roll_t)
nengo.Connection(ens_r,ens_r,synapse=tau)
```

```
    roll_activity_probe = nengo.Probe(roll_muscle_act,synapse = tau)
    roll_in_probe = nengo.Probe(roll_in,synapse = tau)


sim = nengo.Simulator(model)
sim.run(T)


def plot(x,y,title,xlab,ylab):
    plt.figure()
    plt.plot(x,y)
    plt.title(title)
    plt.xlabel(xlab)
    plt.ylabel(ylab)

def activity_plot(x,y,title,xlab,ylab):
    plt.figure()
    for i in range(n):
        plt.plot(x,y[:,i],label='z=' + str(segment_centers[i]))
    plt.title(title)
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.legend()

#Yaw + forward motion plots
plot(sim.trange(),sim.data[omega_probe],'Omega input','Time (s)','Omega/100')
plot(sim.trange(),sim.data[yaw_probe],'Yaw input','Time (s)','Yaw acceleration')
activity_plot(sim.trange(), sim.data[decoded_t_probe],'Plot of decoded tensions','time (s)','T')
activity_plot(sim.trange(), sim.data[a],'Segment activities','Time (s)','Activity')

plt.figure()
avg1 = np.average(sim.data[muscle_a_probe_left],axis=1)
avg2 = np.average(sim.data[muscle_a_probe_right],axis=1)
plt.plot(sim.trange(),avg1,label='left neurons')
plt.plot(sim.trange(),avg2,label='right neurons')
plt.title('Average firing rate (Hz) for first segment')
plt.xlabel('Time (s)')
plt.ylabel('Firing rate (Hz)')
plt.legend()

#Pitch plots
plot(sim.trange(),sim.data[pitch_in_probe],'Pitch input','Time (s)','Pitch angular acceleration')
activity_plot(sim.trange(),sim.data[p_activity_probe],'Pitch muscle activity','Time (s)','Activit

#Roll plots
plot(sim.trange(),sim.data[roll_in_probe],'Roll acceleration input','Time (s)','Acceleration')
activity_plot(sim.trange(), sim.data[roll_activity_probe],'Roll muscle activity','Time (s)','Act
```

11 Appendix A-2 (relative_ang_motion.m)

```
clear all
close all

dt= 0.1;
t = [0:dt:10];
[~,T] = size(t);

phi_ddot = 0.001;

n = 100;
start = 1;
end_x = 100;

l = 0.5;
phi = phi_ddot/2 * t.^2;

x = zeros(n,T);
y = zeros(n,T);

for i = [1:1:n]
    temp_x = zeros(1,T);
    temp_y = zeros(1,T);
    for j = 1:1:i
       temp_x = temp_x + l*cos(phi*j);
       temp_y = temp_y + l*sin(phi*j);
    end
   x(i,:) = temp_x;
   y(i,:) = temp_y;
end

figure();
for time = [1:1:T]
    clf
    plot(x(:,time),y(:,time),'*');
    title(strcat('time =  ',num2str(t(time)) ));
    xlim([0.4,60]);
    ylim([0,35]);
    pause(dt);
end
```

12 Appendix A-3 (roll_angle_simulation.m)

```
clear all
close all

a1 = 1;
a2 = 2;
l = 1;
t = [0:0.1:10];
z = [0.25,0.5,0.75];


for i = 1:1:3
    theta1(:,i) = (a1*(t - 2*pi*z(i)/l + 2*pi*z(1))) .* heaviside(t - 2*pi*z(i)/(l) + 2*pi*z(1))
    theta2(:,i) = (a2*(t - 2*pi*z(i)/l + 2*pi*z(1))) .* heaviside(t - 2*pi*z(i)/(l) + 2*pi*z(1))
end

figure()
plot(t,theta1)
title('Roll angle at each segment at A = 1');
xlabel('time(s)');
ylabel('Angle');
legend('z=0.25','z=0.5','z=0.75');
ylim([0,20]);
figure()
plot(t,theta2)
title('Roll angle at each segment at A = 2');
xlabel('time(s)');
ylabel('Angle');
legend('z=0.25','z=0.5','z=0.75');
```