COMP3352 Modelling and Simulation

Assignment 2

Simulating an Amphibious Robot in Webots.

Student Number: [redacted]

Submitted: 10.05.21

PLAYLIST: https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

Introduction

This project aims to simulate an amphibious robot in the Webots simulator by modelling the gait of a Salamander. Once we have implemented the solution, we will perform various investigations on the model's parameters to control speed, perform direction adjustments, and integrate obstacle avoidance.

A series of investigation and testing videos accompanies this report. Please access the playlist at the following link:

https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

Videos are referenced in their relevant section within the report.

Contents

PLAYLIST: https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

1 Simulating a Salamander in Webots

1.1 Literature Review

Crespi *et al.* (2013) present Salamandra Robotica II, a salamander-like amphibious robot capable of both anguilliform swimming and walking. Alongside offering the mechanical design, which, as shown in Figure 1, is composed of four legs and an actuated spine, Crespi et al. investigate how various gait parameters, such as body-limb coordination, undulation type and frequency, affect speed and direction. The model for control presented by Crespi *et al.* differs from our implementation, which will follow a more straightforward approach. Finally, the paper provides data retrieved from gait analysis of real salamanders, which we can use in our model for verification.

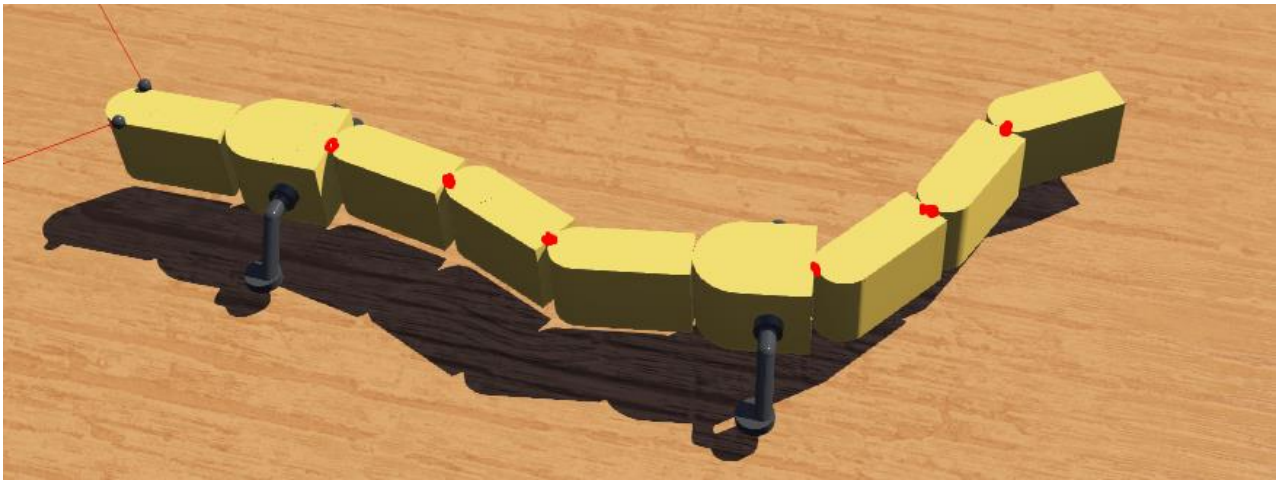

Figure 1. The Salamandra Robotica II as modelled in Webots (Cyberbotics, 2021).

When attempting to model the movement of a salamander, we can draw from the work of Crespi *et al.* and compare it with biological observations. Looking to Figure 2, the movement of the salamander is highly rhythmic. There are clear patterns to the undulation of the body. We hypothesise that manipulating this undulation will vary the salamander's speed and direction.

Figure 2. An animated image of a salamander's locomotion. Adapted from Science News (2016).

1.2 Methodology

A Central Pattern Generators (CPGs) are a biological concept representing the neural networks tasked with generating the rhythmic movement of animals, such as walking, swallowing, or swimming. Our implementation of a coupled, non-linear oscillator CPG will act as a bridge connecting the biological movement of the salamander with the actuation mechanisms of the robot (Wang *et al.*, 2017).

1.2.1 Simulation Environment:

The Webots simulator includes a pre-built simulation of the Salamandra Robotica II within its demonstration files. The Salamandra Robotica II is available as a proto in Webots, providing a blank implementation to apply our model. Looking back to Figure 1, we can see how the Salamandra Robotica II is split into seven segments, each linked by a rotational motor.
Webots offer fluid dynamics built into the system, offering controls of both density and directional water flow. Interaction between objects and fluid are also handled automatically by the Physics plugin. Webots models buoyancy through the Archimedes principle (Cyberbotics, 2021).

1.2.2 Capturing and Processing data

Please refer to video *1 – Capturing and Processing Data* for an explanation and practical example.

For data capture, three functions are present directly within the C implementation of the controller. First, writeOctaveHead is called on the initialisation of the controller. This function writes the syntax to open an array in an octave file, as shown in Figure 3.

```
// Open an array, write to passed file.
void writeOctaveHead(FILE* file) {

        fprintf(file,"A = [\n");

}
```

Figure 3. The writeOctaveHead function.

The writeDataToOctave file takes three parameters: the file to be written to and two variables. These are written to the file separated by a space, as shown in Figure 4. The writeDataToOctave is called in the main loop of the simulation.

```
// write two variables to an octave file, a space seperates their values and there is a new line for each set.
void writeDataToOctave(FILE* file,double time, double speed) {
        fprintf(file,"%f %f \n", time, speed);

}
```

Figure 4. The writeDataToOctave function.

Finally, the writeOctaveTail function will close the array in a given file, categorise the data by index, and plot the data as shown in Figure 5.

```
// Close the array on the given file. Group data by index and plot against one another.
void writeOctaveTail(FILE* file) {
          fprintf(file,"];\n");
          fprintf(file,"t = A(:,1);\n");
          fprintf(file,"speed = A(:,2);\n");
          fprintf(file,"plot(t,speed);\n");
}
```

Figure 5. The writeOctaveTail function.

1.2.3 Modelling a Salamander's Movement

Figure 6. Annotated curvature of a salamander's spine whilst swimming. Adapted from AlbyBatty TECHannel (2019).

A Central Pattern Generator will drive the movement of the salamander. Following the implementation of Price (2021), we will implement a neural circuit to produce a rhythmic motor pattern, modelling a wave across the breadth of the salamander from a single Hopf oscillator. Looking to Figure 6, the highly rhythmic undulations of the salamander map closely to the function sin(Θ).

Figure 7 shows Price's (2021) model for control of the robot's actuators. At the head (segment 0), we have a Hopf oscillator (see 1.2.5), the output of which is fed into a series of 5 phase shift neurons whose degree of shift can be set as a parameter of the robot's controller.

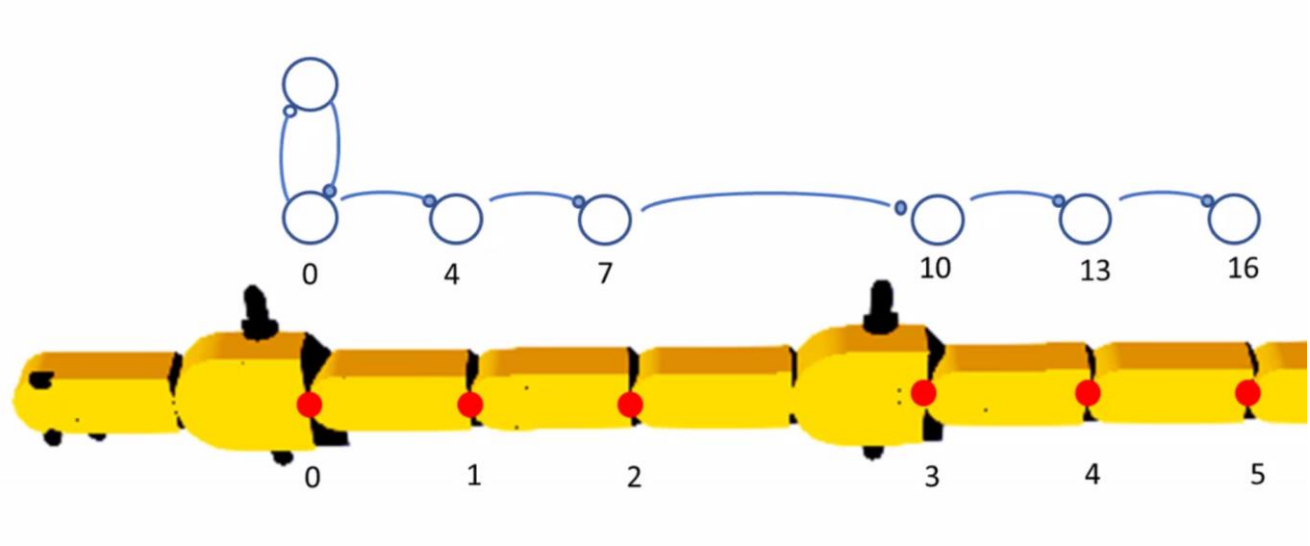

Figure 7. Top: The neural network behind Price's (2021) model for salamander locomotion. Bottom: The Salamandra Robotica II in Webots.

1.2.4 How does this compare to Crespi *et al.*'s implementation?

The work of Crespi *et al.* more closely resembles the biological network of the salamander. Here, an oscillator is proposed for each motor. However, it is unnecessary to have this level of complexity in our solution. Our implementation does not require the individual oscillators, which are only particularly useful if the robot is segmented further. Our solitary oscillator achieves a similar level of control with significantly reduced complexity.

1.2.5 Implementing the Hopf Oscillator.

Hopf oscillators, insensitive to their environment, have stable limit cycles of value $\sqrt{\mu}$. In addition to this, the frequency and amplitudes of the oscillator can be manipulated independently (Wang *et al.*, 2017).
For the initial investigation, the phase shift will be set to 60, produce one wavelength (6*60) across the whole body.

1.2.5.1 The Theoretical Model Behind the Hopf Oscillator.

Looking to figure 8, a single Hopf oscillator is composed of the linear coupling of two neurons. Neuron u_1 excites u_2, whilst u_2 inhibits u_1.

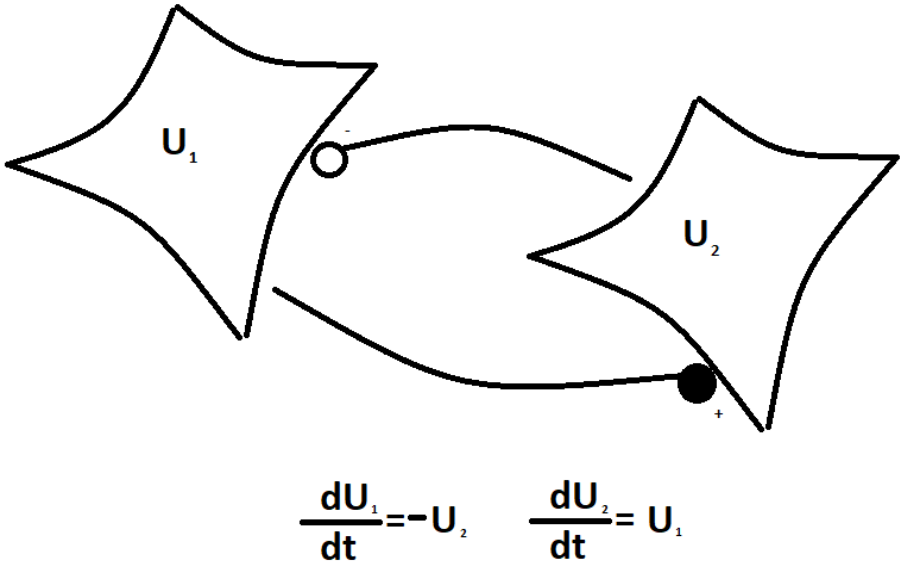

Figure 8. Two linearly coupled neurons.

The oscillator can be formulated as a system of ordinary differential equations, shown in figure 9.

$$\frac{du_1}{dt} = (M - r^2)u_1 - wu_2$$

$$r^2 = u_1^2 + u_2^2$$

$$\frac{du_2}{dt} = (M - r^2)u_2 + wu_1$$

Figure 9. The system of Ordinary Differential Equations (ODEs) behind Hopf oscillators.

The strength of u_2 depends on the strength of u_1. For any arbitrary initial state, the oscillation will converge to a limit cycle defined as $\mu = r^2 = u_1^2 + u_2^2$. Therefore, the Hopf oscillator always produces a stable, rhythmic oscillation (Wang *et al.*, 2017).

1.2.5.2 Implementation of the Hopf Oscillator in Webots.

Figure 10 shows the implementation of the ODEs shown in Figure 9. This produces an oscillation with a limit cycle of a given parameter, $\sqrt{\mu}$.

```
// ------------ calculate r   ----------------------------
r = sqrt(u[0]*u[0] + u[1]*u[1]);

// Oscillator driving motor 0
// canculate the first order derivatives of the output
// signal of each neuron, linearly relative to one another
dudt[0] = alpha*(mu-r*r)*u[0] - omega*u[1];
dudt[1] = omega*u[0]+alpha*(mu-r*r)*u[1];
```

Figure 10. An implementation of a Hopf oscillator, coded in C.

2 Modelling Swimming Behaviour

2.1 Methodology

To implement the swimming behaviour, we will be applying a phase delay to the output of the Hopf oscillator to produce rhythmic movement.

2.1.1 Implementing Swimming Behaviour

To implement the swimming behaviour, we first had to implement the phase delay neurons that take the output from the Hopf oscillator. Each neuron in the chain produces a 20-degree phase delay. There are two neurons between each output neuron, giving a phase delay of 60 between the six output neurons (360 total).

As shown in Figure 11, we utilised a for to calculate the phase delay. Please refer to the code comments for an explanation of this calculation.

```
// For the ith neuron.
// The state of the neuron is equal to 1/tau (the time constant)
// minus the state of the current neuron + the state of the previous neuron multiplied by a gain valuue K.
// Therefore the previous neuron drives the next neuron.
for ( i=2 ; i <= 16; i=i+1)
{
  dudt[i]=(-u[i]+k*u[i-1])/tau + _x;
}
```

Figure 11. the for loop used to calculate phase delay for each neuron.

PLAYLIST https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

Then, the r8_rfk function (Figure 12) solves the ODEs to get the final output as an array (y).

```
// (25) Call to the ODE solver rhs_Salamander_1.c
flag = r8_rkf45 ( rhs_Salamander_1, NEQN, y, dydt, &t, t_out, &relerr, abserr, flag );
```

Figure 12. The r8_rfk function.

Once we have the array of values, we must multiply by our amplitude (given as a controller argument) to get the final value set to the rotation motor (Figure 13).

```
//(29) Mapping from the neuron array y[...] to the salamander segment motors
 seg[0] = ampl*y[0];
 seg[1] = ampl*y[4];
 seg[2] = ampl*y[7];
 seg[3] = ampl*y[10];
 seg[4] = ampl*y[13];
 seg[5] = ampl*y[16];
```

Figure 13. Functionality to map the output neurons to salamander segments.

These final values are then passed into a clamp function, which ensures they remain within range before being set as the value of their respective motors (Figure 14).

```
//(30) Drive the Salamander motors
for (int i = 0; i < 6; i++) {
  seg[i] = clamp(seg[i], min_motor_angle[i], max_motor_angle[i]);
  wb_motor_set_position(motor[i], seg[i]);
}
```

Figure 14. FOr loop to limit rotation angle and set the position of each motor.

2.2 Verification of the Model

To Verify our model, we compared our output from the Hopf oscillator and third motor to data supplied by Price, the model's creator. If we achieved accuracy within 10% of a result known to be correct, we could verify our implementation of the model.

Figure 15 shows our output of the Hopf oscillator in blue, along with Price's results in red. There is minimal difference between the two outputs. Thus, we can verify our implementation of the Hopf Oscillator.

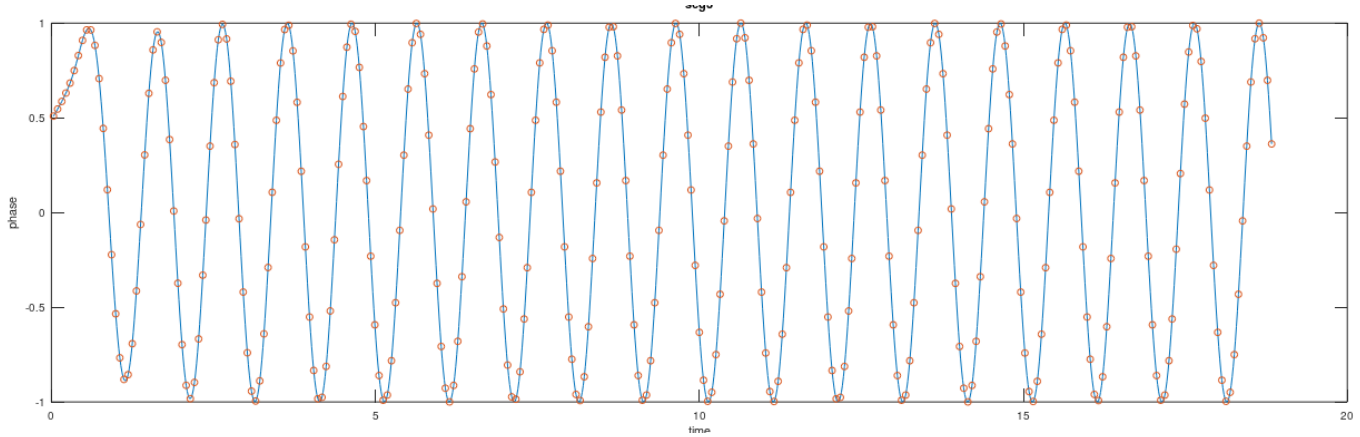

Figure 15. Data from Price superimposed onto our output for the Hopf Oscillator.

Equally important is the verification of our neural chain providing phase delay. If this were to be unsuitable for the purpose, it would throw off the patterns of the salamander's undulation, leading to inconsistent results. To verify our phase delay model, we compared the output of the third segment with data from the same segment of Price's correct solution (Figure 16).

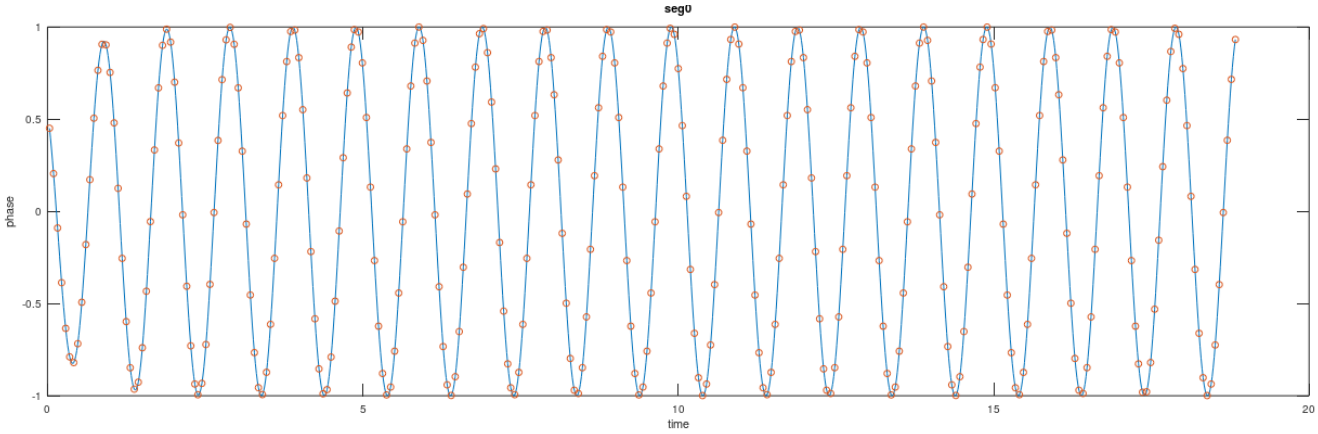

Figure 16. Data from price superimposed onto the output of the third segment.

There were no (distinguishable) discrepancies between our data sets. As such, we can confirm that the model behind the swimming pattern is verifiable. With this information, we can move on to performing investigations into the parameters of the model.

2.3 Investigating Gait Parameters

For Gait parameter investigations, the dependent variable will be speed. This metric has been chosen as it gives a good indication of two properties. Firstly, it will provide a good approximation of how an independent variable affects the velocity of the salamander whilst swimming. Secondly, by taking the variance (a measure of distribution spread from the mean) of speed, we can make an approximation of how 'smooth' the velocity has remained.

2.3.1 Phase Delay

Please refer to video *2 – Phase Race* for a recording of this investigation and an explanation of my initial observations.

Phase Lag Value	Mean Speed	Variance	Analysis of Graph

PLAYLIST https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

5	1.1851	0.17987	Graph with extremely wild results. Seems to be the result of the head swinging rather than forward momentum.
15	0.8915	0.077141	As above.
30	0.5107	0.017708	Faster that higher delays going by the mean value. Speed comes in bursts.
45	0.3592	0.0061891	Far lower range than smaller phase delays, indicative of a smoother swimming motions. Potentially less flailing.
60	0.3194	0.0038681	Whilst the graph does have significant troughs. The vast majority of the speed readings were around the mean, suggesting a more continuous velocity.
90	0.3027	0.012475	The slowest mean velocity, but with no significant peaks or troughs out of what is to be expected. This is because there was no wild flailing, This phase delay made it appear as though the amplitude of the wave had been significantly reduced, but this is just a result of the actuation.

2.4 Evaluation

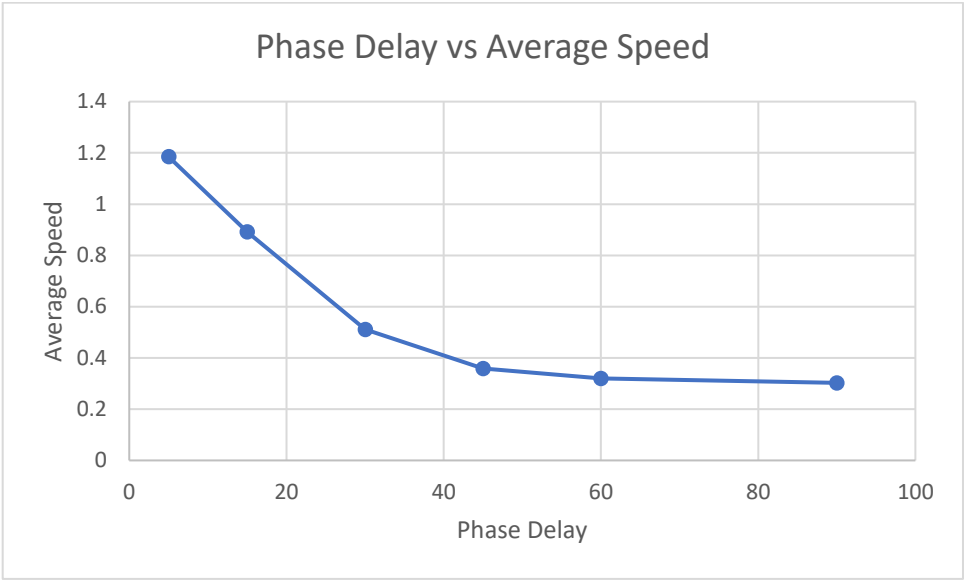

Figure 17. Phase Delay vs Speed Graph.

Figure 17 plots the average speed of each salamander vs its phase delay value. In Video 2, the clear winner was the salamander of phase delay = 45, closely followed by phase delay = 60. The graph indicates that the fastest salamander (and therefore the assumed winner of the race) is the salamander of phase delay = 5. This was not the case. Thus, speed as a dependent variable cannot be taken as an indication of forward velocity. However, we can perform a variance calculation to determine how 'smooth' the motion of each salamander is.

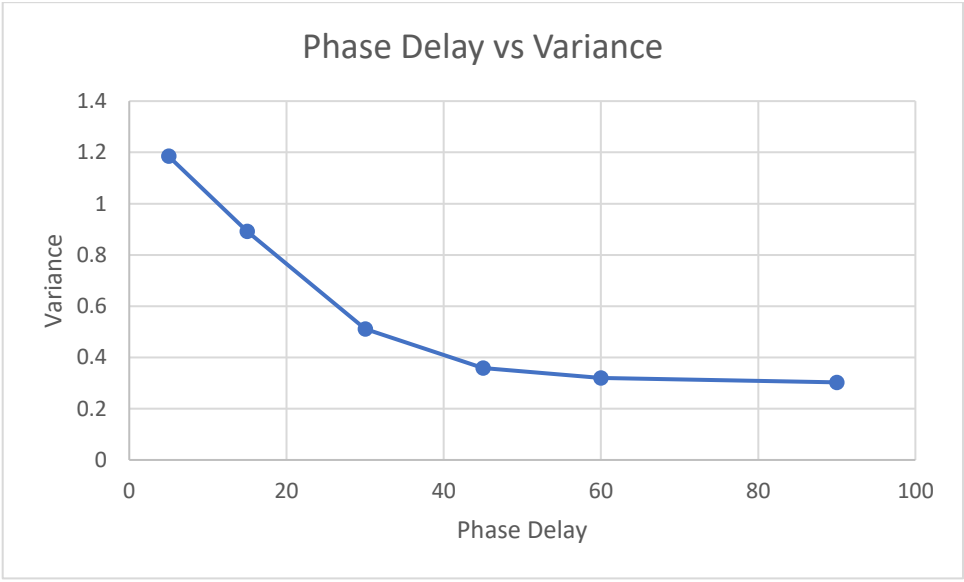

Figure 18. Phase Delay vs Variance Graph

Looking to Figure 18, variance increases as phase delay moves away from 60 degrees. This is particularly problematic as we look to implement the walking behaviour, as contact to the ground is not continuous for each leg. As a result, inconsistency of speed can disproportionately affect the direction. Figure X shows the graph of the salamander's speed vs time, where phase delay was set to 15. With a range of values between 0.2 and 1.3, it is safe to assume that this speed value is not representative of velocity through the water. Instead, the result has been skewed by wild threshing caused by the significant changes in motor positions caused by the minimal delay. If we

compare this graph with Video 2, the salamander is moving very inefficiently forward, so we can conclude that this phase delay is far from optimal.

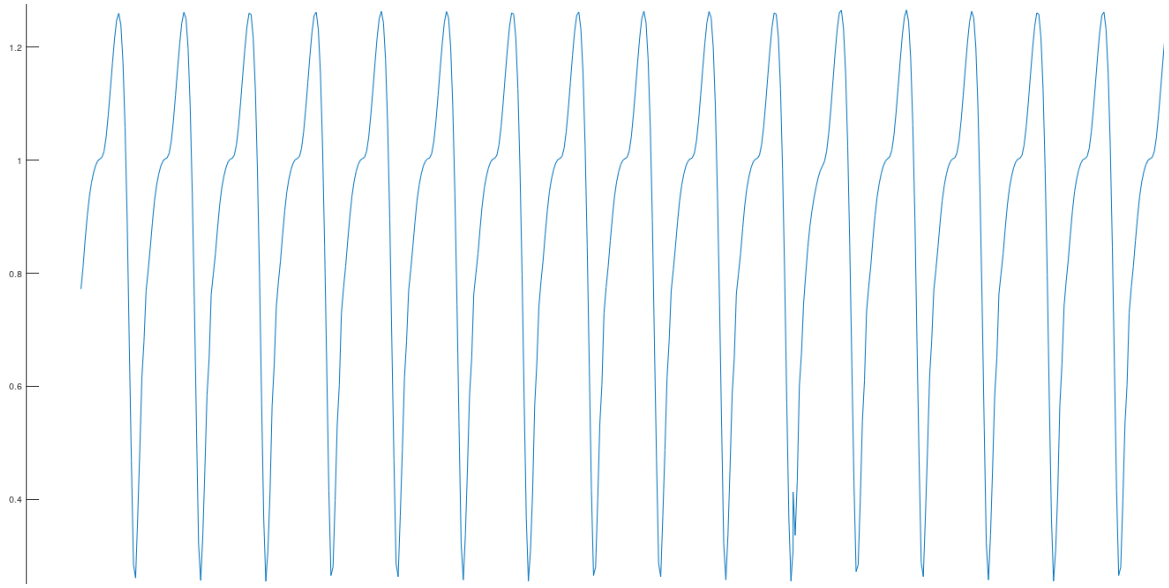

Figure 19. Speed vs time of salamander with phase delay = 15

In contrast, Figure 18 shows minimal variance at phase delay = 60 degrees. This is the ideal phase delay for our solution, as we are not particularly interested in how fast the salamander moves. Instead, we are looking for consistency of speed, which due to the nature of the salamander's locomotion, is closely coupled to directional accuracy.

2.4.1 Amplitude

Please refer to video *3 – Amplitude Race* and video *4 – Amplitude… The Decider* for a recording of this investigation and an explanation of my initial observations.

In this investigation, we change the amplitude as an independent variable and measure its impact on the dependent variable, speed.

Amplitude Value	Mean Speed	Variance	Analysis of Graph
0.25	0.10139	0.00056111	Minimal apparent variance, suggests a very minute but consistent movement.
0.5	0.20728	0.00084431	Minimal variance and almost twice the speed of the previous example.
0.75	0.31967	0.00034834	the lowest variance yet, and a significant speed improvement.
1	0.31977	0.00032596	Almost 10 times the variance and a minimal increase in speed.

1.25	0.33106	0.0046949	A similar variance and a significant speed increase.
1.5	0.33534	0.011308	Significant variance and no noticeable speed increase.

2.4.1.1 Evaluation

Crespi et al. note the diminishment of the efficacy of a turning offset where the amplitude in increased. This inverse proportionality is damaging to the model. SO whilst 1.25 clearly is an improvement, as displayed in video 4, it is not the best choice for our simulation, as It would have a detrimental impact on our steering ability. As such, we will stick to the default value of one while in water, as the steering is efficient due to continuous contact with the means of turning. When on land, however, we will seek to reduce the amplitude to 0.6. This is consistent with both the example salamander project supplied by Webots and Crespi et al.'s conclusion that limiting the amplitude will improve steering ability.

3 Modelling Walking Behaviour

3.1 Methodology

Please see video *5 – Investigating Walking Speed* for a recording of the investigation and an explanation of the methodology for leg control.

Ideally, the Hopf oscillator would control the arms as well as the legs. Initially, we attempted to normalise the output of the Hopf oscillator to drive the legs. However, we ended up with legs that would oscillate backwards and forwards instead of producing a continuous forward rotation. After a great degree of trial and error, we determined that an alternative approach would be needed.

The model behind the rotation of the leg motors is driven by the linear advancement of time. In Figure 20, we multiply the current time by a speed constant. As time continually advances within the simulation, so will the leg motors' rotational position. Each leg is out of phase with its opposite by pi/2. Therefore, whilst one leg is at the top of its rotation, the opposite leg is currently on the ground, producing the alternating movement required for effective locomotion.

```
seg[6] = -speedW*t_out + M_PI;
seg[7] = -speedW*t_out;
seg[8] = -speedW*t_out + M_PI;
seg[9] = -speedW*t_out;
```

Figure 20. the implementation of continual arm movement.

3.1.1 Investigating Speed

The investigation and further explanation of results are included in video *5 – Investigating Walking Speed.*

Walking Speed Constant	Average Speed Achieved
5	0.19330
10	0.29175
15	0.23017

3.1.1.1 Evaluation

The salamander must have sufficient contact time with the ground to thrust its body forwards. But there is certainly a balance between ensuring clean contact and moving the rotational motors too slowly, as evidenced by Figure 21. Therefore, we believe ten is a suitable walking speed constant.

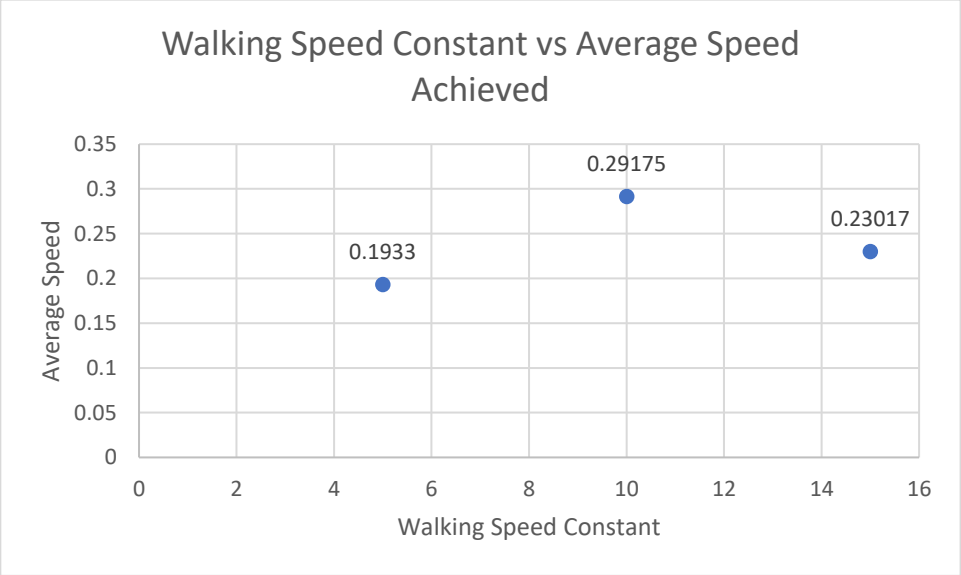

Figure 21. A graph of the walking speed constant vs the resultant average speed

4 Implementing Obstacle Avoidance

PLAYLIST https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

4.1 Literature Review

Crespi *et al.* note that an offset is used to translate the outputted curve about the Y-axis. The offset results in a bias towards one side of the salamander during undulation, allowing the salamander to change direction.

4.2 Investigating the Offset

Please see video *6 – Investigating the Offset* for a recording of the investigation and a practical example of how an offset impacts salamander direction.

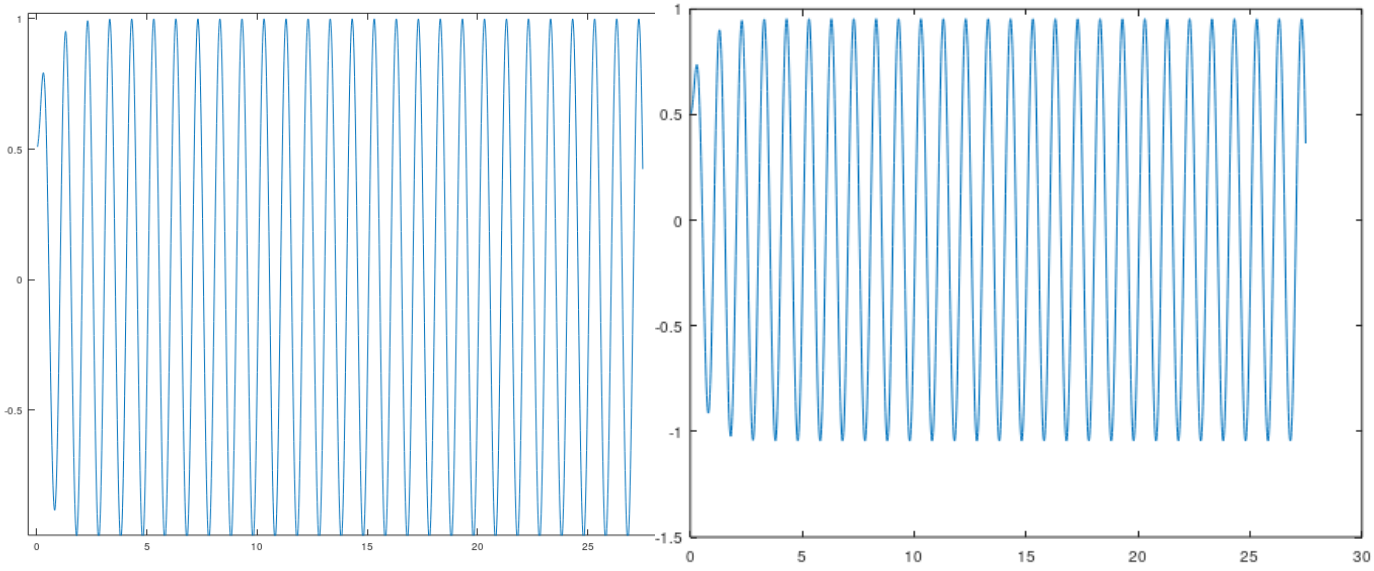

Figure 22. Shows the difference in the output of the same motor where an offset has been applied. Left: Offset = 0. Right: Offset = - 0.2

Although the offset appears marginal, with only a slight difference (-0.2) between the graphs in Figure 22, video 6 proves that it has a highly significant impact. As the distance sensor tops out at 0.7, the maximum difference between the sensors is at most 0.7-0=0.7. Therefore, we don't need to scale the offset value as its maximum is small enough not to break the code but large enough to move the salamander away in good time.

4.3 Implementing the Obstacle avoidance

Please see video 7 – *Implementing Obstacle Avoidance* for an explanation as to how obstacle avoidance has been implemented.

PLAYLIST https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA

To summarise video 7, the offset is applied directly to the calculation of neuron values as a constant _x (Figure 23).

```
for ( i=2 ; i <= 16; i=i+1)
{
  dudt[i]=(-u[i]+k*u[i-1])/tau + _x;
}
```

Figure 23. the offset is applied to neuron values.

The actual calculation of the offset value is achieved by taking the difference of two distance sensors mounted on either side of the salamander. The difference between the two is a signed value, used to offset the spine directly and move the salamander away from the object.

```
void setHeading()
{

  /* perform sensor measurment */
  double left_val = wb_distance_sensor_get_value(eyeL_tag);
  double right_val = wb_distance_sensor_get_value(eyeR_tag);

  /* change direction according to sensor reading */
  offset = (right_val - left_val);

  // call setdirection in RHS (apply offset)
  setDirection(offset);
```

Figure 24. Calculation of the offset by finding the difference between the distance sensors.

4.4 Limitations of the obstacle avoidance implementation

4.4.1 Equilibrium of Offset in Corners

As demonstrated in video *8 - Obstacle avoidance and Hitting an Equillibrium,* the salamander can become stuck in corners due to the values of offset of each side levelling out. To counteract this, we increased the number of rays per distance sensor to 5, essentially reducing the weighting of each rating fivefold. As demonstrated in video *9 - Testing Corners and Beaching with the 5 Sensor Salamander* this prevented the salamander from becoming caught in the corner.

4.4.2 Issues with steep Inclines/Declines

Video *10 - Decline Issue* shows that the salamander can be thrown off course when navigating declined planes. When the salamander lits its body off the ground whilst walking, it rolls slightly in one direction, causing the

near-side distance sensor to detect the floor as an obstacle. When the 'obstacle' is detected, the salamander moves in the opposite direction. As demonstrated, this can have a significant impact in tight spaces.

5 Implementing the Transition Between Water and Land

5.1 Methodology

Video *11 - Explaining the Logic Behind the Transition Model* explains the logic behind transitioning between land and sea, based on GPS coordinates. To summarise the video, the salamander must go slightly below the water before swimming (Figure 25); else, it will not be a smooth transition to swimming. Furthermore, where the salamander approaches the beach, it must move slightly above water level before it starts walking, else it will struggle to deal with the suddenly inclined plane.

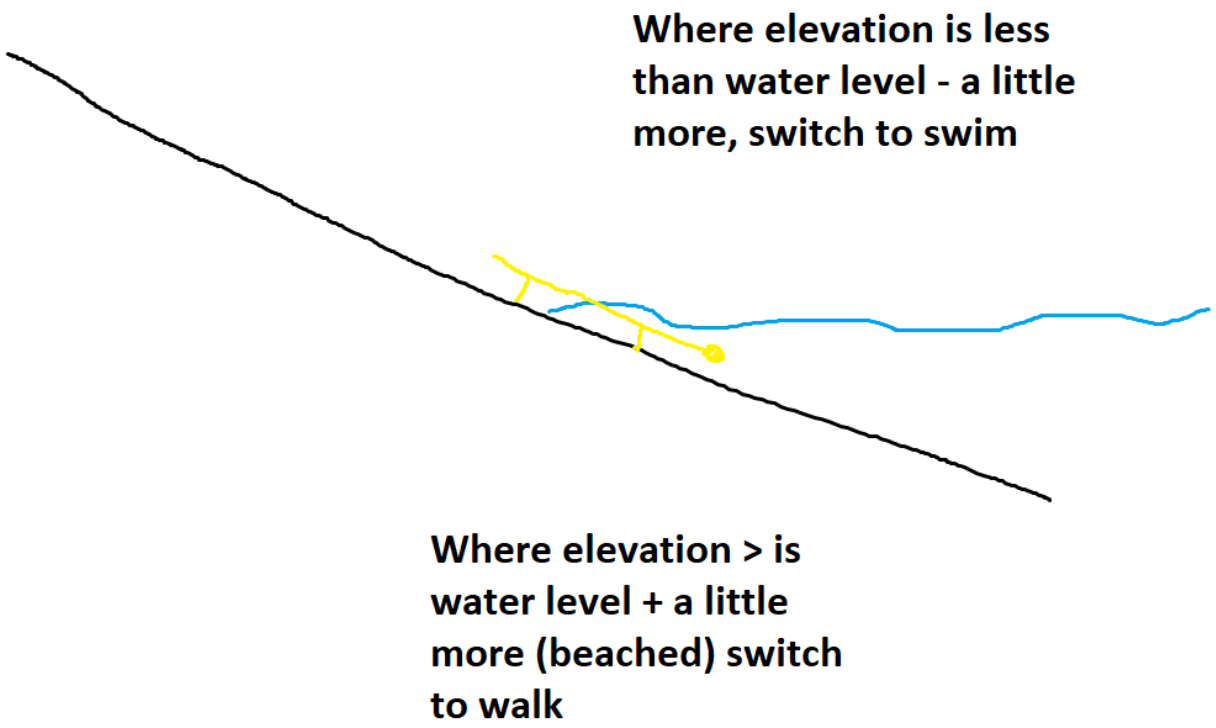

Figure 25. The salamander's transition logic between states.

5.2 Implementing the Model for Transition

The final salamander implementation has two states, walking or swimming. If the salamander is walking and it drops below sea level (plus a little more), it will switch to the swimming state. If the salamander is currently swimming and it beaches itself above sea level, it will begin walking. This logic (As shown in Figure 26) is called in the main loop of the program.

The amplitude of undulations is adjusted depending on whether the salamander is currently at land or sea. Therefore, the offset can have a significant enough impact to turn the salamander whilst walking.

```
// if elevation is greater than the water level
if (locomotion == SWIM && elevation > WATER_LEVEL + 0.04) {

  locomotion = WALK;
  // Reduce amplitude
  ampl = 0.6;
}

else if (locomotion == WALK && elevation <= WATER_LEVEL - 0.015) {

   locomotion = SWIM;
   // Set amplitude to default
   ampl = 1;
```

Figure 26. The transition logic implemented in C

5.3 Demonstration of the Final Salamander Along With a Qualitative Analysis of the Transitional Behaviour

The final video, *12 - A Qualitative Investigation of the Land to Sea Transition*, demonstrates the simulation's final implementation and all behaviours.

6 Additional Qualitative Investigations

Included at the end of the video playlist are a few qualitative investigations, along with an explanation of a few comprises I had to make to have the model work as effectively as possible. These are not essential to the report but might provide insight into the development process.

7 References

AlbyBatty TECHannel. (2019) *Salamander swimming in the pool - Close Up View* Available at: https://www.youtube.com/watch?v=ixhpETPWqEs&ab_channel=AlbyBattyTECHannel (Accessed 3rd May 2021)

Crespi, A. et al. (2013) Salamandra Robotica II: An amphibious robot to study salamander-like swimming and walking gaits. *IEEE Transactions on Robotics*. [Online] 29 (2). Available from: doi:10.1109/TRO.2012.2234311.

Cyberbotics (2021) *Fluid* Available at: https://www.cyberbotics.com/doc/reference/fluid (Accessed 3rd May 2021)

Science News. (2016) *Watch a salamander walk on a treadmill | Science News* Available at: https://www.youtube.com/watch?v=tEb8JMOxjSs&ab_channel=ScienceNews (Accessed 3rd May 2021)

Wang, G., Chen, X. and Han, S. (2017) *Central pattern generator and feedforward neural network-based selfadaptive gait control for a crab-like robot locomoting on complex terrain under two reflex mechanisms* International Journal of

COMP3352 Modelling and Simulation
Assignment 2 [redacted]

Advanced Robotic Systems July-August 2017: 1–13. Available at:
https://journals.sagepub.com/doi/pdf/10.1177/1729881417723440

PLAYLIST https://youtube.com/playlist?list=PLkXUSADeSrqlhSO0vXVwEm0_X2q49zrKA