Automated Optimised Segmentation of Swimming Fish Midlines

MPhil Candidate - Samuel E. A. W. Fetherstonhaugh <sef7@aber.ac.uk>

Project Supervisor - Otar Akanyeti <ota1@aber.ac.uk>

Abstract

While many fish propel themselves using their continuous flexible bodies, fish robots and biological models are often constructed from interconnected discrete segments. Considering this, how many segments are required for a robot or model to represent fish movements accurately? To bridge the gap between biology and engineering, two new methods are presented here, that automatically determine accurate and concise segmented fish models from actual fish data. These two methods are: *segment growing approach*, a greedy algorithm that sequentially 'grows' segments across the fish midline, and *evolutionary algorithm approach*, which progressively 'evolves' different combinations of segment lengths. These methods identify key bending points along the fish body, linking them with rigid segments, so that the difference between fish and modelled midlines is minimised. To verify the utility of these methods, they were tested with the kinematics of ten species during steady swimming, and rainbow trout over four swimming behaviours, and multiple swimming speeds. Broadly categorised as (sub)carangiform swimmers, these fishes exhibit diverse morphology, along with various swimming patterns. From these tests, several trends in results are found:

1) Regarding segment numbers, five segments are sufficient for modelling the kinematics of all fishes with at least 99% accuracy (midline-segment difference < 0.01 body lengths).

2) Segment lengths get progressively shorter towards the tail, for multi-segment models with best performance.

3) Between different species, there is notable variation in locations of segment joints along the body, particularly in the anterior region.

4) Between different swimming behaviours, there are notable variations in segment numbers, along with variation in anterior joint locations.

5) In trout, swimming speed does not affect joint locations and segment numbers.

The findings presented here provide a mechanistic understanding of relationships between kinematics and different fish attributes. These two methods and the models they generate could be applied in several open research areas, including construction of robot fish, investigation of fish muscle activity, and frameworks for lifeforms in virtual media.

Table of Contents

1: Introduction

The swimming capabilities of fishes have interested scientists for decades. Despite the wide variety in fish species, nearly all of these are able to achieve efficient propulsion using their body movements. Some fish species primarily use waves of body undulations for propulsion [1], while others rely more on oscillation of fins [2], and many employ both methods. There is also great variation in the nature of body undulations used for propulsion; some cover the entire body, while others are more focused around the tail, and there are several kinematic parameters such as body wave amplitude, wavelength and frequency that vary with size and speed. Yet for nearly all fish species, these body movements lead to propulsion that is both effective in some regard to the tasks they encounter whilst still being energy efficient enough to survive. How has evolution come to develop such diversity in fish propulsion mechanisms? What forces and factors have led to such wide variety in some cases, and why have they led to little variance in others?

Viewpoints from Robotics and Biology

Some of these questions regarding fish locomotion are of significant interest to engineers and roboticists. Understanding the forces and mechanisms behind fish locomotion can aid constructing better biomimetic robots and bioinspired technologies. For aquatic vehicles such as boats and submarines, a bioinspired approach to locomotion may be preferable over the qualities of a traditional propeller. A bioinspired approach may be less noisy or more energy efficient. Similarly, biomimetic robots may be more suitable for certain tasks, such as investigating

pipes, as in addition to potentially being more energy efficient, they may be more slender, flexible and hence manoeuvrable. Also, biomimetic robots may be better for observing, analysing and monitoring coral reefs, as they are less likely to disturb surrounding wildlife, allowing them to witness more natural behaviour.

Biologists are also interested in understanding fish locomotion for a number of reasons. By understanding the effects and forces experienced by different locomotion systems, biologists can better understand how species have adapted to their environment, and construct evolutionary arguments for why those systems evolved in the first place. Also, understanding how fish locomotion functions could provide valuable insights into other areas such as biomechanics, hydrodynamics, and neural mechanisms. Locomotion is also a hallmark of many other biological processes, including migration, prey capture, predator evasion, mating, and navigating various environments such as different flow conditions. Understanding fish locomotion could contribute to understanding these and other behaviours important to survival.

Interdisciplinary Crossover and Gaps

Additionally, there is crossover between these areas, where roboticists can aid biologists and vice versa. Where experiments with live specimens are impractical, robot fish can provide an alternative platform to test biological hypotheses, with greater control and higher precision. Although it may be difficult to persuade a live specimen to perform certain patterns of behaviour, such as overly strenuous behaviour, repeating behaviour an unlimited number of times, or behaviour that is only exhibited under certain circumstances, a robot can bypass this need for

persuasion. Likewise, biological understanding is necessary for guiding biomimetic design. An understanding of fish shape and body stiffness is desirable in order to replicate it in a robot, and may be necessary to accurately replicate body motions. Additionally, achieving realistic body movements also requires analysis of body movements in real fish. However, gaps exist between these areas which have yet to be bridged. For instance, regarding speed, efficiency and manoeuvrability, most artificial fish are still less effective than their biological counterparts. Consequently, they still have lower technology readiness levels compared with traditional propeller driven aquatic robots [3] [4].

Many of these gaps result from the significant differences between biological bodies and mechanical structures. Biology, and hence real fish, are made from organic materials, such as flesh, bone, muscle, tendons, and ligaments. Additionally, they are capable of being highly adaptable; they can adjust attributes such as body stiffness [5] and fin shape [6] in real time to increase swimming efficiency. In contrast, engineering and robotics are concerned primarily with mechanical and artificial structures, such as metal and plastic, rigid joints, and motors. Thus, where exact replication of biology is not possible, compromises are often required. While not exactly replicating muscles, traditional mechanical motors are commonly used in robotic fish. However, even with current robotics there is still a trade-off between body flexibility and degrees of freedom, with many robots being fixed in the sense that they cannot adjust their body shape or flexibility.

That said, although these limitations restrict real time adaptation, there is still a range of possible implementations of fish robots. Despite the rigidity often inherent to

mechanical engineering, some robots are quite flexible, being made from compliant materials such as silicone [7]. These robots are often predicted to be manoeuvrable, energy efficient, and quiet. However, they are also often under-actuated (employing only one or two actuators), being difficult to control and limited in what behaviours they can exhibit. On the other hand, some robots are quite rigid. As continuous flexible bodies can be difficult to produce, a common construction method is using series of rigid segments, each being controlled by a motor [8]. However, this design can also provide high degrees of freedom, allowing more elaborate control and behaviours, such as multiple swimming maneuverers and turning abilities. The downsides of these robots are that each motor adds more noise and energy consumption to the overall design.

Improving Segmentation

But with regard to the idea of using a segmented body, how would one decide how many segments to use, the relative length of each segment, and are these questions worth considering? The overall financial cost of a robot, and energy required to operate it, are likely to increase with the number of motors used, which in turn are likely to increase with the segment number used, if the joints controlling their movement are active (i.e. motorised) rather than passive (e.g. elastic, free moving). Although in some cases it may be possible (or even necessary) to use non-active, passive or elastic joints, how does one decide where these should be used? Additionally, are joint locations relevant regardless of whether joints are active or passive? While at first it may be simplest to use equal length segments, if some of these joints split areas which would not normally require much bending, they could be a waste of resources or overly complicating the design. Furthermore, given the

existence of these issues with segment numbers and lengths, it may imply that, despite current assumptions, other issues could exist with selecting values for these properties.

The research presented here aims to investigate the potential of developing an automated approach to determining segment numbers and lengths that increase accuracy and reduce complexity of multi-segment fish models. Two methods are proposed: a segment growing approach, and an evolutionary algorithm approach. The former method starts with a target accuracy that the model should achieve (specified by the user), and searches for the minimum number of segments required to achieve this (a most 'parsimonious' model). It does this through sequentially 'growing' segment lengths until their error reaches an error threshold. The latter method starts with a specific fixed number of segments that the model should use (specified by the user), and searches for the best lengths for these segments to maximise accuracy. It does this using a nature inspired method (an 'evolutionary algorithm'), through iteratively 'evolving' many possible segment length combinations by selecting, merging, and altering the more accurate segments over many generations. Both methods are applied to a range of fish datasets, varying in species, swimming speed, and behaviour. This may provide valuable insights into how much automatic segment optimisation could improve accuracy and complexity, patterns in best segment lengths and numbers, and their variations with different fishes and kinematics. In turn, these could aid in guiding future robot design, further understanding of fish biology, and categorising fish kinematic features. The principles behind these two algorithms, how they function, the experiments they were

tested with, and their results and implications, are discussed in detail throughout this thesis.

1.1: Background and State of the Art

Current research in fish robotics focuses on many different topics, including sensing [9] [10], actuation [11] [12] [13], soft body design [14] [15] [16], body and fin control [17] [18], and artificial skin for drag reduction [19] [20]. Below, some of the topics relevant to this thesis are discussed in greater depth.

Studying Fish Movement

Although many developments in fish robotics are fairly recent, analysis of fish locomotion using photography and film has been performed for almost a century. As far back as 1933, Sir James Gray aimed to perform some of the first quantitative studies of body movements of fish swimming, attempting to link them to forces responsible for fish propulsion. He visualised deformations of fish bodies using frames from films he recorded of fish swimming, in conjunction with an ingenious timing circuit for determining precise times between frames. This latter point was crucial for calculating the velocities of points along the fish body. His experiments and approaches laid the foundation of current fish kinematics research that uses high-speed cameras and high-resolution images (see [21] for a short review).

Exploring Robotics and Biology with Artificial Fish

Once technology had advanced sufficiently, roboticists began constructing and experimenting with robot fish. However, how closely robots mimic fish can vary considerably. Some focus more on functionality, employing only very abstract concepts inspired from fish, such as a generic torpedo shaped body and flexible tail [16]. On the other hand, some go to great lengths to construct closer replications of specific species [22]. The degree of mimicry chosen is often related to the goal of the research. In contrast to robots designed for more practical uses (e.g. robot fish employed for aquatic exploration [15] or aquarium exhibitions [23]) closer biomimicry is observed for robots used to explore biological hypothesises (e.g. investigating whether altering stiffness can increase propulsive efficiency of the posterior body [24] and caudal fin rays [25]). Additionally, continuing from this, robot inspired biology (coined by Gravish and Lauder [26]), which focuses on employing physical robots and models to increase understanding of how animals move (including fish kinematics and propulsion), is a fast-growing research area. It has recently resulted in many significant discoveries, such as the forces employed by insect flight [27], mechanisms for transitioning between swimming and walking in salamanders [8], crevice transversal using body shape changes akin to cockroaches [28], propulsive effectiveness and efficiency of median/paired fin undulations [29], and how tail beat amplitude influences performance of steady swimming and acceleration [30].

Robot Fish Actuation and Control

One crucial area that underpins much of robotic fish research is working out how to actuate these robots. Traditionally, roboticists have used motors to actuate rigid sections of their robots [31] [11]. However, in recent years some have chosen to

actuate body movements through pumping liquids or gases into chambers within a flexible body [15] [16]. In other cases, they have chosen to use electrically controlled shape memory alloys (SMAs) to control body bending [12] [32].

In addition to determining how they would like to actuate their robots, roboticists also looked into how to increase the movement capabilities of these robots. Some decided to build robot fish which used thin, semi rigid rays to control pectoral and caudal fins, as it allowed them a realistic way to achieve greater control over both movement and shape of these fins [6] [25]. Also, as the most common approach to constructing a fish robot is to use a series of interlinked rigid segments, a method is needed for generating undulatory segmented body movements for propulsion. To achieve this, many researchers choose to determine the relative movement of each segment by lining them up with a traveling wave equation [23] [33] [34].

Travelling Wave Equation

With this approach, given a location x along a straight line parallel to the direction of fish locomotion at time t, the lateral displacement y of the corresponding body location perpendicular to x is determined using a fairly straightforward mathematical equation:

$$y(x,t) = A(x) \times \sin(kx - \omega t + \phi) \quad (1)$$

where A is the amplitude of the body undulations at location x, k is the body wave number, ω is the angular frequency, and ϕ is an arbitrary initial phase of the body wave. Spatial frequency of body wave peaks can be controlled with angular

256 frequency, which is determined by $2\pi f$, where f is the frequency of body
257 undulations. Likewise, frequency of body waves over time can be controlled with the
258 body wave number, which is determined by $2\pi/\lambda$, where λ is the body undulation
259 wavelength. Amplitude is controlled with A, which can be described using a
260 polynomial function: for instance, for subcarangiform swimmers such as rainbow
261 trout and mackerel it is described as $A(x) = a_l x + a_q x^2$, where a_l and a_q are the
262 linear increase and quadratic increase components of the amplitude [35] [36].
263 Combined, all these parameters allow for a wide variety of undulatory movements.
264 That said, most robotic fish are modelled after (sub)carangiform swimmers (e.g.
265 common carp, *Cyprinus carpio* [37] or rainbow trout, *Oncorhynchus mykiss* [38]). For
266 these fish, it is currently assumed that body undulations are mostly confined towards
267 the posterior of the body, undulation amplitude grows exponentially towards the tail,
268 and undulation wavelength is approximately equal to one body length. Either way,
269 once an artificial midline is generated using this method, it can then be used to
270 support other models or robots, such as aiding with identifying key bending points
271 along the fish and hence model body.

272 There are, however, some notable limitations with this approach. Firstly, it is only
273 really suitable for modelling steady swimming or acceleration. It is not suited to
274 modelling unsteady behaviours, such as burst and coast swimming, or turning and
275 escape maneuverers such as c-start (see [39] for a summary of unsteady
276 behaviours). Also, it does not specify head movement, or the starting location of the
277 body wave, both of which vary between species [40]. It also assumes that simple
278 linear and quadratic functions are sufficient for replicating patterns of body
279 amplitudes, when these may be more complex than originally thought. There is also

little empirical evidence as to how effectively travelling wave equations approximate actual fish midlines during steady swimming [35] [41]. Furthermore, even if they do provide a moderately accurate approximation of midline movement, they do not necessarily allow roboticists to directly infer robot movement from an actual midline.

Segmented Models and Segmentation Algorithms

However, segmented models (including the approaches to generating them presented in this thesis) may have the potential to improve over the limitations of using a traveling wave equation. Particularly, segmented models can potentially replicate less regular, more varied or complex kinematics, as they are not restricted to uniform changes in body movement. Additionally, especially in the case of segmented physical robots, if a traveling wave equation is used to model fish body movement, then extra calculations need to be performed to convert the wave movement into segment movement. However, these extra calculations, and the traveling wave itself, can be bypassed if the segment movement is extrapolated directly from the actual fish midlines, as can potentially be allowed by the two segmentation approaches that are the focus of this thesis.

Several approaches to segmenting curved lines, whether those are actual curvatures or nonlinear series of points, have been proposed over the years. The Ramer-Douglas-Peucker algorithm [42] essentially performs segmentation on a line consisting of a series of interlinked points, by determining which points can be discarded without deviating excessively from the original midline, using a distance threshold. This algorithm starts by comparing a single hypothetical segment directly between the first and last points in the series, and each of the points between the

first and last points, based on the distance of each point from the hypothetical segment. If the point with the largest distance is greater than the specified threshold, then that point must not be discarded. Then, the comparison described is repeated, using a hypothetical segment between this point and the first point, and all points between. If after repeating this a point is found whose distance is less than the specified threshold, then this point can be discarded. This process is repeated, each subsequent time using a hypothetical segment between the most recent point that must not be discarded and either the last point or the next point that must not be discarded. The algorithm finally stops when, between the most recent point that must not be discarded and the last point, there are no more points that must not be discarded. If any points are discarded, this results in a line consisting of fewer segments than the initial number of links between the series of initial points.

A variation of this approach is the Visvalingam-Whyatt algorithm [43]. Like the Ramer-Douglas-Peucker algorithm, it repeatedly evaluates points for removal in a similar manner. However, instead of evaluating points using their distance from hypothetical segments, it uses whole areas between these points and their neighbours. That said, while it may be possible to apply these algorithms to the optimisation problem presented in this these (as each input midline dataset is a series of points), they may be limited by the fact that they rely on existing points, when optimal segment end locations may in fact be located at positions that are not equal to any of the points included in the initial input data.

An instance where segmentation has been applied to a similar domain as that discussed in this thesis (i.e. segmentation of fish midlines) is presented in [44]. In

this case, best locations of segment joints for a robotic fish were deduced using a Big Bang-Big Crunch (BB-BC) algorithm, which is similar to an evolutionary algorithm implementation. The BB-BC algorithm process consists of big bang phases, where potential solutions are semi-randomly generated, and big crunch phases, where the 'centre of mass' of these solutions is calculated. The effectiveness of each random solution generated in the big bang phases are calculated using a cost function. In the case of [44], the cost function is based on the segment error 'envelop' (i.e. the total area between the segment and the corresponding portion of the traveling wave equation employed to approximate the midline) of the segment lengths of the solution in question. Generally, the centre of mass in the big crunch phases is an amalgamation of all the random solutions generated in the previous big bang phase, with a bias towards the more effective solutions. Once this amalgamation has been calculated, it is used as the basis for the random solutions in the next big bang phase. This process of big bang and big crunch phases is repeated continuously, and due to the repeated amalgamation bias towards more effective solutions, is likely to progressively get closer to a best solution.

1.2: Research Questions

1) Can automatic approaches be used to find high accuracy, low complexity segment numbers and lengths for representing fish midline movement?

2) How do the parameters controlling these approaches effect the improved segment numbers and lengths they produce? How does increasing or decreasing the error

limit change the best segment number and lengths? Likewise, how does increasing or decreasing segment number effect the accuracy of the outputted model?

3) How do factors such as fish species, swimming speed, and behaviour effect the best numbers and lengths of segments required?

4) How do results of applying these automatic approaches compare with other approaches (e.g. equal length segment models)?

1.3: Novel Contributions

1) Two automatic segmentation methods have been developed: a segment growing approach, which sequentially grows each segment until they reach an error threshold, and an evolutionary algorithm approach, which progressively evolves many possible segment length combinations.

2) These methods have been applied to a variety of actual fish datasets, covering multiple species, swimming behaviours, and swimming speeds. This has resulted in many multi-segment models, along with comparisons with simpler equal length segments and model differences related to fish attributes.

1.4: Research Outputs

1) Three conference abstracts have been presented: one in The Association for the Study of Animal Behaviour (ASAB) Summer Conference in 2019 (poster presentation), and two to The Society of Integrative and Comparative Biology (SICB) conferences in 2020 and 2021 (both oral presentation).

2) A journal manuscript has been published in the journal 'Bioinspiration & Biomimetics', which consists of a significant amount of the research and findings presented in this thesis [45].

3) An educational app for touchscreen mobile devices has been developed, 'Robobalik', which uses virtual fish based on models generated with the two segmentation methods. This has been used as an outreach tool, having been presented at the 2018 Cardiff National Eisteddfod and the 2019 Royal Welsh Show.

4) Two additional journal publications are being prepared. The first focuses on analysis of a large collection of fish midline data in collaboration with international partners. The second focuses on optimising and exploring the parameters controlling the automatic segmentation methods and further investigation of their limitations.

2: Methods

This section details the problem definition, how multi-segment model accuracy is calculated, and the two automatic segmentation algorithms that are used to generate multi-segment models: the segment growing approach, and the evolutionary algorithm approach.

2.1: Problem Definition

Objective

During this research, fish body kinematics were modelled using series of linear, rigid segments, connected end to end by joints (Figure 2.1 C). Model accuracy was expected to increase positively with increasing numbers of segments, as many shorter segments can fit body curvature more closely than fewer longer segments. However, the downside is that increasing segment numbers also increases complexity. To address this trade-off, two methods are proposed here that enable determination of a parsimonious multi-segment model using a minimum number of segments. A key component of generating these accurate yet concise multi-segment models is using segments of different lengths, each fitted to the different levels of curvature that occur at different locations along the body.

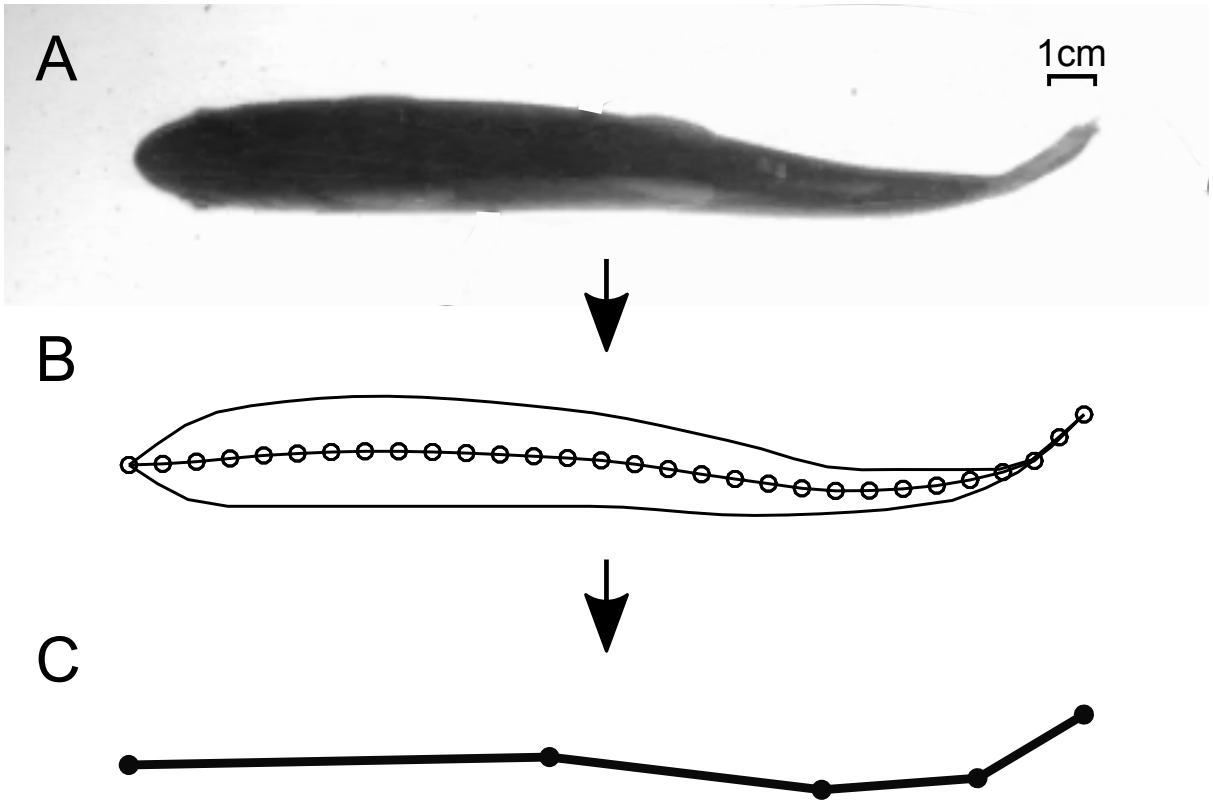


Figure 2.1: Stages of fish midline segmentation. (A) A frame from a high-speed video of a trout, filmed during swimming in a flow tank. (B) A midline and silhouette obtained through digitising the same frame, with the individual midline points highlighted in empty circles. (C) An example of a multi-segment model fitted to the same midline points.

Input Data

The input data passed to the methods consists of fish body midlines, stored as two-dimensional matrices of x y coordinates along the body that move with time. Each x y coordinate defines a specific point along the body between the nose and tail tip, with a whole midline being represented as a series of many points covering the body from head to tail. The ordered set of all points along a midline is referred to as $M = \{P_1, \ldots, P_n\}$, where n is the total number of midline points. All midline points should be equally spaced along the body. Additionally, these points only move across a two-dimensinional plane parallel to the lateral body movements of the fish,

with the assumption that the fish does not roll, pitch, or perform other three-dimensional movement. As was the case here, midline data can be obtained through recording fish swimming in a flow tank from above (Figure 2.1 A). These recordings are then digitised into point sets either automatically or manually, as illustrated in Figure 2.1 B. However, note that the research presented here is not concerned with data collection or midline digitisation, only representing these digitised midlines with multi-segment models.

Output

The two segmentation algorithms presented here represent a multi-segment model as a series of locations where joints connect segments. Specifically, a model is represented using a single row numerical array, where each number denotes the location of one joint.

In this case, a one-dimensional coordinate system is used to represent locations along midlines, as if they were laid straight. In this system, a single value ranging from 0 to 1 represents a location along the body ranging from the nose to the tail tip. Thus, a location one quarter of a body length along the body would be represented as 0.25, a location of three fifths would be 0.6, and so on. Building on this, given a model with four equal length segments, their joint locations would be represented as $[0.25, 0.5, 0.75]$.

2.2: Model Accuracy Calculation

During this research, model accuracy was determined through calculating the difference between model segments generated by the segmentation algorithms, and the actual midlines obtained from fish. This difference is more often referred to as 'error', and is measured in body lengths (*L*). Error is translated into 'accuracy' as:

$$accuracy = (L - error) \times 100 \quad (2)$$

Thus, accuracy is a percentage of the difference between error and one body length. For instance, given an error of 0.25 *L*, the accuracy would be 0.75, or 75%. Consequently, by minimising error, accuracy is maximised.

Model error is split into two components: specific segment error (SSE), and overall model error (OME). The former is the accuracy of individual segments compared to their respective portions of the actual midline, so that they can be evaluated individually. The latter referrers to the accuracy of an entire model, by collating individual segment errors into one number representing the overall model performance. This allows whole models to be evaluated and compared using a single measurement value.

2.2.1: Specific Segment Error

Specific segment error is defined as the maximum time-averaged distance between a segment and a midline point, out of all midline points within the corresponding midline subsection for that segment. This provides a concise measure of how closely the rigid segment fits the curvature of the midline.

Calculating the Difference Between a Segment and a Midline Point

The difference between a segment and any midline point (referred to hereafter as 'midline point error', or 'MPE') is simply the distance between the two. For each individual time frame, $t = \{1,2,3,\ldots,T\}$, of a midline data set, the distance between a segment S_i and any individual midline point $P_j = (x_j, y_j)$ within the corresponding midline subsection M_i is calculated using Equation 3:

$$Distance\ (S_{is}, S_{ie}, P_j) = \frac{|(y_{ie} - y_{is})x_j - (x_{ie} - x_{is})y_j + x_{ie}y_{is} - y_{ie}x_{is}|}{\sqrt{(y_{ie} - y_{is})^2 + (x_{ie} - x_{is})^2}} \quad (3)$$

where $S_{is} = (x_{is}, y_{is})$ and $S_{ie} = (x_{ie}, y_{ie})$ are the start and end points of the segment, respectively, with all P_j of M_i lying between these segment ends (see Figure 2.2 for an illustration).

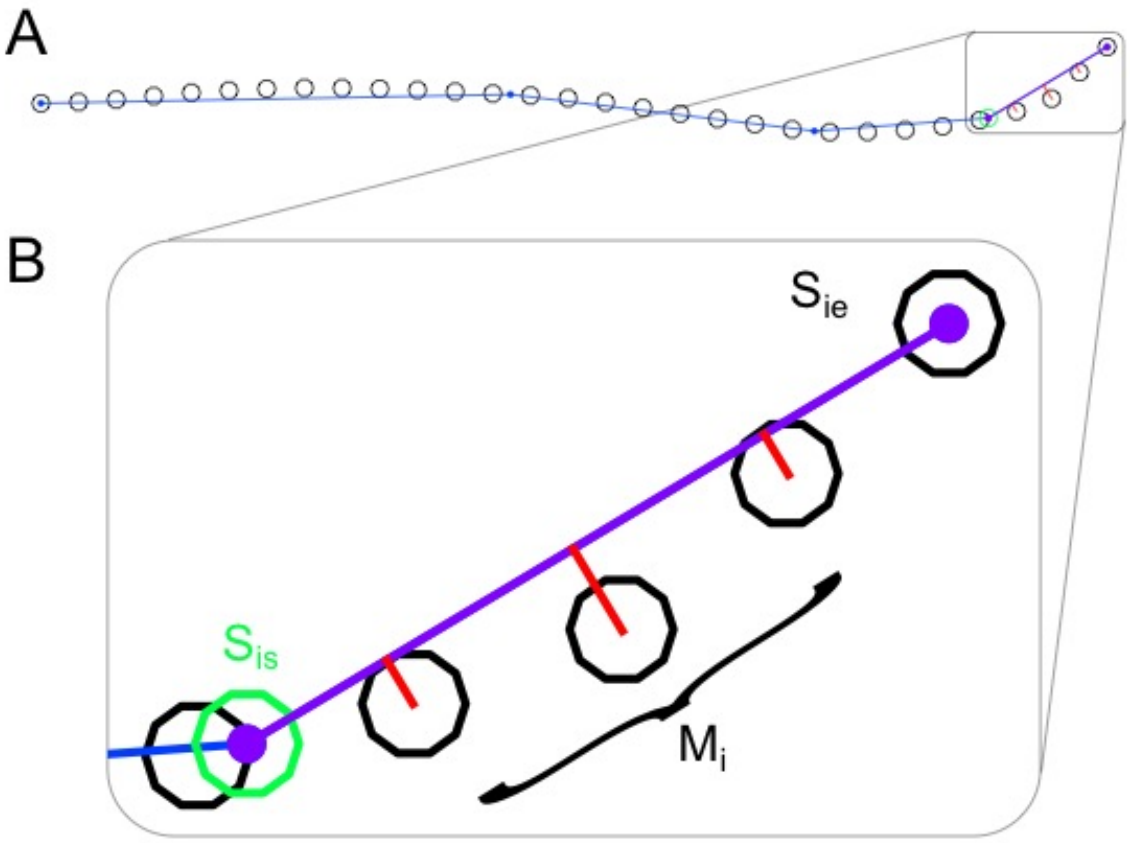


Figure 2.2: Determining the difference between a segment and midline at a single frame, by calculating midline point error (A) A trout midline at a single time frame, represented as a set of midline points (highlighted in black), and a multi-segment model including segments (blue lines) and joints (blue circles). (B) Calculating midline point error, for each associated midline point. Midline point error is defined as the distance between the segment (S_i, purple) and each of the midline points associated with it ($M_i = \{P_{j1}, \ldots, P_{jn}\}$, black circles). In this case, the segment starts at point S_{is} (highlighted in green) whose location is interpolated from the neighbouring midline points, and ends at point S_{ie} at the tail tip. These segment ends, and each of the midline points between them, would be input into Equation 3 to calculate the distance between the segment and the input midline point. To obtain the specific segment error, midline point error is calculated and averaged over each frame t, separately for each midline point, and the maximum average is selected as the specific segment error.

Segment End Position Calculation

Per Equation 3, it is necessary to determine the position of each end of the segment within the x y coordinate space. It is assumed that these segment end positions lie on the fish midline. However, unless they lie directly on top of the originally recorded midline points from the real midline, their coordinates need to be interpolated from these already recorded ones. In this case, these positions are simply interpolated

linearly from the nearest recorded midline points that they lie between. In Figure 2.2, point S_{is} is an example of a segment end whose coordinates need interpolating.

Selecting Midline Points for Distance Calculation

As a segment only models the subsection of a midline located between its segment ends, only midline points located within this subsection are selected for distance calculation. All other midline points are irrelevant for determining error produced by this segment. In Figure 2.2, the points that would be selected are all those within the midline subsection marked M_i.

Averaging Midline Point Error Over Frames

As midline point positions change between frames to represent fish movements over time, it is necessary to time average midline point errors. This time averaging is performed separately for each midline point, by calculating their MPE for each separate frame, then averaging it over all frames. This results in a set of time-averaged MPEs, one for each midline point.

Selecting Maximum Time-Averaged Midline Point Error

Once a time-averaged error is calculated for each midline point, the final step is to determine a single error for the whole segment. Here, out of all frame-averaged midline point errors, the maximum error is selected to represent the specific segment error:

$$SSE_i = \max_{j \in M_i} \frac{1}{t} \sum_{t=1}^{T} Distance\ (S_{is}, S_{ie}, P_j) \quad (4)$$

In Figure 2.2, out of all three midline points within the midline subsection marked M_i, the point in the middle would be the most likely to produce the maximum frame-averaged MPE.

Expected Properties, Strengths and Weaknesses

Given the nature of how they are calculated, both specific segment errors and midline point errors are expected to exhibit patterns in relation to segment and midline attributes. Regarding SSE, as it is based on maximum time-averaged MPE, which in turn is a reflection of the distance between the rigid segment and curved midline, both errors are likely to increase with increasing body curvature. Additionally, as shorter segments can fit curvature better, both errors are likely to increase with segment length, even if curvature does not change.

Furthermore, due to the effects of averaging, time-averaged MPE (and hence SSE) is expected to reflect curvatures and distances somewhere between their minimum and maximums intensities, at any point along the body. Thus, both errors are expected to increase monotonically with segment length and towards the posterior, as although curvature varies in a wavelike pattern along the body at individual time frames, average body curvature over all frames is expected to generally increase towards the tail.

There are, however, some noteworthy issues with these error measurement approaches. By simply using the maximum linear distance between the segment and a single midline point (rather than the entire area between it and the midline portion, as in [44]), it ignores the rest of the two-dimensional space between the segment

517 and midline, and can also be skewed by outliers. While averaging MPE over many
518 frames mitigates outliers to a certain extent, overfitting can still occur, as averaging
519 may produce errors smaller than actual peak MPE. However, as the data sets
520 analysed here are clean and were manually digitised, a conservative maximum error
521 approach is sufficient for generating accurate multi-segment models.

522 ### 2.2.2: Overall Model Error

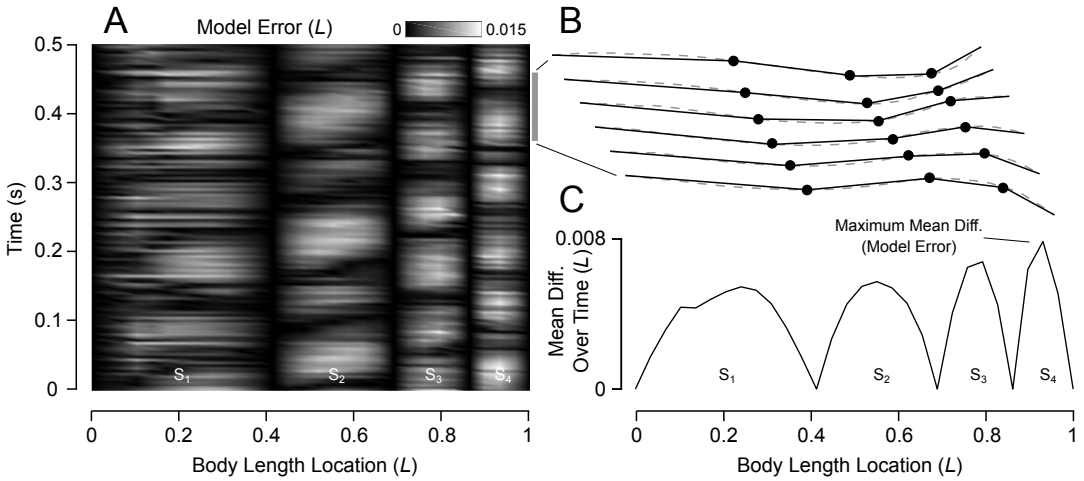
523 While specific segment error calculation is crucial, it is not enough for evaluating the
524 overall performance of a model. Overall performance can be useful for comparing
525 different models, especially considering issues with comparing models through
526 comparisons of their constituent parts. While it may be possible to compare models
527 using the error of each midline point, this increases the amount of data required over
528 comparing SSEs. However, issues can also occur with comparing models solely on
529 their SSEs. One issue is that, although it is intuitive to compare corresponding
530 segments between models with the same number of segments (e.g. comparing the
531 first segment of one model with the first segment of another model, doing the same
532 between each of the second segments, and so on and so forth), this is not the case
533 with models with different numbers of segments, as each segment in one model
534 does not correspond with exactly with one segment in the other model. Also, as
535 segment lengths often vary between models, similar segments between models may
536 represent slightly different areas of the midline. Given these dilemmas, what is a
537 good method of comparing models, when their components (i.e. segments) are not

always directly comparable? Consequently, for each model, it is often necessary to summarise their SSE values into an 'overall model error'.

Here, overall model error is calculated using a 'weakest link' approach. With this approach, out of all segments constituting a model, the SSE with the largest value (i.e. the segment that produces the largest SSE) is used as the overall model error. This ensures that, when comparing overall error, none of the SSE values of the model with the lower OME are higher than that of the other model. This is an issue with simply summarising error using averaging, as models with low average error can still have segments with high SSE. Based on this 'weakest link' approach, in Figure 2.3, the SSE of segment S_4 would be used as the OME. Unless stated otherwise, this approach is what is meant when referring to 'overall segment error' hereafter.

Strengths and Weaknesses

There are still limitations with this approach. While on average specific segment errors are guaranteed to be lower than the overall model error, there are occasional time points where they may be higher. Additionally, when comparing two models, just because a model has a lower OME does not necessarily mean that all its SSEs are lower. Given a lower OME model, while each of its SSEs will be below a higher OME, they are not guaranteed to be lower than all SSEs for that higher OME model. However, a positive effect of this approach is that, by attempting to minimise the OME of a model, its SSEs are inherently limited by the same amount as well.



559

560 *Figure 2.3: Evaluating the accuracy of a segment model. (A) A colour map showing the difference (L)*
561 *between the segment model and actual fish midlines along the body (horizontal axis) and over time*
562 *(vertical axis); dark colours (low difference) and light colours (high difference). (B) A series of fish*
563 *(dashed grey) and model (solid black) midlines over a half tail beat (the grey bar indicates the*
564 *selected time interval). Black circles (filled) indicate the joint positions of the model. (C) Mean*
565 *difference (averaged over time) along the body. The specific segment error for a given segment is the*
566 *maximum mean difference within that segment. The maximum specific segment error out of all*
567 *segments (i.e. the maximum mean difference along the entire body) is used as the overall model*
568 *error. Error is presented in midline body lengths (L).*

569 ## 2.3: Segmentation Approaches

570 Two methods are proposed here: a segment growing approach, and an evolutionary
571 algorithm approach. Key aspects of both algorithms (inputs, parameters, goals,
572 methods, and outputs) are compared and contrasted in Figure 2.4. Both algorithms
573 take a digitised midline as input. However, their other key aspects differ
574 considerably.

575 For the segment growing approach, the goal is to generate a model using the
576 smallest number of segments, whilst ensuring the overall model error stays below an
577 error threshold (the main parameter specified by the user beforehand). To achieve

this, the algorithm employs a linear process; starting from the head, it adds a new segment, grows it along the body until its specific segment error rises above the error threshold, and repeatedly adds and grows segments until the entire midline is covered. Ultimately, it outputs the number of joints and their locations that connect those best segments. It is a greedy algorithm: it attempts to find the locally best properties for each individual part of the model (i.e. each segment) sequentially.

In contrast, for the evolutionary algorithm approach, the goal is to generate a model with the lowest overall model error, whilst using a specific number of joints (the main parameter specified beforehand by the user). Its method of achieving this is an evolutionary algorithm, where potential segment lengths are iteratively improved through evolution, by selecting, merging, and altering many solutions over many generations. Ultimately, it outputs a best set of locations for the specified number of joints. Unlike the greedy strategy of the segment growing approach, it attempts to find a global best model as a whole, considering the combined effect of segments.

Below, each method is discussed in further detail. Along with the specifics of their main goals and associated parameters, additional parameters specific to the algorithms themselves are covered. Furthermore, the specifics of the logic used by each algorithm are further elaborated on.

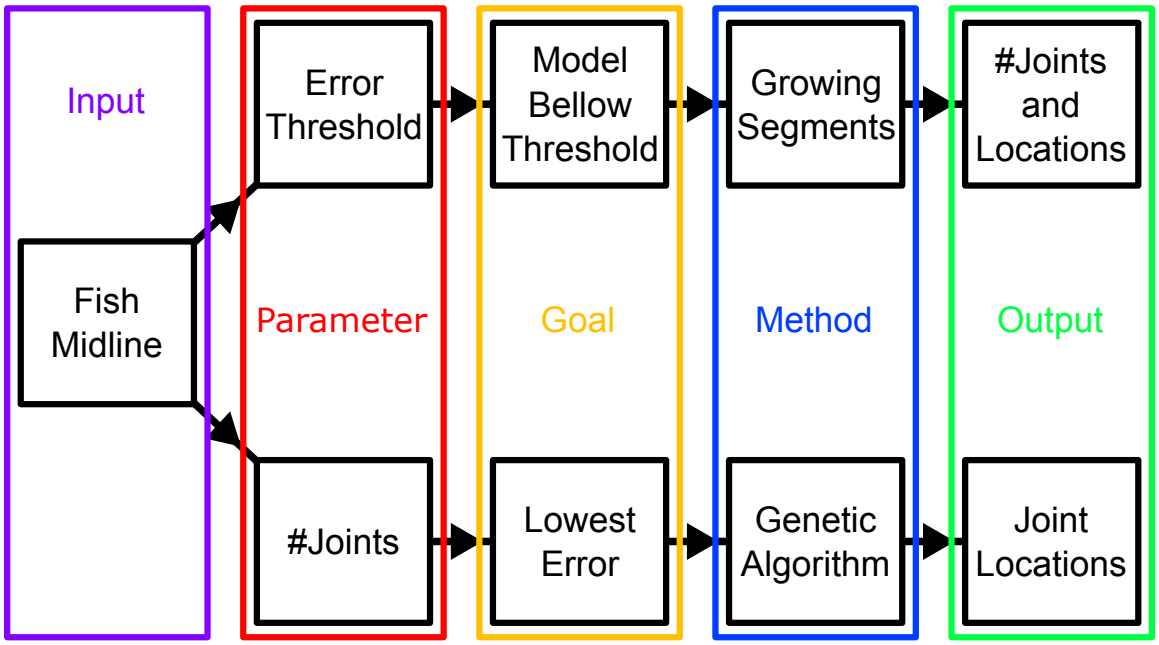


Figure 2.4: Summary of two segmentation methods: segment growing method (top) and evolutionary algorithm method (bottom). Each method takes digitised fish midline points as input. The approach of each algorithm can be defined by its parameters, goal, method, and output. For the segment growing approach, the main user specified parameter is a limit of the error between the model and midline, and the goal is to generate a model below that threshold using a minimum number of segments. Starting from the head, the segment growing method grows a segment along the midline until its error rises above the error threshold. It repeats this process of adding and growing segments, until it covers every midline point along the body. Finally, it outputs the number of joints and their locations used by the segments. In contrast, for the evolutionary algorithm method, the main user specified parameter is a fixed number of joints, and the goal is to find the lowest error model that uses this number of joints. The evolutionary algorithm method does this through adjusting joint locations using evolutionary methods to minimise overall model error. The best joints are then produced as the output.

2.3.1: Segment Growing Approach

For the segment growing approach, the goal is to find the best segment lengths whilst staying below the error threshold, by linearly adding and growing segments from head to tail. Starting at the fish nose (L = 0), the first segment of the model is added. This segment is then grown incrementally, until its specific segment error rises above the error threshold specified a priori by the user. At this point, the longest

segment length that did not rise above the error threshold is recorded as the best length for the first segment. A segment joint is then added at the end of the first segment, and the second segment is initialised. This process of adding and growing segments is repeated, with each subsequent segment starting at the joint at the end of the previous segment. This process continues until the tail tip is reached, and the entire midline is covered in segments. Finally, the algorithm outputs the number of segments used and the locations of the joints connecting them. The main logic of the overall process is outlined in Figure 2.5. The main processes of adding and growing segments are illustrated in Figure 2.6.

Thus, using the strategy of a greedy algorithm, this approach works towards minimising the segment number used, while adhering to the target error threshold and any other constraints. As longer segments are required to reduce segment numbers, and shorter segments are required to reduce error, the goal of reducing segment numbers while restricting error can be thought of as searching for the longest segments that are short enough to meet the error requirements. As this algorithm follows a principle of attempting to maximise value while minimising cost at each step (i.e. searching for the longest segment without violating the error threshold, for each part of the midline), it could be said that this algorithm closely adheres to the strategy and goal described above. While it can be said that the minimum segment length can potentially violate the error threshold in certain cases, if the minimum segment length is considered part of the optimisation problem (i.e. every segment must produce less error than the threshold *or* be no shorter than the minimum segment length), then these violations of the error threshold can be considered acceptable. That said, it is still not necessarily guaranteed to find the

optimal number of segments and their lengths, but rather tries to maximise segment length while adhering to the error threshold (and minimum segment length), with the assumption that this may likely lead to a best solution.

```
Algorithm segmentGrowing()
Inputs: fish midlines (M), error threshold (E_th)
Outputs: joint locations (J)
Parameters: ΔS, S_min
1   J = [], S_sta = 0
2   while S_sta <= 1 - (2 * S_min)
3     S_end = S_sta + S_min
4     while S_end < 1 && segmentError(M, S_sta, S_end + ΔS) < E_th
5       S_end += ΔS
6     end
7     if S_end < 1
8       if S_end > 1 - S_min
9         S_end = 1 - S_min
10      end
11      add S_end to J
12    end
13    S_sta = S_end
14  end
15  return J
```

Figure 2.5: Segment growing algorithm. The parameters ΔS and S_{min} are the segment length growth increment and minimum segment length, respectively. The inputs (fish midline and error threshold), outputs (joint locations), and parameters (ΔS and S_{min}) are all normalised with regard to the fish body length.

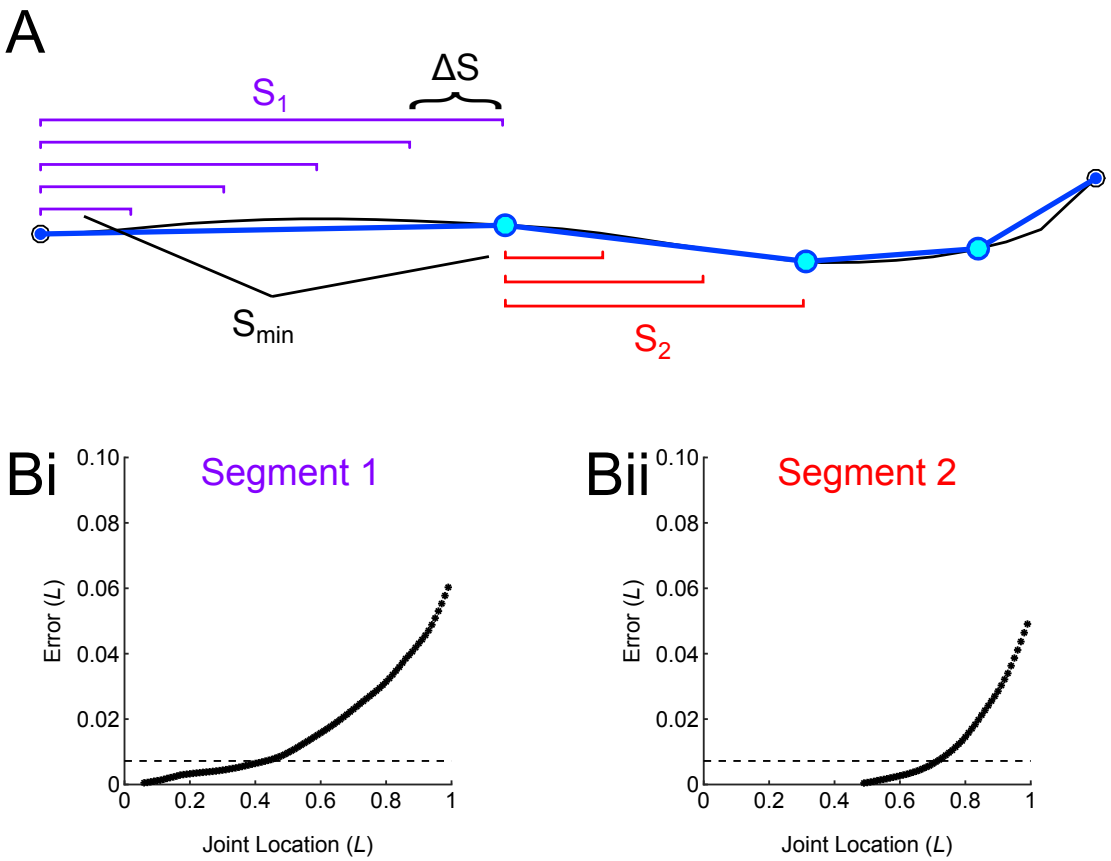


Figure 2.6: The segment growing process. (A) An actual midline and its corresponding segments, at a single time frame. Although this subfigure illustrates a single frame, the growth process actually considers all frames. This is because the specific segment error compared to the error threshold is based on time-averaged midline point error. Two segments (S_1 and S_2) are incrementally stretched over the midline. When S_1 reaches its maximum length before crossing the error threshold, the growth of S_2 starts. Each coloured length bar next to segment S_1 or S_2 represents one iteration of segment growth. S_{min} and ΔS indicate initial segment length, and segment length increment during segment growth, respectively (not to scale). (Bi and Bii) Error plots for S_1 and S_2 respectively as they increase in length. The dashed line is the error threshold. The black dots, whose y values increase with x, mark the incrementally increasing specific segment error values from growing the segments. The monotonical increase in error with segment growth is expected, as because the error values are time averaged, in general they reflect the average curvature of the body relative to the segment. Consequently, as average curvature is expected to increase towards the tail, so too is error. Error is presented in midline body lengths (L).

The segment growing approach has a runtime of $O(1/\Delta S)$. The main parameter that effects computational complexity is the segment growth increment length (ΔS). The shorter this parameter, the more checks are made as to whether the segment error has crossed the error threshold. The effect of the other parameters is expected to be less substantial, as at most they only reduce the number of these error checks, but

666 less than the total number of checks. Consequently, as the amount of complexity
667 due to the number of error checks is linearly proportional to one divided by the
668 segment growth increment length, complexity is expected to increase linearly in
669 $1/\Delta S$, and hence have a linearly increasing runtime.

670 **Error Threshold**

671 The main parameter that controls the segment growing approach is the error
672 threshold, E_{th}, which is specified by the user. This parameter ensures that the output
673 model is as accurate as this value, by ensuring every segment has specific segment
674 error below this threshold. As segments are grown until their SSE rises above E_{th},
675 each segment has the longest length possible starting from that midline location
676 given E_{th}. Consequently, not only does increasing and decreasing E_{th} increase and
677 decrease both SSE and segment lengths, it often decreases or increases segment
678 numbers as well. The error threshold and its effects are illustrated in Figure 2.6 Bi
679 and Bii. Additionally, the effects of altering the error threshold value are illustrated in
680 Figure 2.7.

681 **Growth Increment Length**

682 The increment length of segment length growth is controlled using a growth
683 increment parameter, ΔS. A segment is grown through incrementally increasing its
684 length by an incremental amount. ΔS is the distance that segment length is increased
685 by at each growth increment. This is a normalised distance along the body; for
686 instance, growth using $\Delta S = 0.1$ would result in incremental 0.1 *L* increases in
687 segment length. Larger values of ΔS result in faster segment growth, but raise the
688 risk of skipping over segment lengths that may be better but which lie between

growth increments. On the other hand, smaller values of ΔS result in greater precision in finding best segment lengths, but slow segment growth time and increase the number of computations needed. Interestingly, ignoring S_{min}, the value of ΔS could also predict the maximum number of joint locations the algorithm could consider: $\lceil (1 \,/\, \Delta S) - 1 \rceil$ joint locations ($\lceil\ \rceil$ being notation for ceiling rounding). For instance, if $\Delta S = 0.03$, then $\lceil (1 \,/\, 0.03) - 1 \rceil = 33$ joint locations are possible, as the combined space taken by all these possible joint locations is $0.03 \times 33 = 0.99$ of the available space (the entire length of the fish body), which does not technically allow room for any more joints. The ΔS parameter is illustrated in Figure 2.6 A by the difference in length between the nearest neighbouring segment length bars.

Minimum Segment Length

The algorithm also employs a minimum segment length parameter, S_{min}. This parameter restricts how short segments can be, with each segment being initialised with length S_{min} when first added. Without this parameter, if ΔS is small, segments can become excessively short as body curvature increases, particularly towards the tail. Consequently, although S_{min} can potentially increase the error of outputted segments, it can reduce unwanted extra short segments and hence potentially unnecessary complexity (that is, larger segment numbers result in a more complex model, in the sense that more points of movement need to be considered), while still allowing use of high precision ΔS values. As can be seen in Figure 2.6 A, this minimum segment length value is illustrated as the first and hence shortest length bar of each segment.

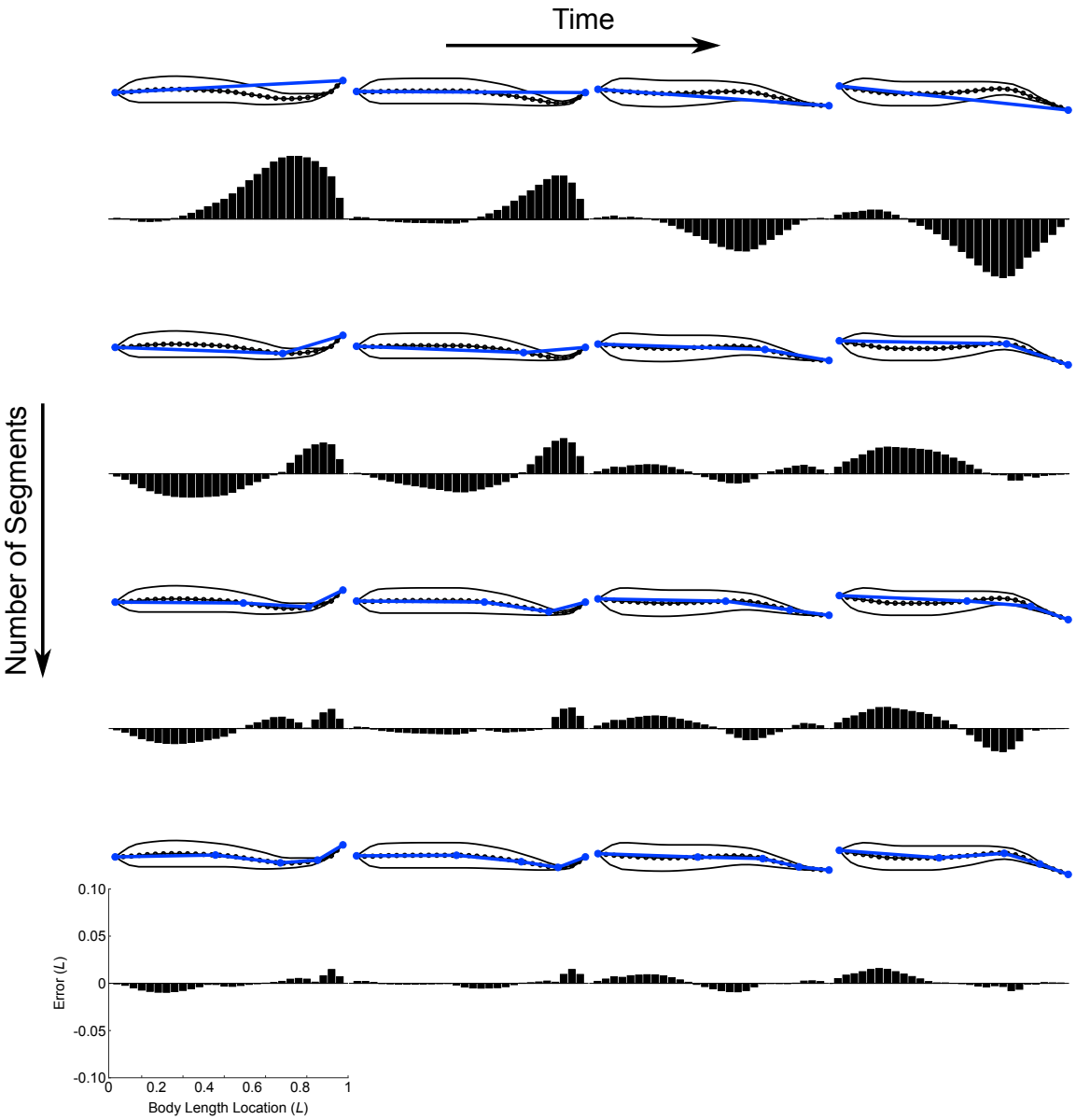


Figure 2.7: Midline point error versus segment number and time. The number of segments increases from one (top row) to four (bottom row), while each column corresponds to different time points. As the error thresholds get smaller, more segments are needed to represent the midline. The bars (black) show the midline point error between the multi-segment model and individual midline points. Crucially, these error bars get smaller as segment number increases. They also vary almost cyclically with time. Note that while midline point error stays below the thresholds for most of the time, it occasionally crosses the thresholds momentarily. This is because midline point error values are time-averaged when calculating specific segment error, which is the final value compared with the error threshold. Error is presented in midline body lengths (L).

2.3.2: Evolutionary Algorithm Approach

An evolutionary algorithm [46] approach is used here to find the best segment lengths for specific numbers of segments. This fixes a notable flaw with the segment growing approach, where a target segment number cannot be exactly specified beforehand. The evolutionary algorithm approach effectively finds the best lengths for specific numbers of segments through evolving many generations of multiple possible solutions, or 'chromosomes'. Each chromosome represents a possible solution, and consists of elements called 'genes', which represent parts of this solution. In this case, each chromosome is a possible set of joint locations (which determine segment lengths), and each gene is the location of a single joint.

The evolutionary algorithm approach starts by creating an initial generation of random chromosomes. From this, it then creates a new generation of chromosomes, using the prior generation as the basis for the next. It does this by creating 'offspring' chromosomes for the next generation from 'parent' chromosomes selected from the prior generation, using a process of fitness calculation, parent selection, crossover, and mutation. Fitness calculation consists of evaluating every chromosome in the prior generation using a fitness function, and assigning each a fitness score. Parent selection consists of selecting chromosomes from the current generation, to be used as parents for creating offspring chromosomes. Higher fitness (i.e. better) chromosomes have a higher probability of being selected, to encourage production of better offspring. All offspring then constitute the population of the next generation. While some offspring are simply clones of a single parent, crossover ensures that not every offspring is a clone, by creating offspring through mixing chromosome

elements, or 'genes', from two parents. Mutation consists of randomly applying additional adjustments to offspring elements to further explore the problem space. This process of creating a new generation is repeated many times, each time using the prior generation as the basis for the next. As higher fitness chromosomes are preferred during parent selection, genes that increase fitness are more likely to be passed on to future generations, while genes that decrease fitness are more likely to be discarded. Consequently, while no generation is guaranteed to improve average fitness compared with its immediate predecessor, over many generations, given enough selection pressure and crossover and mutation do not cause too much unwanted disorder, average fitness is likely to improve [46]. Finally, when the algorithm stops (in this case, after a set number of generations), depending on how effective the chosen parameters were and the nature of the problem space, the chromosomes with the highest fitness in the final generation are likely to represent solutions that are close to a number of local optimums or a single global optimum.

This process of evolution is illustrated in Figure 2.8, with each subfigure representing one generation, and each row within each subfigure representing an individual chromosome (i.e. 200 chromosomes per subfigure). Starting with the random individuals in generation G1, and using the procedures described above, subsequent generations are produced, whose chromosomes have a tendency to be fitter and increase in similarity with each other, compared with those in prior generations. Eventually, having all become more or less the same, they represent a single best set of segment lengths. This is essentially achieved by generation G5.

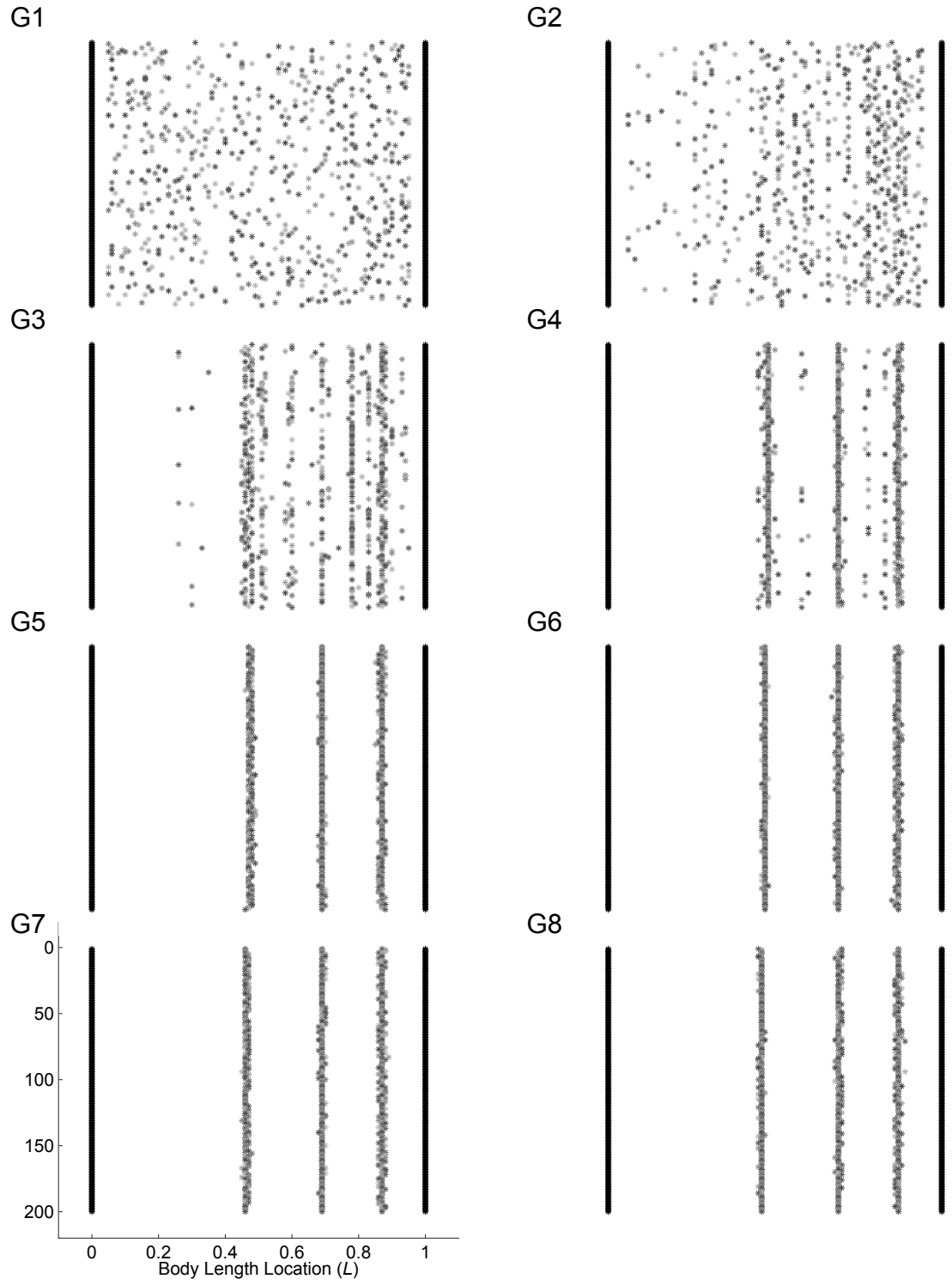


Figure 2.8: Evolution of chromosomes over 8 generations (out of 10 originally). Each chromosome is displayed as a set of joint locations. For each subfigure, each row along the y-axis represents one out of 200 chromosomes. The left and right vertical black lines show the fish snout and tail tip, respectively. (G1) First generation consisting of initial population of random chromosomes. (G2 - G5) Generation 2 to 5. Compared to each immediately prior generation, chromosome fitness/model accuracy noticeably increase, as well as similarity, which culminates in essentially all chromosomes representing a single best model. (G6 - G8) Generation 6 to 8. Negligible increase in chromosome fitness/model accuracy, with essentially no change in best chromosome/model. This trend also continues for Generation 9 and 10.

At this point, it is worth clarifying that, although the goal of this project was *optimisation of segmented models*, this does not include *optimisation of the algorithms themselves*. Thus, along with the fact that both algorithm implementations were made from scratch (i.e. without the use of external libraries), the decision was made to base the structure of the evolutionary algorithm on that of a Simple Genetic Algorithm (SGA) [46]. While it is fairly dated, it could still be sufficient for initial exploration of optimising segmented models using evolutionary algorithms. That said, while the general implementation remains similar to that of the basic SGA, in places it was modified to accommodate differences between the usual SGA implementation and the problem at hand (e.g. while bit strings are often used in SGA implementations, the segmented models described in this thesis are represented as series of non-negative real numbers ($\mathbb{R}_{0+}$)). The details of these deviations from the basic SGA structure are detailed in due course.

Representation and Chromosome Validity

Here, a chromosome is simply a set of joint locations, each gene being the location of one joint. Consequently, all chromosomes have the same length (given the fixed number of segments specified by the user). Thus, the algorithm controls the lengths of segments by altering the locations of joints connecting those segments. See Figure 2.9 for an illustration of chromosomes and their genes, as well as the sets of joint locations they translate into.

However, chromosomes need to be valid, otherwise they translate into illogical multi-segment models. This is due to the way the translation method constructs these models. Beginning at the nose, it essentially adds segments one at a time,

each subsequent segment beginning on the midline where the prior one ended. As it linearly reads the chromosome, the first gene defines where the first segment ends, the second where the second ends, and so on. Thus, progressively larger gene values produce segments that never overlap. However, if a later gene is smaller than the prior one, that segment will end nearer the nose than where it began. Consequently, it will cover a midline portion that has already been covered by a prior segment. As each midline portion should only be covered by one segment, this overlap results in an illogical model. Thus, in addition to the minimum segment length constraint, only chromosomes whose genes increase in magnitude from end to end are valid.

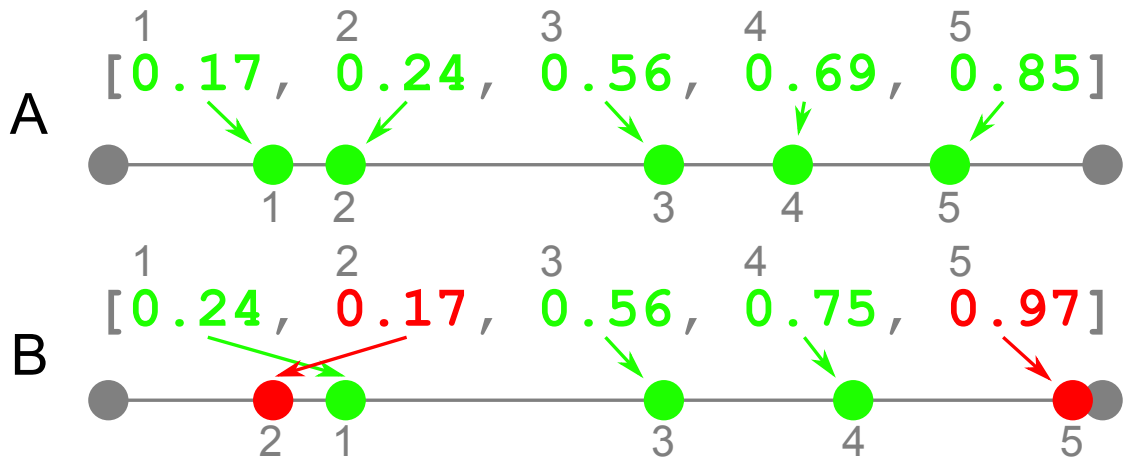


Figure 2.9: Illustration of the format of chromosome representation used in the research presented here, and the corresponding sets of joint locations they translate to. Each chromosome is a numerical array, where each element is a single gene. Each gene of a chromosome is a real-valued number, which translates into a real-valued location of a joint. The position of the gene within the chromosome array dictates which posterior segment end the corresponding joint is located at: Gene 1 is the location of the posterior segment end of the first segment (i.e. the segment going from the nose tip to first joint), Gene 2 is the location of the posterior segment end of the second segment (i.e. the segment going from first joint to second joint), and so on and so forth. Thus, genes can translate into segments that go backwards (which are invalid). Consequently, the validity of a chromosome as a solution depends upon whether the set of joint locations it translates into is a valid set of joints. (Green) valid genes/joint locations. (Red) invalid genes/joint locations. (A) A valid chromosome, and the valid set of joint locations it translates into. (B) An invalid chromosome, and the invalid set of joint locations it translates into. Although Gene 4 translates into a different joint location from Gene 4 in chromosome A, it is not invalid, as the resultant joint location does not violate any constraints. However, Gene 2 is invalid as it causes the second segment to go backwards over the first segment,

while Gene 5 is invalid as the sixth segment is shorter than the minimum segment length (in this case, $S_{min} = 0.05\ L$).

While the basis of the evolutionary algorithm implementation (SGA) usually uses bit strings as the chromosome representation, the decision was made to deviate from this due to the representation described above requiring little or no translation from the problem space. While using bit strings would require an encoding strategy to convert to and from the non-negative real valued segment lengths or joint locations, the chromosome representation presented in this thesis does not require this. Also, it has been previously shown that there are significant drawbacks to using a bit string chromosome representation to encode non-binary values. For instance, if more than one bit (i.e. gene) is used to represent an integer or real valued number, the significance of each bit may interfere with the likelihood of a value being changed into other values: given a binary representation of an integer value of 7, it is more likely that it will be changed into a 6 than an 8 by one of the variation operators, as the former only requires a one bit change, while the latter requires three [46].

Evolutionary Parameters

Two parameters which control the number of chromosomes processed by the evolutionary algorithm are the number of generations and the population size (i.e. the number of chromosomes that constitute a whole generation). The algorithm needs sufficient chromosomes per generation, and sufficient generations, to explore the solution space and find the best solution(s). In the case of the algorithm implementation used here, values for these two parameters are defined manually by the user prior to execution of the algorithm. Preliminary tests on a few data sets are usually necessary to find good values for these parameters, with specific good

values potentially being dependant on the specific data set/problem space. During execution, the algorithm continuously produces and evolves new generations, and stops when it has processed the number of generations specified by the user. It is more or less equivalent to a generational algorithm [46], as whenever a new generation is created the entire population is changed; all individuals from the prior generation are discarded and replaced by those in the new generation (except for clone individuals produced by crossover and that have not been mutated). Larger population sizes may potentially allow a greater variety of chromosomes per generation. Assuming this is taken advantage of through sufficient diversity of chromosomes, this may allow greater coverage of the solution search space at once. Similarly, larger generation numbers may potentially allow greater numbers of alterations to chromosomes, through crossover and mutations. If no other factors that could hinder exploration are present (such as the algorithm failing to explore chromosomes outside a local optimum), this may allow greater exploration of the search space over time. However, increasing either population size or generation number can also potentially increase the number of fitness function calculations, in turn increasing computational complexity and overall runtime.

There are also several other parameters which control specific aspects of the evolutionary process. These include the difference in parent selection probability for low and high fitness chromosomes, the probability of crossover being performed when creating offspring, and the probability of genes being mutated and the magnitude of these mutations. These other parameters are covered in detail in the corresponding subsections where the operators they are applied to are described.

Initial Population

The first step is to create the initial generation using random chromosomes. However, it is necessary to ensure these chromosomes are valid. Per the definition of valid in "Representation and Chromosome Validity", chromosome genes must progressively increase in magnitude from end to end, by at least S_{min}

In this case, to maintain this structure, genes are assigned semi-random values one at a time, with respect to neighbouring genes. An empty chromosome is created, where each gene has yet to have been assigned a value. Then, one of these unassigned genes is randomly chosen, and assigned a value randomly selected from a range bounded by their neighbours. Thus, it is always greater and smaller than the nearest preceding and succeeding assigned genes by one S_{min}, or multiple given intermediate unassigned genes, to provide sufficient remaining space. The exact range is calculated using Equation 5. This process of randomly choosing unassigned genes, and assigning semi-random values, is iteratively repeated, until all have been assigned values.

The range of possible values for a given gene is calculated by determining a lower and upper bound for this range. This requires thinking of genes as joints (J), as well as segment end points ($E = \{0, J, 1\}$). Thus, considering the gene in question as a joint, the range bound in either direction is determined in two parts. The first part is finding the nearest segment end that has been assigned a location. Considering segment end points is necessary, as there may be no joints that have been assigned locations in the direction in question. The second part is adding $S_{min} \times n$ to the location obtained from this segment end, n being the number of intermediate

segments. Either only one segment separates the joint in question and the nearest assigned segment end, or multiple given intermediate unassigned joints. Additionally, n is positive and negative for the lower and upper bounds, respectively. This enforces the minimum segment length between the joint in question, each intermediate joint, and the nearest assigned segment end. n is calculated as the difference in segment end indexes between the nearest assigned segment end and the joint in question. Joint indexes are converted to segment end indexes by adding one, as one segment end (the nose) precedes the first joint. Additionally, the specific possible locations within the range are defined as all ΔS multiples between the lower and upper bound. Thus, given J, E, S_{min} and ΔS, an unassigned joint $J_u = \emptyset$ can be assigned a location from the following range:

$$\left(E_i + \left(\left((u+1) - i\right) \times S_{min}\right)\right) \leq \Delta S \times \mathbb{Z} \leq \left(E_j + \left(\left((u+1) - j\right) \times S_{min}\right)\right) \quad (5)$$

where i and j are the indexes of the nearest segment ends whose locations are unassigned, that precede and succeeded u, respectively, and u is the index of the joint in question, J_u. Thus, given $J = \{\emptyset, \emptyset, 0.63, \emptyset\}$, $S_{min} = 0.05$ and $\Delta S = 0.01$, using Equation 5, the range of possible joint locations for joint J_2 can be defined as:

$$0.10 \leq 0.01 \times \mathbb{Z} \leq 0.58$$

Regarding the overall process of generating the initial population of random chromosomes, the random order that the genes are selected in and assigned values ensures that, while possible locations of joints later in the order are inevitably limited by the locations of joints earlier in the order, this bias is not always towards the same

joint each time, varying between chromosomes. The main logic of the overall process used to generate this population of chromosomes with random genes is presented in Figure 2.10, while Figure 2.8 G1 illustrates the end result of this process.

```
Algorithm createInitGeneration()
Inputs: population size (P_size), chromosome length (C_len)
Outputs: initial generation (P_init)
Parameters: ΔS, S_min
P_init = []
while count(P_init) < P_size
  c = nullArray(C_len)
  while count(nullIndexes(c)) < C_len
    g_i = randomSelect(nullIndexes(c))
    c[g_i] = randomSelect(jointLocationRange(g_i,c,ΔS,S_min))
  end
  add c to P_init
end
return P_init
```

Figure 2.10: Process for creating the initial population of the evolutionary algorithm consisting of random chromosomes - the first generation. Input P_{size} is the population size, the target number of chromosomes this process aims to generate. Input C_{len} is the length of each chromosome, as the number of genes that constitute each one. The output, P_{init}, is the population of the first generation of the evolutionary algorithm, consisting of random chromosomes. The parameters ΔS and S_{min} are the granularity of the possible joint locations and the minimum segment length, respectively. The variable c is the current chromosome being generated, while the variable g_i is the index of the gene whose value is currently being calculated. The function jointLocationRange() simply uses Equation 5 to return an array of possible joint locations, based on the values input into it.

Chromosome Fitness Calculation

A fitness function is used to calculate the effectiveness of each chromosome, so that better chromosomes can be selected as parents for the next generation. Here, the fitness of each chromosome is calculated from its joint locations using the 'weakest

link' approach described in Section 2.2.2. In other words, the fitness score of a chromosome corresponds to the overall model error.

Offspring Generation

After calculating the fitness of each chromosome in a generation, an improved generation needs to be created based on this prior one. To do this, methods are needed for selecting parent chromosomes from the prior generation (parent selection), creating new chromosomes through combining parents (crossover), and applying randomised modifications to ensure further exploration of the solution space (mutation). Additionally, similar to the initial generation, these new candidate chromosomes need to be validated before being added to the next generation. Further details regarding parent selection, crossover, and mutation are discussed below.

Parent Selection

Parent selection gives higher priority to fitter chromosomes, to pass on effective solutions and filter out ineffective ones. However, excessive bias towards fitter chromosomes hinders improvement, by limiting exploration beyond current best solutions. Thus, instead of being ignored entirely, lower fitness chromosomes are still given a low chance of being selected [46]. The parent selection approach used in the research presented in this thesis is what is referred to in the literature as 'ranked roulette wheel' parent selection. This selection method 'spins' a 'wheel' of chromosome assigned segments, each proportional in size to the rank of chromosome fitness relative to other chromosomes, then selects the chromosome assigned to the segment it stops on [47]. While it is usual for basic SGA

implementations to use a 'proportional roulette wheel' strategy for parent selection (were segment sizes are proportional to just the fitness of the chromosome), significant flaws with this selection strategy have been discovered. Specifically, as the segment size (i.e. selection probability) is proportional to the fitness score, a chromosome that is much fitter than other members of the population can dominate the selection process, hampering the survival chances of lower fitness chromosomes that are not related to the dominant individuals [46]. As ranked roulette wheel addresses flaws such as these, the decision was made to implement ranked roulette wheel parent selection instead for the evolutionary algorithm presented in this thesis.

Regarding the implementation used in this thesis, the roulette wheel 'segment' arc sizes are numerical probabilities, and the sum of these segments leads to a probability of 1. 'Spinning' this 'wheel' consists of randomly selecting a number in a 0–1 range. The selected 'segment' is that which, when combined with all prior (i.e. smaller) numerical probabilities, has the smallest cumulative probability greater than the random number. These numerical probabilities are calculated by normalising each rank, multiplying each normalised rank by the same exponent, and subtracting each resulting from the next largest result. In this case, the exponent is a logarithmic function, used to control the differences between probabilities. Hence, controlling this exponent increase or decrease the bias towards fitter solutions versus greater diversity, what is commonly referred to in the literature as 'selection pressure' [46]. However, compared with arbitrary exponents, this logarithmic exponent may allow more control over assigning percentages of low and high fitness chromosomes to probabilities that sum to percentages of the 0–1 probability range. Given a current population of chromosomes $P_{cur} = \{c_1, \ldots, c_{P_{size}}\}$, that are ranked $R = \{1, \ldots, P_{size}\}$

such that the lowest and highest fitness chromosomes have rank 1 and P_{size} respectively, the parent selection probability SP_i of the chromosome with rank R_i is calculated as:

$$SP_i = \begin{cases} (R_i \,/\, P_{size})^{\log_{L_p} L_r} - (R_{i-1} \,/\, P_{size})^{\log_{L_p} L_r}, & i > 1 \\ (R_i \,/\, P_{size})^{\log_{L_p} L_r}, & i = 1 \end{cases} \quad (6)$$

where L_p equates to approximately the percentage of chromosomes deemed as having low fitness, while L_r equates to the approximate percentage of chromosomes selected that will be from this group, due to being roughly the proportion of the 'wheel' assigned to them. It is expected that summing the probabilities of the lower $L_p \times P_{size}$ of the ranks may likely produce a cumulative probability roughly similar in size to L_p, based on preliminary experiments (see Figure 2.11). For instance, given a population of $P_{size} = 100$ chromosomes, $L_p = 0.75$ and $L_r = 0.1$, summing Equation 6 for the 75 lowest fitness chromosomes results in:

$$\sum_{i=1}^{75} SP_i\big(R_i, P_{size}, L_p, L_r\big) = 0.1$$

In other words, there is a fairly high selection pressure. There is approximately only a 10% probability that chromosomes with fitness rank less than or equal to 75 will be selected, while there is approximately a 90% probability that chromosomes with fitness rank higher than 75 will be selected as parents. Thus, the logarithmic exponent may control whether specific percentages of chromosomes are assigned selection probabilities below and above a certain level. Additionally, it ensures the selection probabilities monotonically increase with rank in a nonlinear curve.

However, preliminary experiments indicate that in some cases the predictions of L_p and L_r are less accurate, potentially more so with smaller population sizes. See Figure 2.11 for illustrations of the monotonically increasing probabilities, and how L_p and L_r affect the nonlinear curvature of these increases.

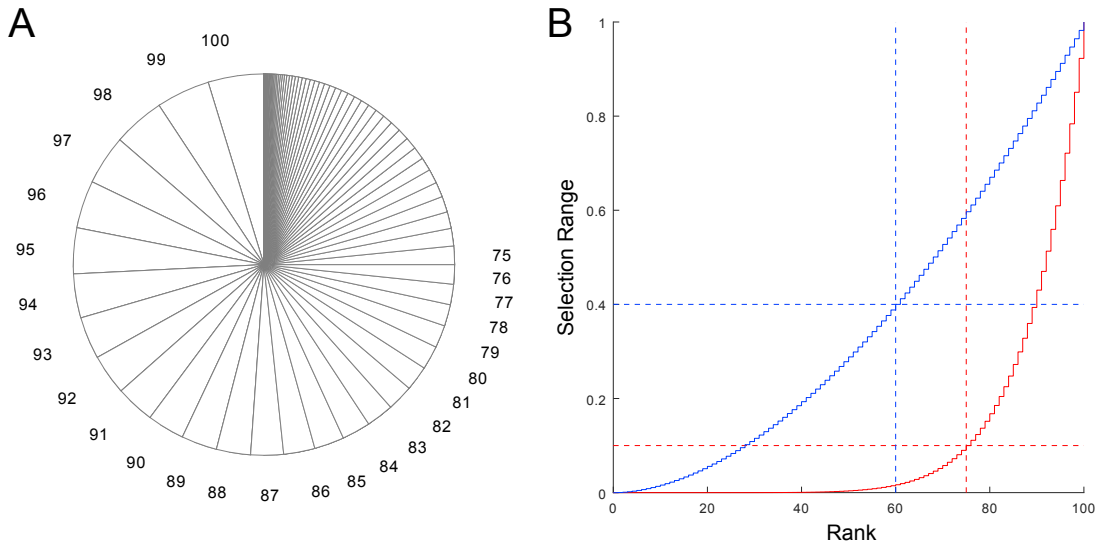


Figure 2.11: Illustrations of parent selection probabilities and how they can be adjusted. (A) 'Roulette wheel" illustration of example selection probabilities for each rank ($P_{size} = 100$, $L_p = 0.75$, $L_r = 0.25$). Note how they progressively increase in size in a nonlinear manner. Ranks 75–100 are labelled next to their respective selection probability 'segment'. Segments are presented for ranks 1–74, but are not labelled due to insufficient space. Also note how the first 25% of the 'wheel' contains the lower 75% of ranks, roughly matching the parameters L_r and L_p. (B) Illustration of cumulative increase in selection probabilities with rank, and how L_p and L_r affect these nonlinear curvatures of monotonically increasing selection probabilities ($P_{size} = 100$, Red, $L_p = 0.75$, $L_r = 0.1$, Blue, $L_p = 0.6$, $L_r = 0.4$). The entire length of the y axis is equivalent to the complete circumference of the 'roulette wheel'. The horizontal dashed lines indicate the values for L_r, while the vertical ones indicate the values for L_p multiplied by the population size. Note how the sharpness and steepness differs between the two curvatures. Also note how, in each case, roughly all ranks lower than L_p percent of the population size are below L_r, and vice versa.

Following from this, parent selection is conducted by selecting parent pairs one at a time, with crossover and mutation being performed, and the subsequent offspring being added to the next generation if they are valid, before another parent pair is selected. When selecting a parent pair, the first parent can potentially be any

chromosome, subject to its selection probability. While this is also true for the second parent, it cannot be the same 'individual' (i.e. have the same index within the population array) as the first parent; in other words, chromosomes cannot mate with themselves, although there is still a possibility that clones may be selected accidentally due to the limitations of the implementation. If the random selector selects the same individual as the first parent, selection is repeated with a different value for the random selector, until a different individual is selected. If either of the offspring translates into an invalid joint set, then both are discarded, and a new pair of parents are selected, until a parent pair produces two valid offspring. In this case these offspring are added to the next generation before starting parent selection again. The process of selecting parent pairs, and generating offspring before selecting another pair, is repeated until the next generation is fully populated with offspring. Given a population of size P_{size}, a total of $\lceil P_{size} \,/\, 2 \rceil$ parent pairs that produce valid offspring are selected. If P_{size} is an odd number, then only the first offspring of the last parent pair is added to the next generation. For instance, if $P_{size} = 101$, and consequently $\lceil 101 \,/\, 2 \rceil = 51$ parent pairs are selected, then only the first offspring of the 51st parent pair is added to the next generation. The main logic for creating the population for the next generation, including parent selection, is presented in Figure 2.12.

```
Algorithm createNextGeneration()
Inputs: current generation (P_cur), population size (P_size)
Outputs: next generation (P_nxt)
Parameters: L_p, L_r, CR_r, M_r, M_p
1  P_nxt = []
2  while count(P_nxt) < P_size
3    PR_idx = []
4    while count(PR_idx) < 2
5      selector = randomFloat(0,1), SP_cum = 0, R_i = 0
6      do
7        R_i++
8        SP_cum += probabilityOfSelectingRank(R_i, P_size, L_p, L_r)
9      until SP_cum >= selector
10     pr_i = getPopIndexOfChromosomeWithRank(P_cur, R_i)
11     if PR_idx[1] != pr_i
12       add pr_i to PR_idx
13     end
14   end
15   PR = P_cur[PR_idx], O = []
16   if randomFloat(0,1) <= CR_r
17     O = crossover(PR)
18   else
19     O = clone(PR)
20   end
21   O = mutation(O, M_r, M_p)
22   if offspringAreValid(O)
23     for o_i = 1; count(P_nxt) < P_size && o_i <= count(O); o_i++
24       add O[o_i] to P_nxt
25     end
26   end
27 end
28 return P_nxt
```

Figure 2.12: Process for creating new generations, including parent selection and invalid offspring handling. The inputs P_{cur} and P_{size} are the population of the current generation and the target

population size of the next generation (usually an identical number to the previous), respectively. The output P_{nxt} is the population of the next generation. The parameters L_p and L_r are the respective parameters from Equation 6 which control the parent selection pressure. The second while loop is the logic for selecting parents, while the second and final if statement is the logic for handling invalid offspring. The 'spin the roulette wheel' element of the parent selection is effectively the function randomFloat(), while the 'which wheel segment has been selected?' element is the do until loop. Variables R_i, SP_{cum}, and $selector$ are the rank currently being considered, the cumulative selection probability of this and all lower ranks, and 'where the wheel stopped', respectively. As the cumulative probability of all ranks equals one (all 'segments' equal one 'wheel circumference'), the selected rank is that where the cumulative probability of all lower ranks is less than the selector (the 'segment' where all others before it are behind 'where the wheel stopped'). The function probabilityOfSelectingRank() simply uses Equation 6 to determine the chance of selecting a chromosome with rank R_i in a population of P_{size} given selection pressure defined by L_p and L_r. The parameter CR_r is the probability that crossover will be performed. The parameters M_r and M_p are the probability of a gene being mutated (applied separately to each gene), and the magnitude of each mutation, respectively.

Crossover

While some offspring are simply clones of one of their parents, the crossover operator ensures many are produced through combining sections from both their parents. In the best-case scenario, this combines both parents most effective elements, producing a superior chromosome. Although in the worst-case scenario the least effective elements produce an inferior chromosome, subsequent parent selection may filter it out. The probability of crossover occurring is controlled by the crossover rate, CR_r (versus the clone rate, CL_r). Parents are split into sections using crossover points, and crossover offspring are created by combining sections on opposite sides with those from opposite parents. Here, a single crossover point is used, CR_p, situated almost exactly half-way along the chromosome. Regarding implementation, CR_p is a joint index, and joints prior and including CR_p and joints after CR_p are combined from opposite parents. This is calculated as $CR_p = [C_{len} \,/\, 2]$, where C_{len} is the number of joints in the chromosome. Figure 2.13 illustrates this crossover approach being applied to two parents. As $C_{len} = 5$ and $[5 \,/\, 2] = 3$, Offspring 1 is composed from the first three joints of Parent 1 and the last two of

Parent 2. Likewise, Offspring 2 is composed from the first three and last two joints of the reversed parents. Note that this can lead to invalid joints (i.e. joints that are located before preceding joints, as described in prior sections). However, this is delt with after mutation (irrespective of whether any mutations actually occurred), before offspring are added to the new generation, as discussed in "Offspring Validation".

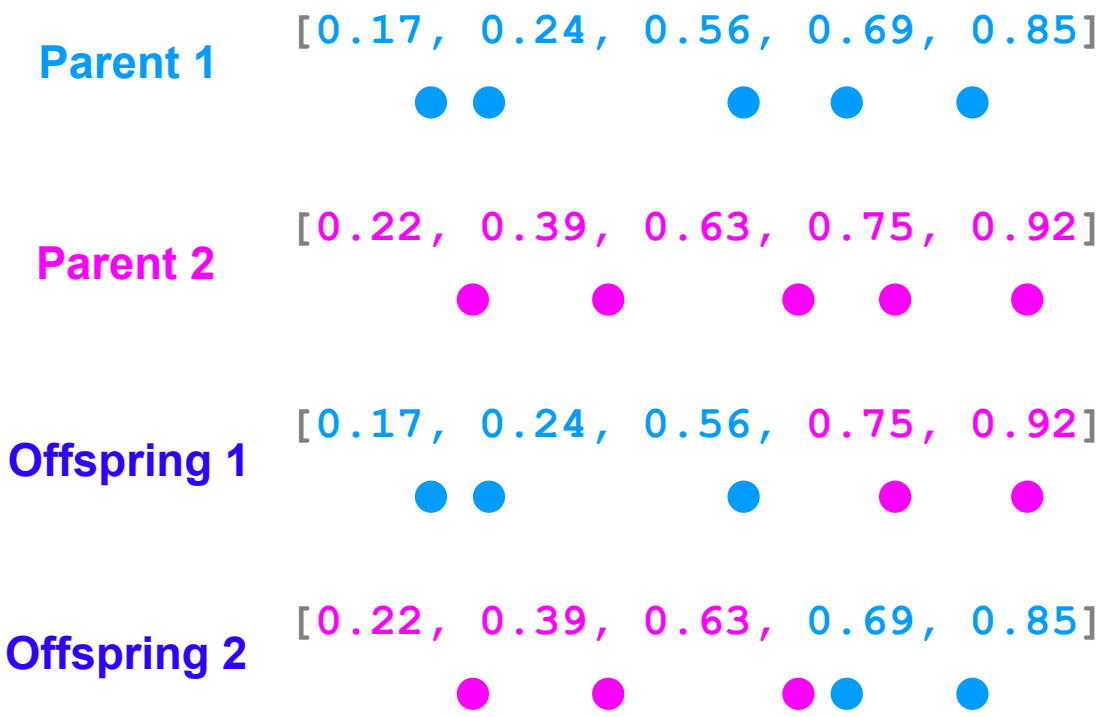


Figure 2.13: Applying crossover during offspring generation. As crossover point CR_p is calculated as $\lceil C_{len} \,/\, 2 \rceil$, and given that in this case $C_{len} = 5$, $CR_p = \lceil 5 \,/\, 2 \rceil = 3$. Consequently, Offspring 1 is constructed from joints from Parent 1 with indexes less than or equal to three, and joints from Parent 2 with indexes greater than three, in that order. Offspring 2 is constructed similarly, except the order of the parents which the joints are selected from is reversed.

Mutation

Mutation randomly alters offspring chromosomes, to help explore solutions beyond their parent chromosomes. Here, mutations are applied using a basic variation of

'Creep Mutation' strategy, by randomly shifting individual joints by short distances along the body [46]. Each offspring can potentially have multiple joints mutated, as the chance of mutation applies separately to each joint. Additionally, as both directions need exploration, the direction a mutation shifts a joint is also randomised (either anteriorly or posteriorly). Two key parameters control mutation: mutation rate, M_r, and power, M_p. M_r controls the likelihood of mutations occurring (the chance of a joint being shifted). However, excessively increasing M_r could destabilise the process, as similarity between individuals collapsing upon a single best solution is less likely when excessively altering solutions. Meanwhile, M_p controls the degree each mutation alters a solution. Here, M_p is the distance a mutated joint is shifted along the body. Thus, increasing M_p increases the extent of exploration beyond current joint locations, also increasing exploration rate. However, excessively increasing M_p risks mutations jumping over better joint locations. M_p could also be considered equivalent to ΔS in the segment growing approach. Figure 2.14 illustrates mutation of a single chromosome/joint set. For each joint, whether it will be mutated and in which direction is randomly determined, forming a mutation vector applied to the whole chromosome. Consequently, Joints 2 and 5 are mutated towards and away from the tail, respectively. While the usual case with the basic SGA is to use bit flip mutation, it is not meaningful to do exactly this with the chromosome representation chosen in this thesis, as it is not composed of bits. Also, while a potentially effective method of controlling the mutation step size is by sampling from a Gaussian distribution [46], the decision was made to make the mutation step size constant for ease of implementation.

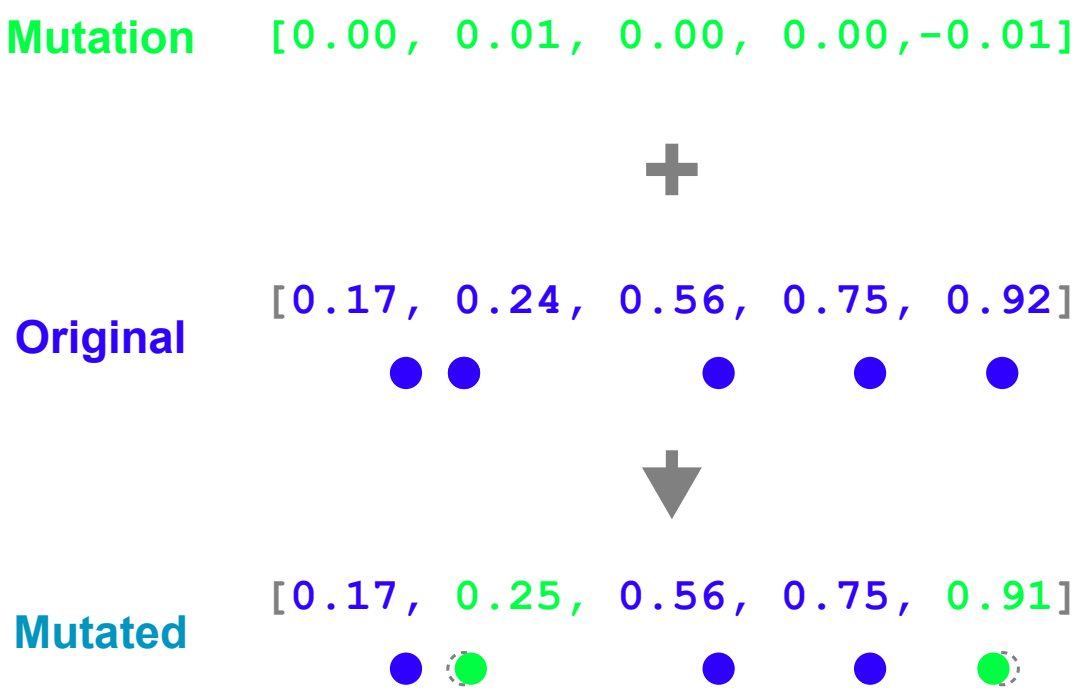


Figure 2.14: Mutation of an offspring. A mutation vector is generated for each offspring, each value corresponding to a joint in its associated offspring. Each value is by default 0, but has a randomised chance proportional to the mutation rate M_r of being the mutation distance M_p. Also, each value has an equal probability of being in either direction along the midline (positive or negative). Mutation vectors are added to their associated offspring, non-zero values mutating their corresponding joint.

Offspring Validation

Before adding an offspring to a new generation, it is necessary to ensure its chromosome is a valid joint set, as described previously. Even if both its parents were valid joint sets, combining them through crossover and applying mutations can still result in joints becoming invalid. Consequently, validation can only be performed after all changes due to crossover and mutation are made. Valid offspring are added straight to the new generation. However, if an offspring is determined to be invalid, it is discarded entirely, and the process starts parent selection, crossover, and mutation again.

3: Results

This section presents the results from verifying the two automatic segmentation methods described in the prior section (segment growing approach, and evolutionary algorithm approach) using actual datasets. Both methods were applied a variety of fish midline datasets, obtained from a previously published study [30], and the outputs produced by these two methods are analysed. This analysis focuses on answering two general questions: 1) What is the minimum number of segments necessary for modelling fish body movements accurately, and 2) How do segment numbers and joint locations vary between models of different fish species, swimming behaviours, and swimming speeds.

The results are presented in four subsections, each focusing on a different objective: (Section 3.1) comparing the performance of multi-segment models generated by the automatic segmentation methods to models with equal length segments, (Section 3.2) investigating how multi-segment models vary between different species during steady swimming, (Section 3.3) investigating how multi-segment models change between different behaviours, and (Section 3.4) investigating how multi-segment models change between different swimming speeds.

General Methodology and Parameters Investigated

All code implementations and data analysis were made using custom written Octave scripts (Octave 4.4.1). This includes implementations of both automatic segmentation algorithms. All midline datasets were normalised using their respective body length (L), and all distance measures are presented in L. All datasets were

processed using both automatic segmentation algorithms, the models and results they outputted were analysed, and overall model errors were calculated afterwards.

Parameter Tuning

Regarding parameter tuning, for the segment growing approach, the focus was on finding appropriate values for the segment growth increment (ΔS) and the minimum segment length (S_{min}). For ΔS, a value of 0.01 *L* was chosen, for multiple reasons. 0.01 is a fairly straightforward unit to work with: 1 / 0.01 = 100 units, which as a multiple of 10 is fairly intuitive to perform calculations with (compared with say 30 units, with each segment length being a multiple of 0.03, which are less familiar units to work with than multiples of 10). Also, it intuitively translates into measurements such as percentages (it could be useful to express segment lengths as percentages of the whole midline length). Furthermore, while smaller multiples of 10 could be more precise, they lead to longer runtimes (see Table 1). So, from this perspective, $\Delta S = 0.01\ L$ was chosen to minimise runtime costs. As for S_{min}, a value of 0.05 *L* was chosen, as this is intuitive to work with in a similar way to 0.01, large enough to not be inconsequential (for instance, it may not be worth setting $S_{min} = 0.02\ L$, as it is not much different from ΔS), but small enough to not increase overall model error to an excessive degree.

Wall-Clock Runtimes: Segment Growing Approach (E_{th} = 0.01 L, S_{min} = ΔS)		
ΔS (L)	**Resultant Segment #**	**Runtime (s mean ±SD)**
0.1	4	~0.17 ±0.02
0.01	3	~1.45 ±0.08
0.001	3	~13.91 ±0.40
0.0001	3	~140.18 ±1.76
0.00001	3	~1394.90 ±9.33

Table 1: The mean runtime (s) of the segment growing approach given different values for ΔS. The error threshold used in these cases was constant (0.01 L), and given this threshold the algorithm outputted models that used similar numbers of joints (most often three joints) and similar joint locations each time.

Regarding parameter tuning for the evolutionary algorithm approach, the focus was on finding values for the population size and the number of generations. A range of values were investigated for these two parameters (see Table 2). It was found that, when both of these parameters were set to fairly large values (200), the algorithm ran quite slowly. As a total of 45 midline datasets were tested to generate the results data discussed in sections 3.1, 3.2, 3.3, and 3.4, it would be very time consuming to test the algorithm on all these datasets with both population size and generation number set to large values. Ultimately, it was decided that the generation number would be set to a small value, while the population size would be set to a large value. While a large generation number with a small population size could be used, it was thought that there would be a higher chance of finding a better solution if the initial coverage of the solution space was fairly broad, which may be more likely given a large population size. Following from this, the selection pressure (i.e. the selection bias towards fitter solutions) was also set fairly high, to increase the likelihood that

chromosomes would evolve towards a good solution despite the small generation number chosen, by increasing focus on the better chromosomes to a large degree.

Wall-Clock Runtimes: Evolutionary Algorithm (s mean ±SD)						
		Population Size				
		10	**50**	**100**	**150**	**200**
Number of Generations	**10**	~1.28 ±0.01	~6.28 ±0.06	~12.95 ±0.62	~20.08 ±1.48	~25.46 ±0.22
	50	~8.45 ±3.36	~32.26 ±0.42	~65.01 ±1.13	~96.59 ±1.09	~129.08 ±1.05
	100	~13.37 ±0.51	~64.12 ±0.61	~128.26 ±0.78	~193.61 ±1.94	~257.76 ±1.41
	150	~19.54 ±0.20	~97.93 ±1.56	~193.11 ±1.66	~290.99 ±2.57	~387.08 ±3.86
	200	~27.04 ±1.60	~128.43 ±0.95	~257.52 ±1.51	~387.90 ±2.94	~517.44 ±2.30

Table 2: The mean runtime (s) of the evolutionary algorithm given different values for the population size and generation number. In these cases, the number of joints was kept constant (three joints).

However, less focus was put on the mutation power (M_P) and minimum segment length, as they were set to the same values as the equivalent parameters used in the segment growing approach (ΔS for the former). These values were made consistent so that comparison between the two approaches could be more straightforward, reducing the number of differences that needed to be taken into account. Additionally, regarding the seed used for the randomised elements of the evolutionary algorithm, this research used the default settings for the Octave random

number generator (in this case, from /dev/urandom, see Octave 4.4.1 documentation for more information).

Generation of Multi-Segment Models

Regarding generation of the models used for obtaining the results data discussed in the results section, for segment growing approach, segment length increment was set to $\Delta S = 0.01\ L$, while minimum segment length was set to $S_{min} = 0.05$. Initially, many models were generated per midline dataset using various error thresholds, ranging from 0.0001 *L* to 0.1 *L* in 0.0001 *L* increments (i.e. 1000 models per midline dataset). Then, for each joint number ranging from 1 to 9, out of all models generated with the same joint number, the one with the lowest overall model error was selected as the best. Thus, the best models generated for 2 to 10 segments (1 to 9 joints) were used in the results for comparisons.

Likewise, for the evolutionary algorithm approach models used for obtaining the results data, multi-segment models ranging from 2 to 9 joints (3 to 10 segments) were generated and tested. However, as a more direct strategy (or even a brute force strategy) can easily generate accurate two-segment models, for this case, instead of using the evolutionary algorithm approach, a single best joint location was determined iteratively, by evaluating all possible joint locations from head to tail. For this analysis, the same minimum segment length and segment length increment were used as specified above. For each model with more than one joint generated using the evolutionary algorithm, solution population sizes of 200 chromosomes were tested per generation, for 10 generations. For parent selection, the parent rank bias control parameters were set to $L_p = 0.75$ and $L_r = 0.1$ respectively. Parent

crossover rate (versus cloning rate) was set to $CR_r = 0.3$ (versus $CL_r = 0.7$), while mutation rate was set to $M_r = 0.05$. Similar to segment growing approach, minimum segment length was set to $S_{min} = 0.05\ L$, while mutation power was set to $M_P = 0.01\ L$, being equivalent to segment length increment in the former algorithm. For each segment number, only one best model generated was selected for producing the results. The limited number of models selected was due to the number of models generated being limited by the runtime cost of generating models using the evolutionary algorithm.

Additionally, to evaluate the benefit of the 'uneven' segment lengths used by models generated from the automatic segmentation algorithms, comparisons were conducted between the performance of these models, and the performance of simplistic 'equal-length segment' models, that were generated specifically for these comparisons (Section 3.1).

The analysis also looked at the most parsimonious multi-segment models (i.e. most accurate models using a minimum number of segments) that can represent fish midlines with a certain accuracy. For these cases, a maximum overall model error of 0.01 *L* was used. For these tests, changes in the number of segments and the location of joints used were compared between different fish species (Section 3.2), swimming behaviours (Section 3.3), and swimming speeds (Section 3.4).

Datasets Tested

For the tests presented here, the segmentation algorithms were applied to datasets from a previously published study [30]. These were 1) Rainbow trout (*Oncorhynchus*

mykiss) datasets, and 2) Multi-species specimen datasets. The details of the individual specimens are presented in the text and tables in each 'Datasets' subsection of each of the four main sections (Section 3.1, Section 3.2, Section 3.3, and Section 3.4). Each specimen was recorded in a flow tank using a high-speed camera (more than once in some cases). Each midline was digitised into x and y coordinates for 30 midline points, covering from head to tail with near equal distances between.

3.1: Equal vs Automatic Segments

Objective

For these tests, the main objective is to investigate how multi-segment models generated using automatic segmentation approaches (segment growing approach and evolutionary algorithm approach) improve over models consisting of equal length segments. When comparing models that consist of the same number of segments, how does accuracy differ between automated versus equal-length segment models? Alternatively, how do segment numbers and joint locations differ between automated segment models and equal-length segment models, to achieve the same level of accuracy?

Method

Multi-segment models of varying segment numbers were generated using equal-length, segment growing, and evolutionary algorithm approaches. The resultant models were then compared based on overall model error. Considering

each model, the analysis focused on how OME decreases with increasing segment numbers, and how these decreases differ between segmentation approaches. Additionally, the analysis investigated how error varies along the body at different locations, and for different approaches. Also, the analysis studied how joint locations differed between approaches and segment numbers.

Datasets

A rainbow trout of length 23 cm, recorded during steady swimming at a speed of ~3.4 L s^{-1} over three tail beats, was used as the primary data set for these particular tests.

Results

When using automatic segmentation approaches, four segments were sufficient to describe the movement of the fish body midlines with an overall model error less than a 0.01 L threshold. Thus, the automatic segmentation approaches did generate noticeably more parsimonious models, compared with equal-length segments (which required seven segments to stay below the same threshold).

Similarly, when comparing models with equal numbers of segments, automated segment models were noticeably more accurate than equal-length segment models. For instance, for four-segment models, relative reduction in OME was approximately ~69% (~0.0071 L < ~0.0228 L). For models with many segments (e.g. eight segments), the absolute difference between the two approaches was less pronounced. For overall model error produced by all approaches for 1–10 segments, see Table 3.

Overall Model Error for Different Segmentation Approaches and Segment Numbers (*L*)				
		Segmentation Approach		
		Equal Segments	**Segment Growing**	**Evolutionary Algorithm**
Number of Segments	**1**	~0.0630		
	2	~0.0474	~0.0262	~0.0255
	3	~0.0318	~0.0126	~0.0126
	4	~0.0228	~0.0071	~0.0082
	5	~0.0165	~0.0051	~0.0053
	6	~0.0124	~0.0036	~0.0047
	7	~0.0082	~0.0031	~0.0033
	8	~0.0060	~0.0024	~0.0032
	9	~0.0047	~0.0028	~0.0026
	10	~0.0038	~0.0028	~0.0027

Table 3: Overall model error for the three segmentation approaches tested, for 1 to 10 segments. There is only one error value for the single-segment model, as the model for this segment number does not change between approaches. It is expected that the slightly larger values for automatic approach models with 9–10 segments are related to a mixture of the bias towards growing segments near the head first and the minimum segment length for segment growing approach, and not properly finalising search for best joint locations for evolutionary algorithm approach. Error is presented in midline body lengths (L).

Additionally, in equal-length segment models, the error distribution across segments is not uniform. For example, with the four-segment model, the last two posterior segments have more error than the first two anterior segments. In contrast, this problem is mitigated by the 'uneven' segment lengths used by the automated models, thanks to shorter segments in the posterior region.

Analysis of joint locations for models suggest that, for best performance, segment lengths should get progressively shorter moving towards the body posterior (for models with less than seven segments). This is not unreasonable, as for most fish the posterior body moves and bends more than the anterior body. Below, each of these points are examined in further detail.

Error and Segment Numbers

Models generated using automatic segmentation approaches are noticeably more accurate at modelling fish swimming, compared with equal-length segment models, regardless of the number of segments used in the comparison (see Figure 3.1 and Table 3). As a baseline (i.e. worst-case scenario), a single-segment model (which assumes that the whole fish body is a rigid plate) produced a maximum overall model error of ~0.0630 *L*. When considering two, three, and four segments, equal-length segment models reduced OME to ~0.0474 *L*, ~0.0318 *L*, and ~0.0228 *L*, with absolute OME reductions of ~0.0156 *L*, ~0.0312 *L*, and ~0.0402 *L*, respectively. However, for multi-segment models generated using segment growing approach (SG), OME reduced further, to ~0.0262 *L*, ~0.0126 *L*, and ~0.0071 *L*, with absolute OME reductions of ~0.0368 *L*, ~0.0504 *L*, and ~0.0559 *L*, respectively (for more information on how best error values were found for specific segment numbers despite users lack of control over this with segment growing approach, see subsection "Generation of Multi-Segment Models" under Section 3). Similarly, for models generated using heuristic automatic segmentation methods (i.e. single best joint (SBJ), and evolutionary algorithm approach (EA)), OME reduced to ~0.0255 *L*, ~0.0126 *L* and ~0.0082 *L*, with absolute OME reductions of ~0.0375 *L*, ~0.0504 *L* and ~0.0548 *L*, respectively.

The same trend continues, even for high fidelity models with more segments (up to 10 segments). This is illustrated in Figure 3.1; both step plots for SG and EA models remain below the step plot for equal-length segment models, from two segments onwards. However, as the number of segments increases, the differences between the automated segments models and equal-length segment models become less prominent. For example, the absolute overall model error difference between SG models versus equal-length segment models are ~0.0212 *L* (two segments), ~0.0192 *L* (three segments), and ~0.0157 *L* (four segments). This indicates that, as the number of segments used increases, the specific segment length ratios used become less important.

Additionally, evaluation was performed on how the accuracy of models vary when increasing numbers of segments. This was done by examining the rate of decrease in overall model error, when segment number is incremented by one. OME for two, three, and four-segment equal-length segment models, compared to that of their immediately preceding segment numbers, decreased by ~0.0156 *L*, ~0.0156 *L*, and ~0.0090 *L*, respectively. Similarly, OME for two, three, and four-segment SG models, compared to that of their immediately preceding segment numbers, decreased by ~0.0368 *L*, ~0.0136 *L*, and ~0.0055 *L*, respectively. Also, OME compared to its immediately preceding segment number for the SBJ segment model decreased by ~0.0375 *L*, and for three and four-segment EA models it decreased by ~0.0129 *L* and ~0.0044 *L*, respectively. This shows that, while to begin with, increasing segment numbers results in large increases in accuracy, these accuracy increases become progressively smaller as segment numbers become large (e.g., the steps in Figure 3.1 become smaller for large numbers of segments). Additionally, while

automated segment models show greater increase in accuracy than equal-length segments, this is less prominent with larger numbers of segments.

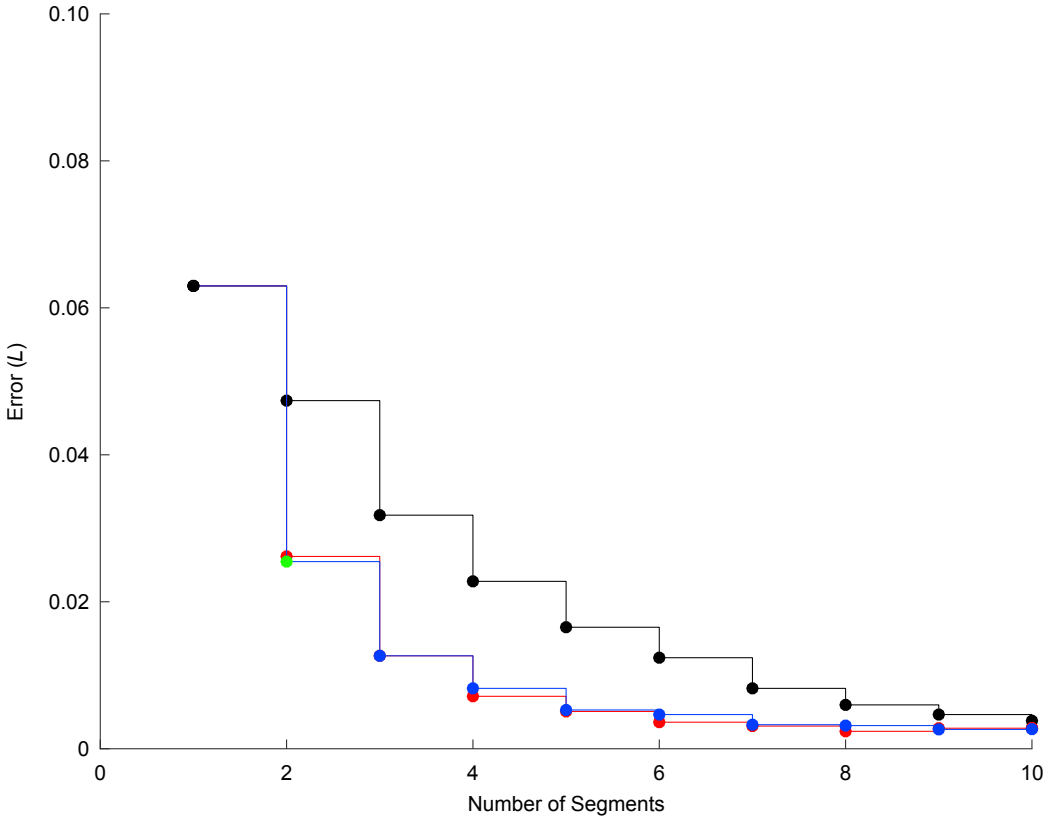


Figure 3.1: Overall model error versus increasing numbers of segments for equal segments (black) and automatic segments (red for segment growing approach, and blue for evolutionary algorithm approach). The top leftmost point shows the error for the single-segment model. In place of two-segment model for evolutionary algorithm, a single best joint is used (green). Note: for more information on how error values were obtained for specific segment numbers with segment growing approach, despite lack of user control over segment numbers with this algorithm, see subsection "General Methodology and Parameters Investigated" under Section 3. Error is presented in midline body lengths (L).

From another perspective, it could be said that automated segment models improve over equal-length segment models by achieving the same level of accuracy using fewer segments, as illustrated in Figure 3.2. Seven equal length segments are required to achieve less than ~0.01 *L* overall model error (resulting in OME of ~0.0082 *L*). However, only four segments were required for SG and EA segment

models to remain below the same OME threshold (resulting in OME of ~0.0071 *L* and ~0.0082 *L*, respectively). Thus, the uneven segment lengths used by the models generated from automatic segmentation approaches are more parsimonious (i.e., more accurate, whist using fewer segments) compared to equal length segments. Notably, when considering models using four equal length segments, accuracy deteriorated by almost three times compared to uneven length segment models.

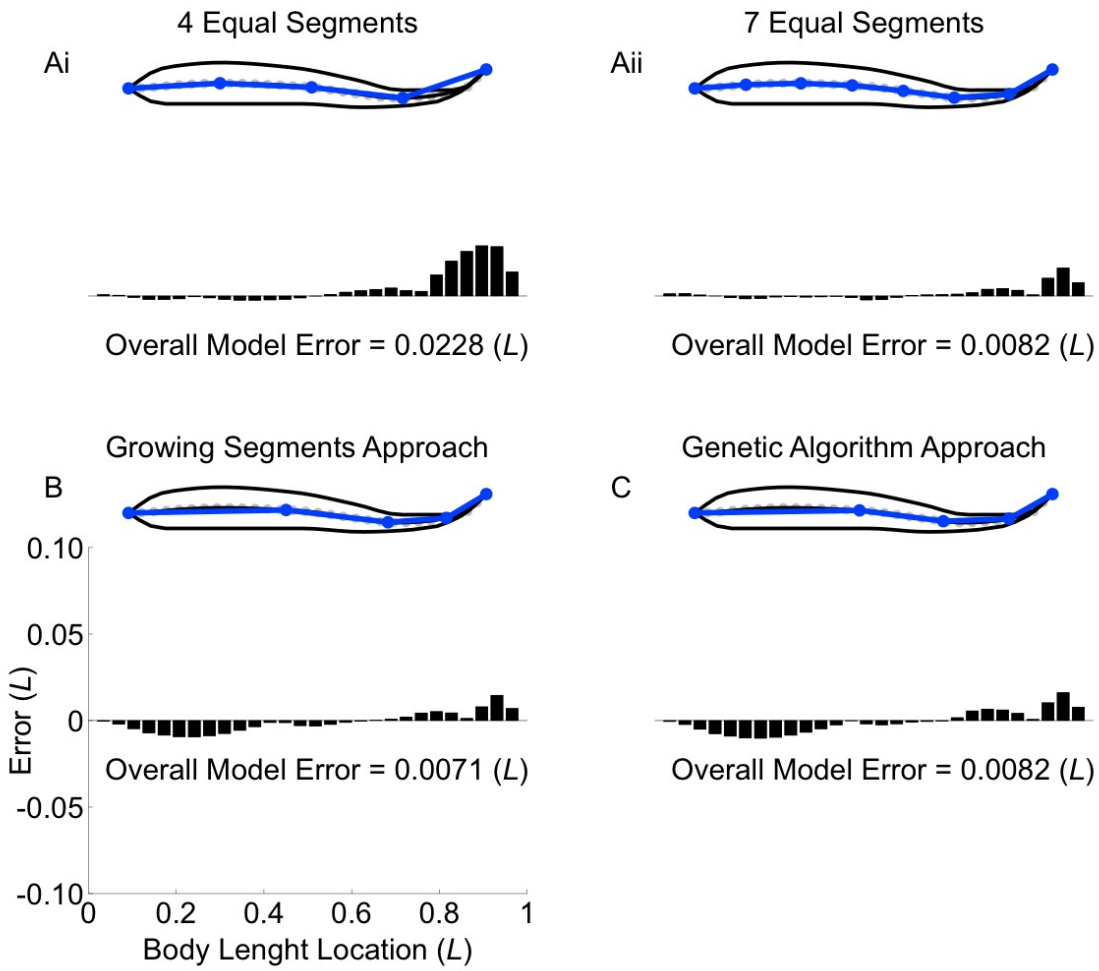


Figure 3.2: A snapshot comparing modelled (blue) versus actual midlines (black). Error bars indicate model error for each point along the midline. Four different models are shown: (Ai and Aii) four and seven-segment equal-length segment models. (B) four-segment model generated by segment growing approach. (C) four-segment model generated by evolutionary algorithm approach. Error values below error bars indicate error value for overall model error. Error is presented in midline body lengths (L).

By analysing instantaneous error at each midline point (i.e. midline point error), such as those illustrated as error bars in Figure 3.2, it can be seen why shorter segments are needed to accurately represent the posterior body. For each segment, the difference between the actual midline and the segment line is lowest at the joint locations, and increases gradually towards the middle of the segment, where the body curvature is highest. As the body bends much more at the tail than near the head, it is reasonable that segments are shorter towards the tail, as shorter segments can more accurately fit a higher curvature than longer segments.

With the four-segment equal-length segment model (Figure 3.2 Ai), for the first, second, third, and fourth segments, maximum average error of ~0.0033 *L*, ~0.0031 *L*, ~0.0066 *L*, and ~0.0218 *L* occurs at points 4, 13, 20, and 27, respectively. Thus, the error at the tail is noticeably greater than the threshold of 0.01 *L*. However, for the four-segment SG and EA models, the error distribution is much more even (Figure 3.2 B and C). For the SG model, the maximum average error is ~0.0059 *L* (first segment), ~0.0061 *L* (second segment), ~0.0063 *L* (third segment), and ~0.0071 *L* (fourth segment), occurring at points 8, 18, 24, and 28. Similarly, although slightly less evenly distributed, for the EA model, the maximum average error is ~0.0065 *L* (first segment), ~0.0040 *L* (second segment), ~0.0072 *L* (third segment), and ~0.0081 *L* (fourth segment), occurring at points 8, 17, 23, and 28, respectively. Thus, these overall more even and reduced error distributions can be attributed to the unevenly spaced joint locations used by the automatic segmentation approaches, with average error increasing and decreasing with segment length relative to body curvature.

Joint Locations

While equal-length segment models have equally spaced joint locations, the multi-segment models generated by SG and EA approaches use unevenly spaced joint locations. As segments need to be shorter towards the tail, for models with six or less segments, these joints are sparse towards the anterior, but dense towards the posterior, with most joints being located around two-thirds along the body length (2/3 *L*). For models with more than six segments, however, joint locations are slightly more evenly distributed, which may be related to segment lengths being less important for these segment numbers. See Figure 3.3 to see these trends illustrated.

Agreement Between Segmentation Approaches

Figure 3.3 shows that there is good agreement between models generated by the two automatic segmentation methods, SG and EA. When comparing the location of each joint in models generated by SG with the equivalent joints in the equivalent models generated by EA, the difference in joint locations between SG and EA is less than ~0.025 *L* mean absolute error (with standard deviation of ±~0.027 *L*). For the two automatic segmentation approaches, comparing each joint location of each model outputted by one approach to the location of the corresponding joint of the corresponding model outputted by the other approach, there is no statistical difference in joint locations between the two automatic approaches (p-value = ~0.06, Wilcoxon test). The null hypothesis is that the joints produced by both these approaches come from the same distribution. However, qualitatively there is a slight disagreement between the two approaches (in models with six or more segments), which may be related to slight problems with the evolutionary algorithm not properly collapsing on a single best solution (i.e., 10 generations may be insufficient).

Additionally, another factor that may have affected disagreement could be increasing numbers of locally optimal joint locations; as segments shorten, their joint locations have less impact on reducing error. Either way, these trends in strong agreement between these two algorithms appear to continue for the results presented in Section 3.2, Section 3.3, and Section 3.4, as well.

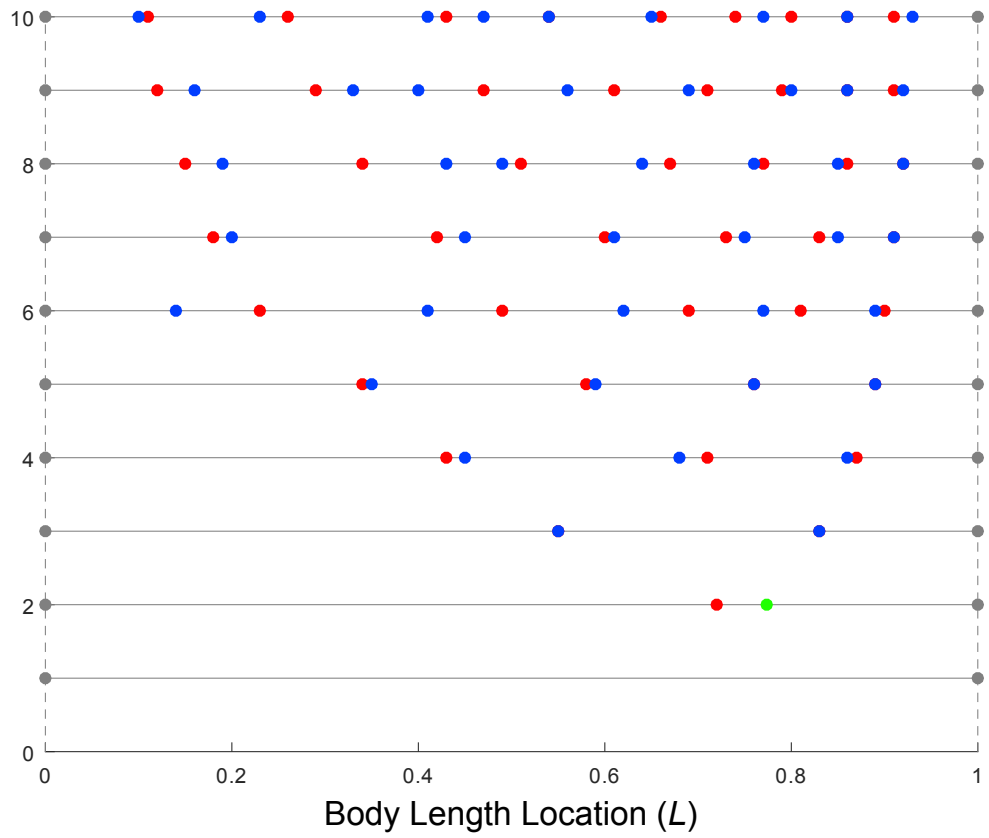


Figure 3.3: Joint locations for increasing numbers of segments generated by segment growing approach (red), evolutionary algorithm approach (blue), and single best joint (green). In cases where circles for different approaches overlap, this only indicates that these joints are in the same location, unless specified otherwise (e.g. that the approaches in question came to different conclusions regarding the number of joints needed). While joints more or less agree until five segments, beyond this they start to disagree. This may be due to the evolutionary algorithm approach not properly collapsing upon a single best solution beyond this number of segments, given the parameters values used (e.g. population size, generation number, etc.).

Furthermore, similarity between the automatic segmentation approaches (and improvement over equal length segments) can be seen by analysing a 'fitness

landscape' of joint locations versus accuracy. Figure 3.4 provides a visual illustration of a fitness landscape for possible three-segment models of a trout specimen. This figure was produced using a brute force method that linearly checked all possible joint locations, and the error they produce for this data set. The x and y axes are the possible locations of the joints connecting these segments, while the shading illustrates the accuracy of the model, with lighter being more accurate. Thus, accuracy of different models is illustrated by their position on the landscape and the shading at that position. As can be seen, both segment growing approach and evolutionary approach (red and blue cross, respectively, overlapping) lie at almost the same position, very close to the best combination of joints (black plus). Also, while equal-length segments (black cross) appear moderately accurate, both automatic segmentation approaches appear to be noticeable improvements. Although this is not necessarily conclusive regarding whether the automatic segmentation algorithms are guaranteed to find, in the general case, the best joints (or even find similar ones), if other fish and models did have similar fitness landscapes, then it could indicate that these algorithms may be likely to find best solutions in these cases.

Overall, close similarity between the joint locations outputted by these two automatic segmentation algorithms could be promising. It may indicate that this solution space is unimodal (i.e. there is only a global optimum with no local optimums), or that if local optimums exist they are more or less equivalent to the global optimum (although both these are speculative). Additionally, if all the best (i.e. satisfactory) solutions were similar, and either algorithm had a flaw that compromised their effectiveness at finding one of these best solutions, unless they had the same flaw,

then they may easily come to different conclusions. Likewise, as the results obtained indicate that they did come to similar conclusions, it may indicate that both algorithms are at least somewhat effective at finding satisfactory solutions.

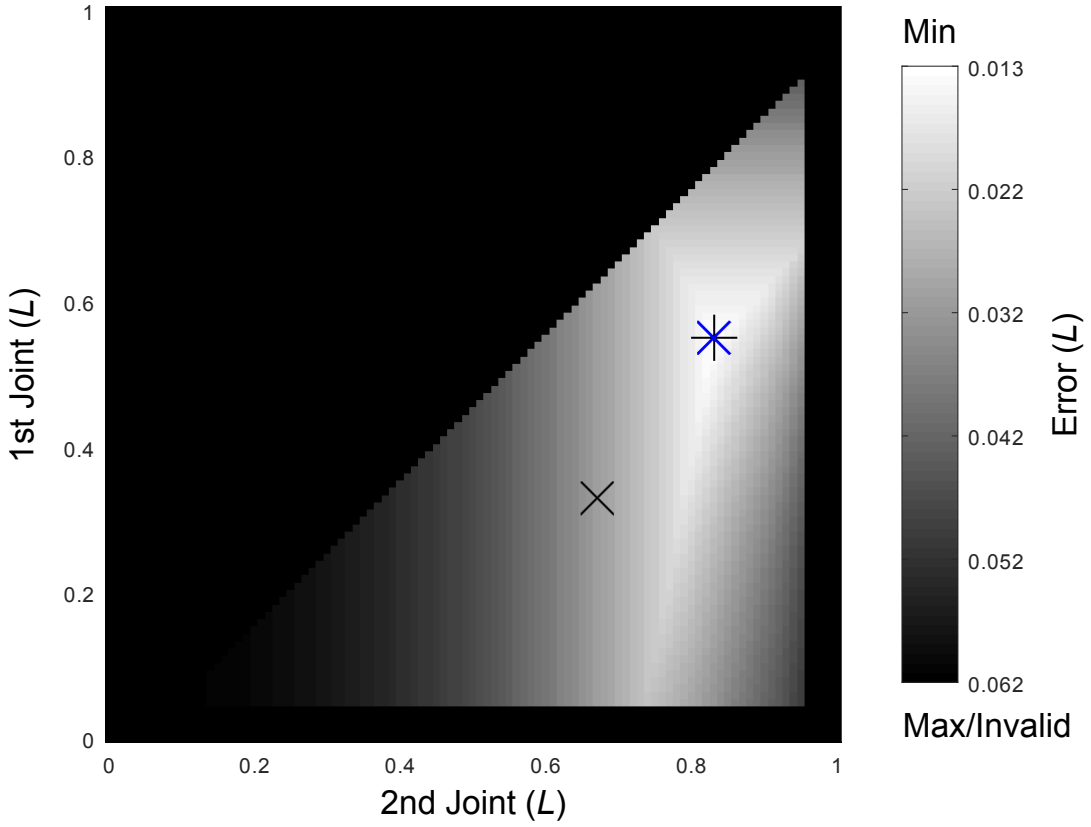


Figure 3.4: Visualisation of overall model error 'landscape' for a two-joint (three-segment) model of a trout specimen, generated using a brute force approach that searched all possible joint configurations. Each location on the landscape indicates the error of a different joint configuration: y and x axis locations indicate the locations of the first and second model joints, respectively. White areas indicate models with lowest error. Grey areas indicate models with higher error, with areas being darker as model error increases. Black areas indicate either highest error models (e.g. lower left corner) or invalid models (joints too close together, or 2nd joint being located before 1st joint). The model with the lowest error is marked with a black plus. The error produced when using equal length segments is marked with a black cross. The error produced by the lowest error models generated using segment growing approach and evolutionary algorithm approach are marked with red and blue crosses, respectively. Note that, in this case, as both the segment growing approach and evolutionary algorithm approach have generated models with almost the lowest error possible (given S_{min} and ΔS), their markers are located directly on top of each other, directly on top of the black plus marker indicating the lowest possible error model.

3.2: Species Comparison

Objective

In these tests, the goal was to find out how multi-segment models vary between different fish species. Do different species need different segment numbers? Do joint locations remain consistent or vary between species? Either way, where along the body are these joints located?

Method

Models were generated for each species using segment growing approach and evolutionary algorithm approach. For each species, the model used for the comparison was the model with the smallest number of segments that kept its overall model error below 0.01 *L*. The main attributes investigated when comparing models of different species were segment number, and joint locations. Along with specific joint locations, mean joint locations and standard deviation were also considered.

Datasets

Ten (sub)caragiform species were tested, one specimen per species, during steady swimming (see Table 4 and Figure 3.5). Fishes varied in length from 19.4 cm (knifefish) and 45.5 cm (jack), and swimming speed from ~1.0 $L\ s^{-1}$ (gar) and ~3.4 $L\ s^{-1}$ (trout). The trout data set was the same data set that was used in the previous analysis (Section 3.1).

Species Comparison Specimens		
Specimen ID (Species name (Latin name))	**Body Length (cm)**	**Swim Speed (L s^{-1})**
Florida gar (*Lepisosteus platyrhincus*)	36.9	~1.0
Northern barracuda (*Sphyraena borealis*)	33.0	~2.6
Clown knifefish (*Chitala ornata*)	19.4	~1.4
Rainbow trout (*Oncorhynchus mykiss*)*	23.0	~3.4
Sheepshead (*Archosargus probatocephalus*)	32.0	~1.8
Mangrove snapper (*Lutjanus griseus*)	23.0	~3.2
Indo-pacific tarpon (*Megalops cyprinoides*)	23.0	~2.7
Tomtate (*Haemulon aurolineatum*)	20.0	~1.0
Pinfish (*Lagodon rhomboides*)	28.0	~2.9
Crevalle jack (*Caranx hippos*)	45.5	~3.1

*Table 4: Specimens used in the species comparison tests, each belonging to a different species. Each of these specimens was recorded during the same swimming behaviour: steady swimming. *The Fish 1/trout data set used in the tests for equal vs automatic segments (Section 3.1).*

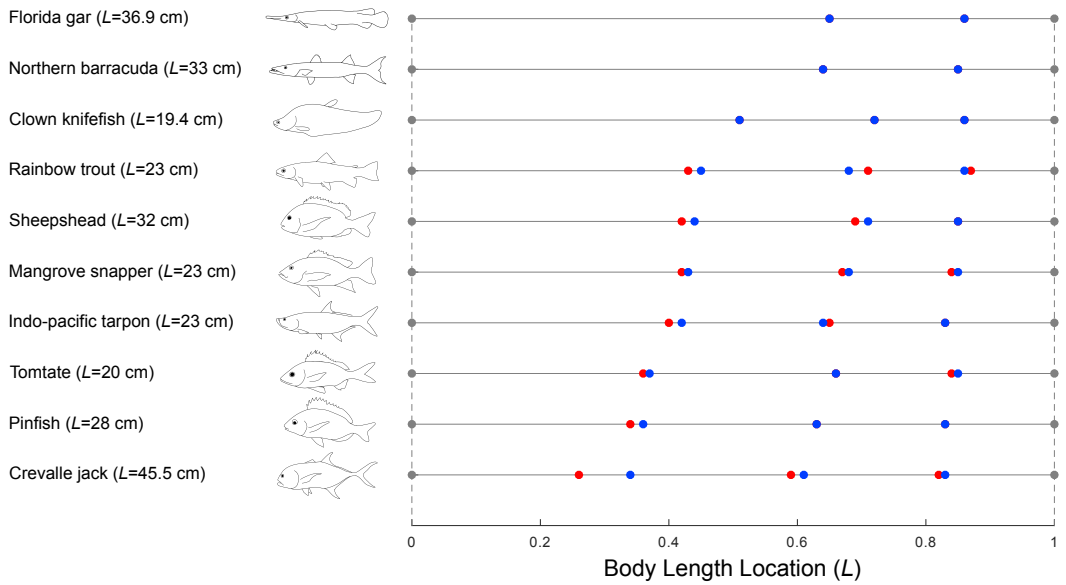


Figure 3.5: Joint locations for 10 different fish species, calculated using segment growing approach (red) and evolutionary algorithm approach (blue). In cases where circles for different approaches overlap, this only indicates that these joints are in the same location, unless specified otherwise (e.g. that the approaches in question came to different conclusions regarding the number of joints needed). The body shape of each species is illustrated on the left, as well as specimen length.

Results

Four and fewer segments are sufficient to model (sub)carangiform swimmers whilst maintaining less than 0.01 *L* overall model error. Additionally, there is noticeable variation in locations of joints in the anterior body between species, while all species exhibit very similar joint locations in the posterior body. Overall, joint locations were biased towards the posterior, continuing the trend of progressively shorter segment lengths seen before (Section 3.1). The results are examined in more detail in the text below.

Joint Numbers

Some variation in joint number was observed. As illustrated in Figure 3.5, for most species, only four segments (three joints) were necessary. However, for gar and barracuda this was reduced to three segments (two joints). Without considering joint locations, this difference in joint number would indicate differences in flexibility. It could indicate that gar and barracuda are less flexible than other species, or at least in specific areas. This result is consistent with the fact that both gar and barracuda display limited body bending in their anterior body.

Joint Locations

Joint locations appeared to noticeably vary between species, especially for the anterior body. The average location of the most anterior joints was ~0.44 *L* (SG) and ~0.46 *L* (EA) with standard deviations of ~0.12 *L* (SG) and ~0.11 *L* (EA). In contrast, the average most posterior joint location was ~0.85 *L* (SG) and ~0.85 *L* (EA) with standard deviations of ~0.02 *L* (SG) and ~0.01 *L* (EA). These illustrate that the most anterior joint locations congregate around 1/2 *L* along the body, while the most

posterior joint locations congregate around 4/5 *L* along the body towards the posterior. Interestingly though, they contrast in their distributions. Around where they congregate, the most anterior joint locations are widely distributed, while the most posterior joint locations are narrowly distributed, showing high variance and consistency between species respectively. These joint location averages and distributions are also apparent from observing the left and right most joints in Figure 3.5.

Further considering the extent of these variations in joint locations between species, the most anterior joint closest to the head was found in jack (0.26 *L* (SG) and 0.34 *L* (EA)), while the most anterior joint furthest from the head was found in gar (0.65 *L* (SG) and 0.65 *L* (EA)). In comparison, the most posterior joint closest to the tail tip was found in trout (0.87 *L* (SG) and 0.86 *L* (EA)), while the most posterior joint furthest from tail tip was found in jack (0.82 *L* (SG) and 0.83 *L* (EA)).

3.3: Swimming Behaviour Comparison

Objective

With these investigations, the goal was to explore how models vary between different swimming behaviours. Specifically, how do segment numbers and joint locations change between different behaviours?

Method

Models were generated for each behaviour using segment growing approach and evolutionary algorithm approach. For each behaviour, the model used for the comparison was the model with fewest segments and overall model error below 0.01 *L*. Overall, models for each behaviour were compared with those of every other behaviour (see Figure 3.6). The main attributes investigated in the comparisons were segment number and joint locations.

Datasets

In addition to the baseline steady swimming trout dataset from Section 3.1, three other behaviours from the same fish were examined: gliding, Kármán gaiting, and acceleration. Gliding (or coasting) involves straightening the body in an attempt to maintain momentum which has already been achieved, often through acceleration [48]. Kármán gaiting involves weaving between Kármán vortices shed periodically behind a bluff body to save energy [36]. These flow regimes (called Kármán vortex streets) are often used to analyse unsteady swimming in fishes. Acceleration involves using large, fast body undulations to increase swimming speed [49]. Example midlines for gliding, steady swimming, Kármán gaiting, and acceleration can be seen next to their respective best joints in Figure 3.6.

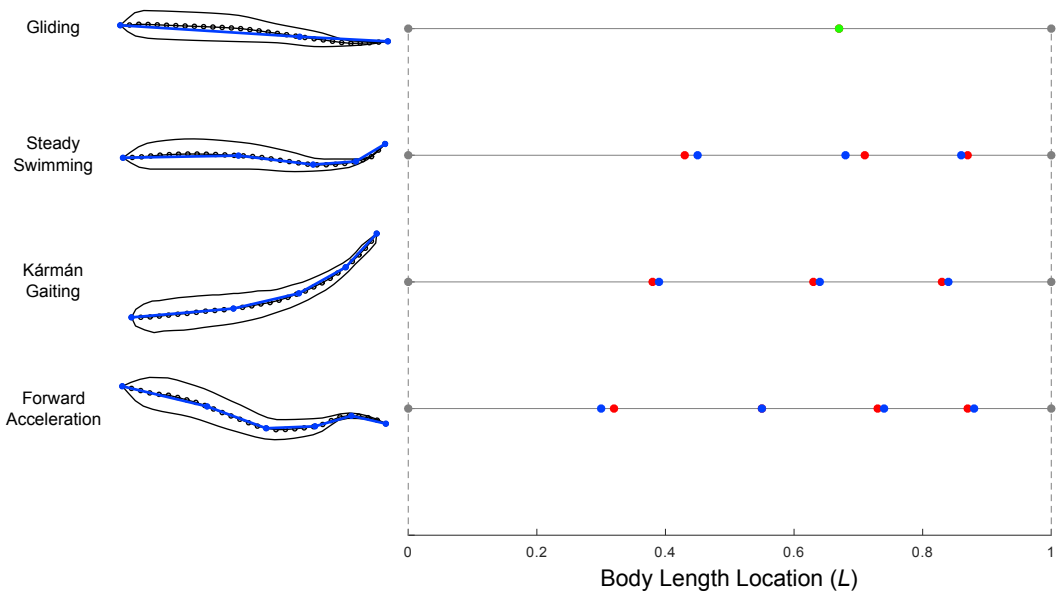


Figure 3.6: Comparison of joint locations versus different swimming behaviours for trout. (Red) segment growing approach. (Blue) evolutionary algorithm approach. (Green) single best joint. In cases where circles for different approaches overlap, this only indicates that these joints are in the same location, unless specified otherwise (e.g. that the approaches in question came to different conclusions regarding the number of joints needed).

Results

The number of segments required to accurately model different swimming behaviours appears to noticeably vary. To maintain overall model error just below 0.01 *L*, two segments (one joint) were required for gliding, four segments (three joints) were required for steady swimming and Kármán gaiting, and five segments (four joints) were required for acceleration.

Joint Numbers and Locations

Gliding required fewer joints than steady swimming. As illustrated in Figure 3.6, gliding only requires one joint, two less than steady swimming. However, this joint is situated in approximately the same general area as steady swimming joints, being located approximately 2/3 *L* towards the posterior.

Although the same number of segments (four segments) were needed for modelling both steady swimming and Kármán gaiting, there are noticeable differences in the best joint locations. During Kármán gaiting, joints shift slightly towards the head, resulting in more even distribution of joints along the body.

Acceleration required the maximum number of segments (five segments), one more than steady swimming and Kármán gaiting. Although joint locations are shifted towards the head for acceleration, they are still more densely grouped approaching the body posterior, continuing the trend of segments becoming progressively shorter moving towards the tail.

3.4: Swimming Speed Comparison

Objective

In these investigations, the goal was to examine whether models vary with increasing steady swimming speeds. Specifically, does segment number need to increase, or do joint locations shift along the body with increasing swimming speed?

Method

For each dataset, models were generated using segment growing approach and evolutionary algorithm approach, such that the models with the smallest number of segments that maintained overall model error below 0.01 *L* were used in the comparisons (see Figure 3.7).

Datasets

Five specimens of trout were tested. These ranged in length by 6.5 cm, the shortest and longest being 18.5 cm and 25 cm, respectively (see Table 5). Similar length specimens were chosen so that there would be less doubt over whether variations were caused by differences in specimen length. Five to eight speeds were tested for each specimen. In total, these ranged by ~3 L s^{-1}, with minimum and maximum speeds tested of ~0.8 L s^{-1} and ~3.9 L s^{-1}, respectively. These speeds fall within the natural swimming speeds observed in nature for these fish.

Swimming Speed Comparison Specimens	
Specimen ID	**Swim Speed (*L* s^{-1})**
Fish 1	~1.1
Fish 1	~1.6
Fish 1	~2.1
Fish 1	~2.5
Fish 1	~3.0
Fish 1*	~3.4
Fish 1	~3.9
Fish 2	~1.1
Fish 2	~1.6
Fish 2	~2.0
Fish 2	~2.5
Fish 2	~2.9
Fish 3	~1.1
Fish 3	~1.5
Fish 3	~1.9
Fish 3	~2.3
Fish 3	~2.7
Fish 3	~3.2
Fish 3	~3.6
Fish 4	~0.8
Fish 4	~1.1
Fish 4	~1.4
Fish 4	~1.7
Fish 4	~2.0
Fish 5	~0.8
Fish 5	~1.1
Fish 5	~1.4
Fish 5	~1.7
Fish 5	~1.9
Fish 5	~2.2
Fish 5	~2.5
Fish 5	~3.6

*Table 5: Rainbow trout (*Oncorhynchus mykiss*) specimens used in the speed comparison tests. The specimen lengths were: 23.0 cm (Fish 1), 23.5 cm (Fish 2), 25.0 cm (Fish 3), 18.5 cm (Fish 4), and 19.0 cm (Fish 5). Each was recorded during the same swimming behaviour: steady swimming. *The Fish 1/trout data set used in the tests for equal vs automatic segments (Section 3.1).*

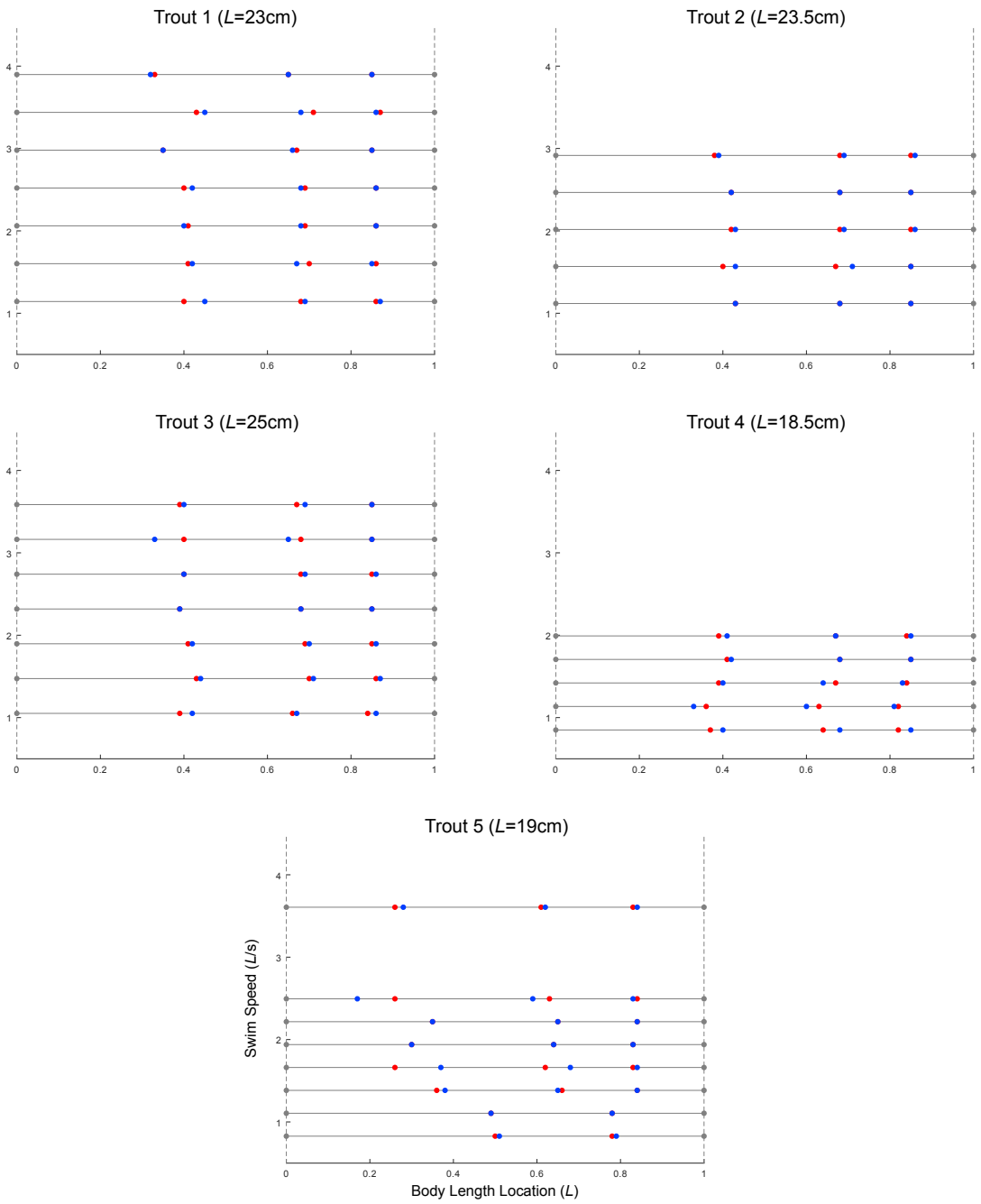


Figure 3.7: Comparison of joint locations for five trout specimens swimming at various speeds. (Red) segment growing approach. (Blue) evolutionary algorithm approach. In cases where circles for different approaches overlap, this only indicates that these joints are in the same location, unless specified otherwise (e.g. that the approaches in question came to different conclusions regarding the number of joints needed).

Results

It appears for rainbow trout during steady swimming, speed does not change the number of segments required for accurate modelling, with four segments being

sufficient to achieve overall model error below 0.01 *L*, although a few specimens at low swimming speeds required one less segment. Considering locations of joints, while the most posterior joint location remained fairly consistent, the most anterior joint showed considerably variation. While for a few there is a trend that joint locations shift anteriorly with increasing swimming speed, there appear to be several cases that contradict this trend. Also, there appears to be little to no trend of joint locations in relation to specimen length. These findings are examined in further detail below.

Joint Numbers and Locations

Joint number showed little to no variance between speeds and specimens. As can be seen in Figure 3.7, in nearly all cases a minimum of four segments (three joints) were necessary to meet a < 0.01 *L* error requirement. That said, for the two slowest speeds of Trout 5, this dropped to three segments (two joints).

The most posterior joint showed the most consistency, remaining between 0.8 *L* and 0.85 *L* in nearly all cases. In contrast, the most anterior joint showed the most variation. While for the two cases where a specimen only used two joints it tended to be close to 0.5 *L*, for cases that used three joints it ranged from just under 0.3 *L* to just over 0.4 *L*.

However, the most anterior joint locations in Figure 3.7 do not seem to exhibit a clear pattern. For Trout 5, the most anterior joint locations appear to shift towards the head with increasing speed. Additionally, Trout 1 and 2 may also show slight trends where the most anterior joint shifts towards the head with increasing speed. However, if

there is a trend for Trout 5, it appears to be quite noisy. Also, Trout 4 and 3 do not seem to exhibit a trend were the most anterior joint shifts towards the head with increasing speed.

Similarly, there did not seem to be any pattern in joint location variation associated with specimen length. It could be said that the shorter specimens (Trout 4 and 5) exhibited more variation in most anterior joint location than the longer ones, although Trout 1 did exhibit similar levels of variance at higher speeds. That said, if there were greater range in lengths of specimens tested, patterns between joint locations and length may have become more apparent.

4: Discussion

This section covers four topics: 1) discussion and analysis of the strengths, weaknesses, and potential improvements for the two automated segmentation algorithms and their experimental setup, 2) analysis and interpretation of the experimental results presented in the previous chapter, 3) potential application areas where the automatic segmentation methods can be applied to, and 4) limitations of the current work and potential improvements in the future.

4.1: Automated Segmentation Algorithm Design and Experimental Setup

Here, the design choices made for the segment growing and evolutionary algorithm approaches, and the strategy employed when using them to generate results, are discussed. This includes how design choices may have affected the outputs of the algorithms, and how the experimental setup may have influenced the experimental results. Also, potential improvements over these choices are discussed.

4.1.1: Segment Growing Approach

There are multiple advantages to the segment growing approach. Given the specified error threshold, it will likely find a best number of segments and lengths

necessary to meet this threshold while reducing segment count. It is also fairly fast and computationally cheap, especially when compared with the evolutionary algorithm approach.

However, this approach does have some limitations. A principal one is that it does not allow the user to specify beforehand the exact number of segments that should be used in the outputted model. Additionally, there are likely other areas where it could be improved upon.

Potential Directional Bias in Defining Segments

A potential flaw with this algorithm is the fact that it may be biased towards prioritising improvement of segments near the head over those near the tail. This is due to the fact that it linearly searches from head to tail for joint locations. This bias manifests in initial segment lengths being chosen regardless of positive or negative impact on subsequent segments, while options for subsequent length choices become increasingly dominated by prior ones. In other words, segments towards the tail may possibly be more accurate if their lengths were not restricted by segments already established towards the head. Furthermore, this may be hampered by the fact that it does not backtrack. Even if it would benefit globally, it will not change segment lengths once it determines a locally satisfactory one (i.e. long enough without violating the error threshold or minimum length). This is due to the fact that, as a greedy algorithm, it assumes a globally optimal solution can be reached through local optimisation, even if this is not actually the case.

However, running the algorithm in reverse (i.e. from tail to head) may help to determine whether it is biased towards segments nearer the head. This could highlight whether segments unfairly benefit from being defined before later ones, and hence lead to imbalanced models. For instance, suppose the algorithm was run twice, starting once from the head and once from the tail, and segment lengths were compared between runs. If the tail segment was longer when the algorithm started from the tail, it may imply that starting from the head does suppress the best length for this segment. Likewise, if the head segment was longer when the algorithm started from the head, it may imply that this does induce a bias towards improving segments near the head. And if there is no difference in length, the algorithm may not be biased in either direction, at least given the parameters and data set tested.

Another possible way of mitigating potential directional bias could be to change the order in which segments are defined. Suppose an alternative algorithm: starting with one segment spanning the whole midline, it repeatedly splits this and subsequent segments into two more accurate ones, until a desirable accuracy is reached. While this may be biased towards the first segments created, it would not necessarily be biased towards those located at either end of the midline. However, the downside is that, at least with this simple version, it is limited to generating models with odd numbers of joints (i.e. even numbers of segments). That said, as more advanced algorithms without these limitations may still exist, it would be worth investigating alternatives to linearly growing segments from one end of the midline.

Improvements Over Linearly Searching for Segment Lengths

Another potentially inefficient aspect of the segment growing approach is the strategy it uses for finding good segment lengths. Although it checks almost every length between the shortest and best, it may not actually be necessary to check all these intermediate lengths. Instead, a simple alternative could be to repeatedly split the search space in half. For the first segment, set the lower and upper bounds of the search space as the nose and tail tip, respectively. Beginning from the head, set the end of the segment to half-way between the lower and upper bounds. Then halve the search space, by moving the lower or upper bound to the end of the segment, if it is too short or long, respectively. Because anything longer or shorter would also be too short or long, in the former and latter case, assuming SSE always increases with segment length. This process is iteratively repeated, moving the segment end and then the bounds, based on the rules above. The search stops when the distance separating the bounds reaches a minimum (e.g., ΔS). Finally, either the lower or upper bound is selected as the best segment length, depending upon whether selecting the latter would violate the error threshold. This whole process is repeated for each subsequent segment, with the initial lower bound being set as the end of the previous segment. Ultimately, though, this search problem could be a variant of the general problem of searching an ordered linear array. Consequently, as existing algorithms may have successfully solved similar problems, it may be worth experimenting with applying them to this search space.

4.1.2: Evolutionary Algorithm Approach

There are several advantages to the evolutionary algorithm approach. Compared with the segment growing approach, the number of segments can be precisely controlled. This can allow determination of best segment lengths for models that have pre-defined specific segment numbers, rather than relying on trial and error. Additionally, it is not biased towards linear exploration from head to tail, instead essentially applying equal priority to all areas of the body in both directions.

However, there are notable limitations with the evolutionary algorithm. Not only with the implementation presented in this thesis, but also with using an evolutionary algorithm to find a solution for this particular problem. Regarding the implementation, in many cases, it may not be as optimised as other possible implementations of evolutionary algorithms. Likewise, while this implementation may appear to optimise segment length combinations, there is no proof that it definitely does.

Regarding a more state of the art design, several limitations of the evolutionary algorithm approach may stem from it being based too closely upon the somewhat archaic simple genetic algorithm. Since the SGA was originally proposed, several discoveries and advancements have improved over its design, highlighting limitations with the operator implementations it employs. Consequently, it would probably be worth improving the design of the evolutionary algorithm approach, by using a more state of the art structure, operators, and choice of parameter values.

Improved Chromosome Representation and Constraint Handling

While the SGA usually employs chromosomes consisting of binary strings (strings of zeros and ones), the representation used in this thesis employs strings of real-valued numbers. This is significant, as binary string representations have been shown to have limitations (see Section 2.3.2, "Chromosome Representation" for more details). However, the representation used in this thesis may have other limitations. For instance, it allows segments which go backwards along the midline, despite this being invalid. This is due to the method used for translating chromosomes into joint locations. The position of the gene within the chromosome determines which joint the value of that gene belongs to. The first gene determines the location for the joint closest to the head, the second gene determines that for the joint second closest to the head, and so on and so forth. However, if a gene occurs after another gene within the chromosome, but their value precedes the value of the other gene along the midline, the resultant segment goes backwards. One solution could be to use a different method for translating chromosomes into joint locations. As a possible alternative, the gene values could determine which joints their value belongs to. Thus, the gene with the smallest value is the joint closest to the head, the gene with the second smallest value is the second closest joint to the head, and so on and so forth. However, this introduces the problem that multiple chromosomes can represent the same solution, as the same set of joint locations can occur with different orders of genes. Additionally, it does not enforce other constraints, such as ensuring no joints are closer than the minimum segment length S_{min}.

Alternatively, constraint handling methods could be used to ensure these conditions regarding joint locations are met. Several approaches of constraint handling have

already been experimented with by researchers. Some of these include simply discarding invalid solutions, applying penalties to solutions depending on their invalidity, repairing invalid solutions, handling constraints when decoding solutions, and considering constraints and objectives separately [50]. [51] used two separate chromosome rank scores during parent selection, one based on fitness and one based on constraints violated. [52] handled constraints through encasing particle swarm optimisation processes within a P-system membrane algorithm. In the research presented in this thesis, a potentially effective form of constraint handing is used when generating the initial population. However, this is not the case during offspring generation. Instead of discarding invalid chromosomes, and generating new ones each time, better constraint handling methods may improve computational efficiency of the evolutionary algorithm.

Addressing Crossover Directional Bias and Lack of New Values

Regarding the crossover method used in this research, this is also not necessarily without flaws. Specifically, as only one crossover point is used to split parents, and it is always situated half-way along the chromosome, it may be prone to positional bias [46]. In other words, there may be a severe tendency to separate combinations of genes spread over both halves of the chromosome. On the other hand, there may also be an overwhelming tendency to keep together genes located in the same half of the chromosome. This is despite the potential effectiveness of gene combinations in the former case, or ineffectiveness in the latter. While problem spaces may exist where these biases are beneficial, it is not presently clear if they are beneficial for finding effective combinations of joint locations. That said, other methods may have different biases that are not necessarily better, such as uniform crossover. With this

method, each individual gene in the offspring is randomly selected from the equivalent in either parent, and the other offspring is the reverse of the first. However, the potentially unwanted bias with this is the tendency to create offspring consisting of almost 50% of each parent. This may reduce the chance of transmitting a larger portion of genes from individual parents, even if those genes are more effective grouped together than fragmented. That said, this bias is already present to an even more severe degree in the implementation presented in this thesis. Either way, to determine an appropriate crossover method for chromosomes representing joint locations, it may be worth investigating how the potential biases of each method positively or negatively impact finding better joint locations.

Another potential disadvantage of the implemented crossover method is that it does not create new gene values. It only shuffles existing ones, and relies on the mutation operator to create new values. A possible alternative is constructing offspring from gene values that are a combination of both parents. Examples include combining corresponding parent genes using a weighted sum of their values. However, this is limited by the possible gene values only covering the area between the parents. Consequently, offspring are likely to be more similar than the original parents. To offset this, another approach is to apply a similar strategy, but extend margins beyond each parent, outside the inter-parent area. This potentially balances offspring lying between their parents with ones lying outside this area [46]. Additionally, improvements to the crossover implementation may also depend on changes made to improve the solution representation (i.e. chromosome format). This could introduce additional constraints that affect suitability of potential crossover methods.

Improving Mutation with Gaussian Mutation Power or Self-Adaptive System

Also, the mutation operator implemented in this research may be fairly limited, particularly due to how mutation power (i.e. distance) affects joint locations. While before execution the mutation power M_p can be fixed to one of many different values, the value cannot change during the evolutionary process. Consequently, if M_p is too small or large, mutations may not explore enough, or jump over better joint locations. While good M_p values could be found with test data sets by testing different values, they may not be effective in all cases. However, instead of implementing a fixed value, one solution could be implementing mutation power as a probability distribution. That is, whenever a mutation is applied to a joint location, the shift distance could be randomly selected from a range of possible distances. Selecting a given distance would then depend on a probability distribution, such as gaussian distribution of probabilities. Furthermore, implementing a self-adapting system of mutation could improve performance at different stages of the evolutionary process. Different mutation powers may perform better at different stages of the search, with larger or smaller values possibly being better depending on whether exploration or exploitation (i.e. refinement of solutions) is preferable [46]. [53] experimented with adaptive mutation implemented for a differential evolutionary algorithm, comparing viability with other existing evolutionary algorithms. However, implementing an adaptive strategy may benefit from preliminary investigation into how best parameter values change during evolutionary searches, and in this case, when finding joint locations.

Excessive Selection Pressure, Population Size, and Low Generation Number

Regarding the experimental setup used to generate results data with the evolutionary algorithm, there may be room for improvement. For instance, it may not be effective to use a large population size, small number of generations, and high selection pressure. Instead, it may be better to use the reverse: a smaller population size, larger generation number, and reduced selection pressure. The reason for this is that, due to the fairly high selection pressure, diversity of chromosomes and hence solutions was lost fairly quickly (see Figure 2.8). This could be problematic, as these lost chromosomes may have still led to better solutions over time, given sufficient chance to evolve. Consequently, as the high selection pressure was partially due to the small generation number, it may be better to increase the generation number and relax selection pressure. Likewise, increasing generation number may require decreasing population size, as the runtime cost of the algorithm may require scarifying one over the other. Alternatively, both population size and generation number could be simultaneously large if the algorithm was run on more powerful hardware, or by increasing efficiency of the implementation.

However, the population size may still need reducing, as it may have made the evolutionary process somewhat inconsequential. This is possibly implied by sometimes close similarity between the best chromosome in the first and final generations (Figure 4.1 A). This could be because a global best chromosome already existed in the first generation, due to so many random chromosomes being created. In other words, a 'random search' was unintentionally performed in the first generation. As Figure 4.1 implies some evolution, it seems unlikely that the algorithm is purely failing to evolve chromosomes. Also, assuming the same situation occurred

in Figure 3.4, it implies a chromosome near the global best already existed in the first generation. This is problematic, as it may make the evolutionary process redundant (although it can still refine these best solutions, see Figure 4.1 B). That said, a smaller population will not necessarily improve performance, unless it can be effectively evolved by the algorithm structure, operators, and parameters.

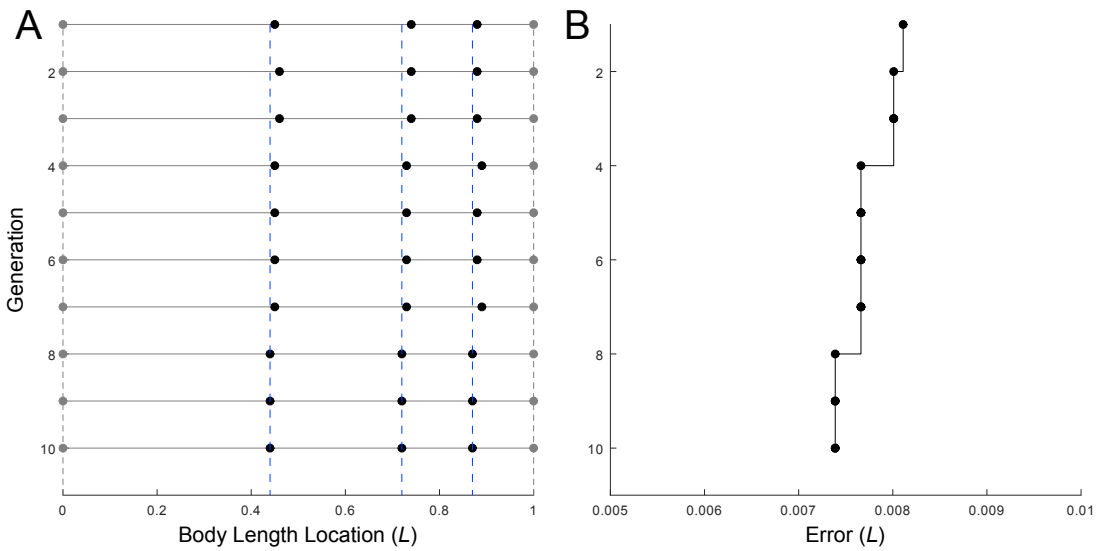


Figure 4.1: Illustration of how the best chromosome and its fitness (i.e. the best set of joint locations and the overall model error it produces) in the population changes over multiple generations. In this case, the 'best' chromosome in a population for one generation is the chromosome with the highest fitness. Both subfigures illustrate changes over the same 10 generations, the first being the initial population of random chromosomes, and the 10th being the final generation the algorithm produced. (A) Joint locations of the best chromosome for each generation. While there is only a little variation, joint locations do move closer to those of the best chromosome of the final generation, which also exhibits the highest fitness/lowest overall model error. The vertical dashed lines (blue) illustrate the distance of joint locations from those of the best chromosome in the 10th and final generation. (B) Overall model error of the best chromosome. While the variation is small (the x axis only spans a 0.005 L error range), there is a noticeable decrease in error for a few generations, and in general over all 10 generations.

In general, though, the overall lack of in-depth parameter tuning is a potential problem with the evolutionary algorithm experiments. While parameter values in other areas may appear adequate, closer examination could unearth unforeseen flaws, as well as better values. Additionally, having established potential flaws with

the parameter values for parent selection, population size and generation number, identifying better values may require further investigation. Consequently, parameter tuning is still an open area for this problem space, and further exploration and refinement of parameter values could be beneficial.

Need for Comparison of Multiple Runs

Additionally, another potential problem with the experimental setup was that only one run of the evolutionary algorithm was used per data set. The only exception was the wall-clock runtime analysis. In the other cases, if the assumption that the results remain unchanged between runs is actually false, then one run is unrepresentative of any variation. Even if there is actually no variation between runs, one run is probably still insufficient for determining this conclusively. Thus, future analysis may benefit from comparing multiple runs on each data set, to establish which of these possibilities is the actual case.

In Relation to Alternative Approaches to Midline Segmentation

Finally, for this particular problem, it may be overkill to employ an evolutionary algorithm. Alternative approaches may be capable of obtaining equally good results, but with reduced runtime and computational complexity. For instance, while the evolutionary algorithm is computationally more expensive than the segment growing approach, it does not guarantee better segment models or in less time. Additionally, while not necessarily a weakness, it is worth taking into account that the evolutionary algorithm is potentially less deterministic than alternatives. Results may vary between runs despite constant parameter values, due to the randomised elements of the algorithm, unless the random seed is also constant.

4.2: Analysing Experimental Results

Here, the implications of the results presented in this thesis are discussed. Before going into discussion specific to each of the four results subsections (Section 3.1, 3.2, 3.3, and 3.4), discussion focuses on covering general points relevant to all these results.

In general, the results presented here appear to indicate that, in most cases, five segments are sufficient for generating models with at least 99% accuracy (model error < 0.01 *L*). Additionally, in relation to joints locations, for best performance, segment lengths (and hence distance between joints) should decrease as they approach the tail, this trend being consistent in some way for all datasets tested. However, in contrast, few robots are designed in such a way. Limited research has demonstrated that optimising segment numbers and lengths using actual fish data could improve robot swimming performance [33]. Consequently, it would be desirable to expand upon such studies.

Regarding specific joints, the role of both the most anterior and posterior joint location appear to be crucial, but in different ways. In terms of the most anterior joint, selecting its location appears to require more consideration, as its best location appears to vary depending on different attributes (particularly on morphology, but potentially swimming behaviour and possibly speed as well). Consequently, this suggests that decisions on where to place anterior joints are more crucial for accurate modelling. In contrast, in terms of the most posterior joint, its best location appears to be consistent, independent of varying fish attributes. Thus, the fact that it

always congregates around just over 4/5 *L* along the body towards the posterior indicates that it may always be required at this location. With these characteristics of these joints in mind, further properties could be deduced. For instance, the consistency of the most posterior joint location may imply consistent body curvature between different fish beyond this location. Additionally, while the most anterior joint could be related to fish head morphology, the general higher variability of the more anterior joints may be useful for categorising and identifying different swimming models, and their associated fish attributes, such as species.

For steady swimming, kinematic characteristics of fishes are broadly divided into four categories: anguiliform, subcarangiform, carangiform, and thunniform, as defined by Breder Jr [54] and Lindsey [40]. However, recent research and data increasingly indicates that variations in kinematics between fish actually lie along a continuous spectrum, calling into question the validity of these rigid categories. In accordance with this trend, the results presented here may also support this theory; variation in joint locations between several species within the same category (here, (sub)carangiform swimmers are considered) would suggest that their swimming patterns are actually not so similar. In fact, particularly for the anterior body, joint locations appear to be distinct between each species. This could be due to differences in body shape, fins, bone structure (skull, ribs, pectoral girdle, etc.), and internal organs, each of which being responsible for physically limiting where and to what degree bodies can bend. However, the species data sets used here include specimens with different body lengths and swimming at different speeds. It is recognised that variations in size and speed may affect the segment lengths and

numbers in these models, and further experiments are needed to form a more comprehensive understanding of the diversity of (sub)carangiform swimming.

Having covered the more general points, the results more specific to each subsection are discussed in detail below.

4.2.1: Equal vs Automatic Segments

Regarding the results presented in Section 3.1, for nearly all segment numbers, joint locations appear to possibly be linked with fish body curvature. (Sub)carangiform swimmers, such as rainbow trout (the fish species used in this set of tests), exhibit body curvature that increases steadily towards the tail. This is in contrast with anguilliform swimmers, where curvature is fairly constant, and thunniform swimmers, where curvature sharply increases only at the more extreme posterior. However, the joint locations for the specimen used in this particular set of tests appear to nearly always congregate around 2/3 *L* along the body towards the posterior, show a moderately wide distribution, and steadily get shorter towards the tail. These segment lengths and their distribution appear to align fairly closely with the curvature pattern described for (sub)carangiform swimmers [54] [40]. This could indicate that the models generated using the automatic segmentation are actually responding to and representing some attributes inherent to body bending and fish morphology.

4.2.2: Species Comparison

Regarding the results discussed in Section 3.2, joint numbers and locations for the species comparison also appear to possibly be linked with body curvature, further supporting the potential conclusions draw from the Section 3.1 results. The fact that most joint locations congregate around two-thirds along the body could be linked with body movement and curvature increasing towards the posterior of many fish species. On the other hand, for the Section 3.2 results, the large variation in joint locations near this area of the body length may be linked with the variation in this trend for different species [55].

Specific joint locations may also be linked with these trends, while also providing further detail. The average most anterior and posterior joint locations, and particularly the fact that the most anterior joint location is further from the anterior than the most posterior joint location is from the posterior, further reflects the bias of more curvature towards the posterior. However, the large standard deviation for the most anterior joint location versus the small standard deviation for the most posterior joint location could be linked with how the anterior shows the most variation in movement and curvature between species [55]. Furthermore, based on the different patterns of these extreme anterior and posterior joint locations, it could be predicted that other joints might share one or the other of these characteristics, or a mixture of both, depending on whether they are near to the anterior or posterior. Additionally, the variation apparent in the most anterior joint location depending on the species of the fish could potentially be used to categorise and even identify models and their associated species.

These trends can also be observed to a certain extent with examples of specific species. In the case of gar and barracuda less joints were required than other species, implying overall more rigidity. They also required less joints towards the anterior and more towards the posterior. Both of these may be linked with these species exhibiting a more rigid anterior, with most body movement and curvature being focused in the far posterior [56]. In contrast, for jack and trout, more joints appeared towards the anterior, being more evenly distributed overall. This could be linked with both of these species exhibiting a more flexible anterior, with more evenly distributed body movements [57] [58].

However, some of these results also appear to disagree with biological observations. Joint locations for jack seem more biased towards the anterior than for trout, implying the former exhibits a more flexible anterior. However, biological observations conclude jack exhibits a more rigid anterior than trout, being classified as employing carangiform and sub-carangiform propulsive body movements respectively [58]. Additionally, some of the results in other sections may conflict with these results, depending on how representative they are. That said, these contradictory results may be due to the effects of different swimming speeds and body sizes.

4.2.3: Swimming Behaviour Comparison

Results for the behaviour comparison all seem to possibly be linked with the expected behaviour characteristics of their respective data sets. Still, there does appear to be some unexpected outcomes.

Gliding, as expected, required less joints than the other behaviours. This could be linked with noticeably reduced curvature observed with gliding due to virtually no body movement [48]. However, interestingly, gliding did still require one joint, located approximately 2/3 *L* along the body towards the posterior. This is somewhat unexpected, as gliding is predicted to employ a straight body posture with no undulation. Perhaps, at least in this case, the fish was exhibiting very slight undulation, or was statically curved throughout gliding, potentially in an attempt to turn slightly whilst performing this manoeuvre.

Kármán gaiting employed an identical number of joints compared with steady swimming, as well as similar joint locations. This reflects how, at first glance, Kármán gaiting shows similar body movements to steady swimming, at least in comparison with gliding and acceleration. However, Kármán gaiting exhibited joint locations slightly further towards the anterior than steady swimming. This could be due to Kármán gaiting being characterised by a signature weaving motion between vortices [36]. This may result in more of the body anterior being employed and exhibiting more curvature, which would explain the shift in joint locations towards this direction.

Unsurprisingly, acceleration required the largest number of joints out of the four behaviours, and slightly further shift in joint locations towards the anterior. This is likely due to the apparent increase in body curvature over steady swimming, resulting at least partially from increased body wave amplitude, decreased body wave length, and undulation increasing near the anterior [30] [49]. However, while the fairly noticeable parallel between the increase and shift towards the anterior for joint locations given the qualities of acceleration, and hence an increase in swimming speed, was expected, it contrasts somewhat with the fairly less definitive results regarding swimming speed for the similar speed comparison results.

4.2.4: Swimming Speed Comparison

Again, results for Section 3.4 agree with those for 3.1 with regards to congregating around two-thirds along the body towards the posterior, showing a moderately wide distribution, and steadily getting shorter towards the tail. Given that all fish specimens for this test were recorded during steady swimming, this may further reinforce the conclusion that these models are capturing attributes inherent to the fish themselves. Also, the fact that all these specimens were rainbow trout may further indicate that these attributes are inherent to their species as well.

However, the variation that does exist between the joint locations and numbers observed may or may not be related to variation in swimming speed, and could suggest that the results in other comparisons are less than definitive. Some of the results could be interpreted as pointing towards swimming speed having an effect on

joint locations and numbers. The fact that joints for Trout 1, 2 and 5 (particularly for Trout 5) do appear to shift somewhat towards the anterior, and the fact that for Trout 5 segment number increases from three to four as speed increases, imply curvature increasing towards the head and in general, respectively. Segment changes corresponding to kinematic changes is not implausible, as there is evidence that kinematics do change with swimming speed; tail beat frequency has been found to change with swimming speed [59]. However, these changes in joint location and segment number are not necessarily linked with changes in tail beat frequency. Furthermore, the fact that segment number only changes for Trout 5, that joints for Trout 3 and 4 show virtually no shift towards the anterior, and anterior joint shift for the other three specimens is only slight and could potentially just be a by-product of the general variation in joint locations between speeds, all call into question whether these differences in joint location and number are at all linked with swimming speed. Additionally, after applying linear regression to the first joint of each four-segment model versus the swimming speed of their respective midline, there does not appear to be a significant correlation between the two (p-value, Table 6).

Furthermore, the fact that there is generally slight yet noticeable variation in joint locations between the specimens and speeds calls some doubt upon whether, at the very least, the joint locations for trout specimens in other results are as definitive as they might appear, particularly for the species comparison. Assuming that the general variation in joint location observed with the speed comparison would also occur if different rainbow trout specimens were used in the species comparison, they may appear more or less similar to other species depending on the specimen used. If this general joint location variation was the case for the other species as well, this

would call doubt upon whether the models in their current state could be used to identify common values for particular species, as well as differences between different species. However, if these variations are due to speed, then ensuring that specimens tested in a species comparison all swam at equivalent speeds would be necessary, and potentially mitigate against the problems recently described. This in of itself encourages further tests to establish whether speed has an effect on joint locations and numbers.

Linear Regression - Joint 1 (Most Anterior) vs Swimming Speed						
Specimen	**Segment Growing**			**Evolutionary Algorithm**		
	Slope	**P-value**	**R^2**	**Slope**	**P-value**	**R^2**
Fish 1	-0.0179	0.2619	0.2422	-0.0296	0.1609	0.3512
Fish 2	-0.0178	0.2522	0.4000	-0.0200	0.0880	0.6750
Fish 3	-0.0059	0.4154	0.1361	-0.0254	0.1074	0.4344
Fish 4	0.0315	0.1614	0.5329	0.0385	0.4042	0.2382
Fish 5	-0.0306	0.2954	0.2656	-0.0549	0.2576	0.3032

Table 6: Linear regression results for comparing first joint (most anterior joint) with swimming speed, for four-segment models of the five trout specimens. Slope is the calculated slope of the regression line, p-value is the p-value for the regression, and R^2 is the r-squared value.

4.3: Potential Application Areas

In addition to discussing the implications of the results from applying the automatic segmentation algorithms to actual fish data, it is worth discussing the potential applications of these automatic segmentation algorithms and the models they produce. Specifically, this discussion here focuses on potential applications from engineering and biological perspectives. There are potentially several areas where

the research presented here may be relevant. Below, some of these application domains, and in what way these automatic segmentation approaches and associated research may be applicable, are discussed.

4.3.1: Robot Design

A significant area where segmentation of fish bodies is particularly relevant is robotics. In many cases, instead of designing a completely flexible body, roboticists chose to build a segmented robot instead. In addition to other issues such as complexity, a solid reason for choosing at least two body segments is that real fish usually consists of a rigid head and skull, but a flexible body and tail. However, the chosen number of segments and locations of their joints often vary between different robots.

Figure 4.2 shows the locations of joints for 12 robots developed over the past 20 years. While there is much variation, some trends can be seen. For instance, for every robot the most anterior joint is between 0.3 *L* and 0.6 *L* along the body. This is fairly consistent with what was found with the automatic segmentation approaches. Also, the maximum number of segments used is five, while the minimum is two, although every two-segment robot has a flexible posterior, while at least one five-segment robot also has a flexible final segment. Results for automatic segmentation would indicate that, for the five-segment robots, segments (and motors) used could be reduced while still maintaining accuracy. However, this is assuming that they are only steady swimming; some robots [23] do explore other

behaviours, such as turning, obstacle avoidance, and wall following. Similarly, in cases with less than four segments, increasing segments could increase accuracy, though this may be unnecessary if they already use flexible segments. Regarding joint locations, while most robots exhibit more or less evenly spaced joints, automatic segmentation appears to suggest that they would benefit from ones spaced by progressively decreasing distances. However, locations of most posterior joints in cases without a large flexible segment are fairly consistent with the automatic segmentation.

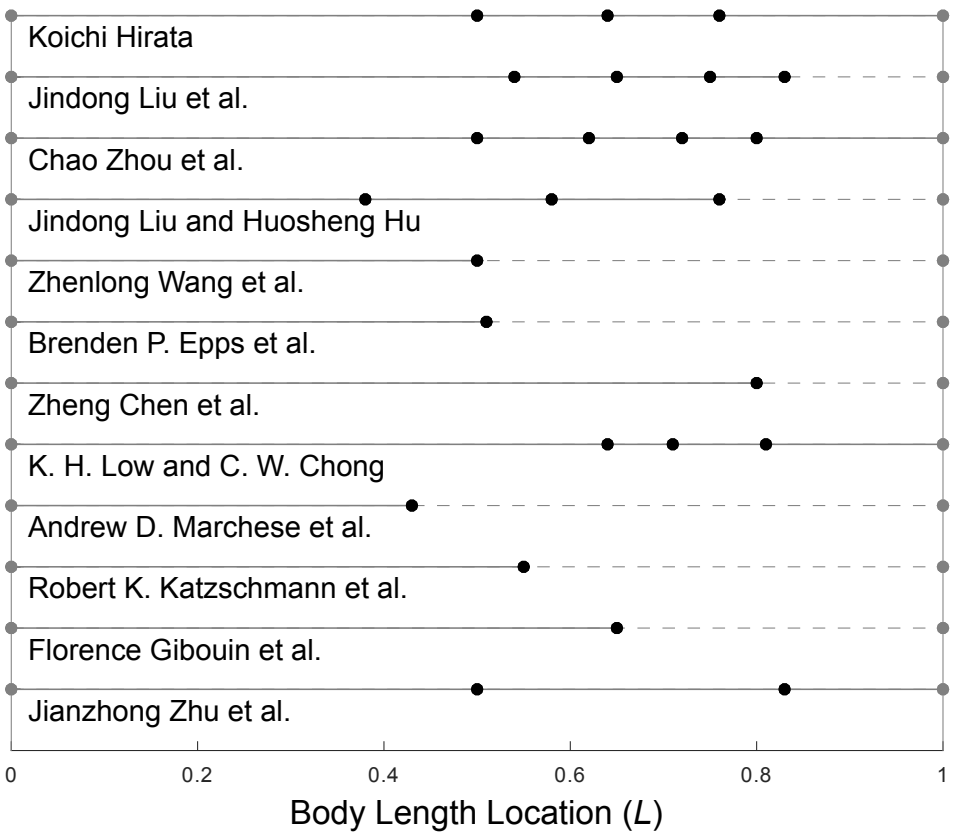


Figure 4.2: Physical joint locations for 12 robots build over the past 20 years. Each robot is composed of either entirely rigid segments (solid lines), or a mixture of these and flexible segments (dashed lines).

4.3.2: Robot Control

After determining for a given robot the ideal number of segments and the locations of their joints, the next question is how to actuate these segments so the robot movement accurately mimics the actual fish. During the research presented here, an initial attempt was made to address this problem for steady swimming behaviour. Although there was a focus on distinct segment movements, when these segment oscillations are combined, they are capable of generating waveform motions over the entire body.

A principle assumption made was that each segment has one degree of freedom: rotating around a single point over a two-dimensional plane. For each segment, this point is located at one end of that segment. This rotation is referred to hereinafter as pitch. For steady swimming, pitch movement is described using a sine equation:

$$\theta_i = \alpha_i * \sin(\omega t + \varphi_i) \quad (7)$$

where α_i and φ_i are the pitch amplitude and phase (timing), respectively. i is the index of the segment, indicating its position within the set of segments constituting the model, and ranges from 1 to n from head to tail, n being the total number of segments. t is the time stamp, and $\omega = 2\pi f$, where f is the tail beat frequency. f is determined through measuring the movements of the caudal fin over several tail beats. Then, for each segment, α and φ are determined through aligning that segment with the midline over many frames for several tail beats. From this, α is

determined from maximum angles experienced by the segment, and φ is determined from its angle relative to the current time frame and the other segments.

The combined kinematics of the whole n-segment model is then represented using $2n + 1$ movement parameters, $[f, \alpha_1, \varphi_1, \alpha_2, \varphi_2, \ldots, \alpha_{n-1}, \varphi_{n-1}, \alpha_n, \varphi_n]$, and n joint locations along the body, $[r_1, r_2, \ldots, r_{n-1}, r_n]$, defining the point of rotation for each segment. Additionally, segments are divided into two groups: the head segment (S_1) and the body segments ($S_2 \ldots S_n$). With this configuration, both the head segment and most anterior body segment rotate around the same joint location ($r_1 = r_2$), but oriented in opposite directions. Timing across data sets is standardised using the phase of the head segment, such that phase values of other segments are defined relative to this value: $[f, \alpha_1, 0, \alpha_2, \varphi_2 - \varphi_1, \ldots, \alpha_{n-1}, \varphi_{n-1} - \varphi_1, \alpha_n, \varphi_n - \varphi_1]$. These values, the locations on the model segments where they are applied to, and the way they change segment movement over time, are illustrated in Figure 4.3.

The fact that this model can translate actual movements into target angles and timings to achieve could be useful for roboticists. As discussed above, one way past robots have been articulated is by basing their movements on a travelling wave equation [11] [60]. However, the automatic segmentation approaches presented here could improve on this by determining segment configuration from the midline data itself, without needing to calculate a traveling wave equation first.

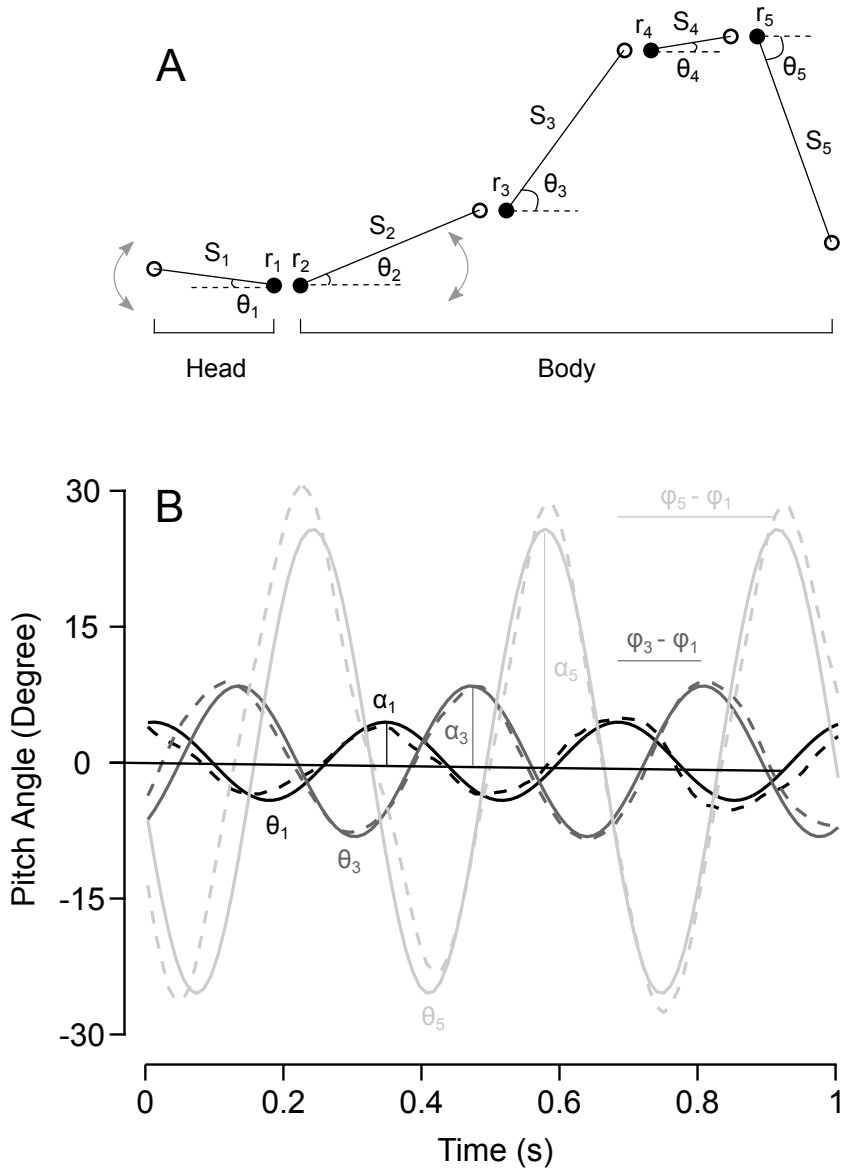


Figure 4.3: Animating segments using pitch angles and movement phases. (A) An illustration of the movement parameters of a five-segment model, with the head (S_1) and body segments (S_2-S_4) marked. Each segment (black lines) rotates around a joint (filled circles). Each segment end is indicated by an empty circle. (B) Changing segment pitch angles for segments S_1, S_3 and S_5 over three tail beat cycles. (Dashed lines) the measurements taken from actual fish. (Solid lines) the sine wave approximations used to control segment pitch. The sine wave parameters, pitch amplitude, and the movement phase timings relative to the head segment are also shown.

4.3.3: Muscle Activation

Biologists have long been interested in understanding the role of muscles in swimming. In general, fish employ two types of muscle: red muscles and white muscles. Measurements of muscle activation are made using electromyographs (EMGs) [61]. Scientists insert electrodes into muscles to measure their activation patterns. It is generally thought that red muscles are aerobic and have high stamina, and are used for steady prolonged swimming. In contrast, white muscles are anaerobic, producing greater force faster but having lower stamina, and are used for fast movements such as escape manoeuvres [62] [61]. Also, in addition to flexing fish bodies, muscles can control body stiffness to aid in propulsion [63] [64]. However, what is less certain is how muscles along the body make a concerted contribution to propulsion. That is, how are their activations coordinated to achieve propulsive swimming patterns.

It can be difficult identifying where and when specific muscles should activate, as a fish body is essentially one large dense piece of distributed muscle architecture. For instance, experiments indicated EMGs for single myotomes overlapped with over 20 neighbouring myotomes on the same side of the body for some species [61]. Often, scientists place electrodes using trial and error based on the fish morphology. However, as the models presented in this thesis highlight intense bending locations, these or their surroundings could also correspond to where muscles should activate. Additionally, these locations could aid guiding optimal placement of electrodes.

4.3.4: Morphology and Physical Properties of The Body

It is worth considering that intense bending locations can also correspond to intense structural stress locations. Given a plastic ruler, although it is capable of bending, increased bending leads to increased stress, until it ultimately overwhelms its structural integrity and breaks. How then do fish exhibit intense flexibility while maintaining structural integrity? Hypothetically, some morphological adaptations may be responsible for sustaining structural integrity and enduring intense stresses, although this is purely speculative. Fish bodies are highly three dimensional and complex, including muscles, tendons, skin and bones, all being intertwined [65]. As such, there are many structural candidates to investigate. Ultimately, further experiments are required to understand fish bodies and bending capabilities. That said, intense bending (and potentially stress) locations highlighted by the models presented in this thesis could help pinpoint where to investigate first.

4.3.5: Hydrodynamics

To obtain realistic estimations of forces applied to fish bodies and surrounding water from flow velocities and pressures, better understanding of fish-fluid interactions is needed. There are two main methods of measuring these interactions: Digital Particle Image Velocimetry (DPIV), and computational fluid dynamics. The former relies on tracking physical particles introduced into real water flows [66], while the latter attempts to simulate fluid movement using purely mathematical means [67]. In addition, if live fish are unsuitable, these techniques need a simulation of fish body

movements. Often these movements are generated using a travelling wave equation.

However, the automatic methods presented here provide an alternative systematic, and potentially more versatile, approach to simulating body movements. Not only can they allow a greater range of movements, but through controlling the number of segments used, they can allow experimenters to decide how simple or complex they want these body movements to be. This can also be used to evaluate the advantages of fish design.

4.3.6: Swimming Theory

Another significant area of interest to biologists is swimming theory. That is, understanding how fish kinematics and morphology interact with the surrounding water lead specifically to propulsion. Lighthill investigated fish propulsion through attempting to predict immediate reactive forces between fish and water for fish undulations of arbitrary amplitude, both regular or irregular [68]. Additionally, Taylor attempted to better understand propulsive forces generated by elongated, slender animals through conceptualising a body as an elongated flexible cylinder interacting with surrounding water. Effects of the surrounding water velocity, the cylinder velocity through this water, its movement as a wave of variable amplitude, wavelength and frequency, and resulting cylinder-water interactions were calculated and investigated [69]. However, similar to hydrodynamic techniques, the methods presented in this thesis could be used both as an alternative way of representing

body movements, and as a way of allowing investigation into simple or complex models.

4.3.7: Artificial Life

Movement models are also relevant in other media. Models in films, television, video games, and other media all need ways of ensuring their movement is realistic. One such example is a mobile app, 'Robobalik', that has been in development concurrently with the research presented here.

Robobalik is an educational game about how fish swim and aquatic life, designed for mobile touchscreen devices. Users are tasked with guiding their chosen fish out of a seaweed maze whilst avoiding hazards and collecting food. Some of the core aspects of Robobalik are illustrated in Figure 4.4.

Figure 4.4: A level from Robobalik. The user must navigate their virtual fish out of a seaweed maze, eat food to maintain energy and health and increase their score, and avoid health loss from touching jellyfish stings. The user controllable virtual fish (grey) is located in the figure centre. Algae (dark green), seaweed (yellow), and jellyfish (pale blue) are located nearby. The health meter (light green) and the energy meter (light blue) are located in at the top right and left respectively, either side of the score.

Initially, body movements of each fish were modelled using a travelling wave equation. However, while this can model steady swimming, it cannot model other manoeuvres such as c-start turning. In comparison, the multi-segment models presented here could allow representation of a far wider range of behaviours in Robobalik. In addition, the ability to vary the number of segments used could help developers find a balance between computational complexity and visual accuracy.

4.4: Limitations and Future Work

Although a primary aim of this research was to explore a broad range of scenarios, there are some flaws with the strategy used and areas that were somewhat overlooked. Thus, it is worth considering how these factors could be improved when conducting future research.

Regarding the automatic segmentation approaches themselves, it would appear that there is room for improvement, and aspects that could do with amending. For the segment growing approach, it appears that it would be worth investigating starting the algorithm from different ends of the body, to see whether there is a bias towards some segments depending on which direction the algorithm is running, and whether this significantly changes segment error. Also, it may be worth investigating other methods of growing these segments that does not rely on linearly checking all joints between the start and the best joint location. For the evolutionary algorithm approach, several areas could do with addressing. Firstly, it would appear that selection pressure, generation number, and possibly population size need adjusting, to reduce potentially detrimental fast loss of population diversity. Additionally, it may be worth investigating alternative crossover methods, as that which was employed in this thesis may not be the best suited to this problem area in part due to its tendency towards positional bias, as well as its potential limitation of only shuffling and not creating new gene values. Mutation could also do with further investigation, as implementations of parameters such as mutation power may be limited. Also, it may be worth searching for alternative chromosome representations and investigating

better constraint handling, so that the evolutionary process can better handle what constitutes a valid set of joint locations.

While the automatic segmentation algorithms and approaches developed during this research appear to perform well with the datasets tested, it is worth considering whether they could still function with less ideal datasets. It may well be the case that there is a limit on how well they can function given lower frame rates, fewer tail beat cycles, and fewer midline points. What are the minimum values for these properties necessary for the algorithms in their current form, and what are the minimum values necessary given the most robust algorithm possible? Given these and similar questions, it would be worth performing further tests using a range of sparser datasets to determine current algorithm robustness and how it could be improved.

Although the research presented here indicates that there may be merit in optimising segment numbers and lengths, this does not necessarily explain how these properties would actually affect the performance of a robot fish. Performance is often measured using swimming speed and power efficiency. Could a four-segment robot swim just as fast as a five-segment robot without sacrificing more energy? If not, would altering the lengths of these segments to match patterns similar to the models outputted by the automatic segmentation algorithms really mitigate this deficiency? With questions like these in mind, further systematic experimentation by constructing robots with different numbers and lengths of segments, and testing them under consistent conditions, could be very useful. Alternatively, if constructing this number and complexity of robots is not sufficiently feasible, it might be worth replicating this

with a virtual swimmer, saving some time and resources while also allowing (albeit potentially more simplified) study of hydrodynamic effects.

Also, while the results presented here regarding attributes such as species and behaviour are interesting, there is still doubt as to whether they actually result from these attributes, or from qualities specific to the individual specimens used in the experiments. In several situations, only one specimen was used per attribute case. If multiple specimens with the same attribute produced inconsistent results, it would imply that these results are not due to this attribute. On the other hand, consistent results between specimens would imply that at least they are less likely to result from qualities specific to individual specimens. Consequently, it would be worth significantly increasing the number of specimens used for further tests and evaluation, to perhaps at least five per experiment.

5: Conclusion

The principal aim of this research was to explore automatic generation of accurate and concise multi-segment models of fish kinematics. Two algorithms have been designed and implemented for this purpose: a segment growing approach, and an evolutionary algorithm approach. Extensive tests were made on these algorithms using a variety of fish data sets spanning ten species, four swimming behaviours, and multiple swimming speeds. Results found models generated using these algorithms to noticeably reduce segment number and error over equal-length segment models, and segment numbers and joint locations responded in specific ways to species, swimming behaviour, and potentially swimming speed. Discussion explored the implications of these results regarding potential relationships between model and fish attributes. Furthermore, discussion investigated how these algorithms can further biological and robotics research, including robot design, muscle activation, morphology, and swimming theory, as well as artificial life in media. In future, further endeavours include additional investigation of algorithm robustness with sparse datasets, testing the models generated by these algorithms using physical robots and fish kinematic simulations, and applying these algorithms to many more fish specimens and species.

Acknowledgements

The Author Would Like to Thank:

Dr. Otar Akanyeti

Dr. Christine Zarges

Dr. Inaki Rano

Sean Fetherstonhaugh

Sue Walker

Martin Nelmes

Dr. Alexandros Giagkos

Allison Zwarycz

Rowan Ashbrooke

Daniel Bloor

Melissa O'Reilly

James Bradley Strong

Arshad Sher

Asif Khan

This work is supported by Knowledge Economy Skills Scholarship to OA and SEAWF. This grant was provided by European Social Funds through the Welsh Government.

This Manuscript is Dedicated

in Loving Memory to

Cog

References

[1] B. Y. R. Bainbridge, “Caudal Fin and Body Movement in the Propulsion of some Fish,” *J. Exp. Biol.*, vol. 40, no. 1, pp. 23–56, 1963.

[2] L. Rosenberger, “Pectoral fin locomotion in batoid fishes: Undulation versus oscillation,” *J. Exp. Biol.*, vol. 204, pp. 379–394, Feb. 2001.

[3] C. C. Eriksen *et al.*, “Seaglider: a long-range autonomous underwater vehicle for oceanographic research,” *IEEE J. Ocean. Eng.*, vol. 26, no. 4, pp. 424–436, 2001.

[4] D. Ribas, N. Palomeras, P. Ridao, M. Carreras, and A. Mallios, “Girona 500 AUV: From Survey to Intervention,” *IEEE/ASME Trans. Mechatronics*, vol. 17, no. 1, pp. 46–53, Feb. 2011.

[5] J. H. Long Jr and K. S. Nipper, “The Importance of Body Stiffness in Undulatory Propulsion,” *Am. Zool.*, vol. 36, no. 6, pp. 678–694, Dec. 1996.

[6] J. L. Tangorra, G. V. Lauder, I. W. Hunter, R. Mittal, P. G. A. Madden, and M. Bozkurttas, “The effect of fin ray flexural rigidity on the propulsive forces generated by a biorobotic fish pectoral fin,” *J. Exp. Biol.*, vol. 213, no. 23, pp. 4043–4054, Dec. 2010.

[7] B. P. Epps, P. Valdivia y Alvarado, K. Youcef-Toumi, and A. H. Techet,

"Swimming performance of a biomimetic compliant fish-like robot," *Exp. Fluids*, vol. 47, no. 6, pp. 927–939, Dec. 2009.

[8] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model," *Science*, vol. 315, no. 5817, pp. 1416–1420, Mar. 2007.

[9] A. Raj and A. Thakur, "Fish-inspired robots: design, sensing, actuation, and autonomy—a review of research," *Bioinspir. Biomim.*, vol. 11, no. 3, 2016.

[10] A. G. P. Kottapalli, M. Asadnia, J. M. Miao, G. Barbastathis, and M. S. Triantafyllou, "A flexible liquid crystal polymer MEMS pressure sensor array for fish-like underwater sensing," *Smart Mater. Struct.*, vol. 21, no. 11, 2012.

[11] J. Liu, I. Dukes, and H. Hu, "Novel mechatronics design for a robotic fish," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 807–812.

[12] C. Rossi, J. Colorado, W. Coral, and A. Barrientos, "Bending continuous structures with SMAs: a novel robotic fish design," *Bioinspir. Biomim.*, vol. 6, no. 4, Dec. 2011.

[13] R. K. Katzschmann, A. D. Marchese, and D. Rus, "Hydraulic Autonomous Soft Robotic Fish for 3D Swimming," *Exp. Robot.*, pp. 405–420, 2016.

[14] T. Li *et al.*, "Fast-moving soft electronic fish," *Sci. Adv.*, vol. 3, no. 4, Apr. 2017.

[15] R. K. Katzschmann, J. DelPreto, R. MacCurdy, and D. Rus, “Exploration of underwater life with an acoustically controlled soft robotic fish,” *Sci. Robot.*, vol. 3, no. 16, Mar. 2018.

[16] A. D. Marchese, C. D. Onal, and D. Rus, “Autonomous Soft Robotic Fish Capable of Escape Maneuvers Using Fluidic Elastomer Actuators,” *Soft Robot.*, vol. 1, no. 1, pp. 75–87, Mar. 2014.

[17] A. Crespi, D. Lachat, A. Pasquier, and A. J. Ijspeert, “Controlling swimming and crawling in a fish robot using a central pattern generator,” *Auton. Robots*, vol. 25, no. 1–2, pp. 3–13, Aug. 2008.

[18] N. Kato and T. Inaba, “Guidance and control of fish robot with apparatus of pectoral fin motion,” in *Proceedings - 1998 IEEE International Conference on Robotics and Automation*, 1998, vol. 1, pp. 446–451.

[19] B. Dean and B. Bhushan, “Shark-skin surfaces for fluid-drag reduction in turbulent flow: a review,” *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 368, no. 1929, pp. 4775–4806, Oct. 2010.

[20] G. V. Lauder, D. K. Wainwright, A. G. Domel, J. C. Weaver, L. Wen, and K. Bertoldi, “Structure, biomimetics, and fluid dynamics of fish skin surfaces,” *Phys. Rev. Fluids*, vol. 1, no. 6, Oct. 2016.

[21] G. V. Lauder and E. D. Tytell, “Three Gray Classics on The Biomechanics of Animal Movement,” *J. Exp. Biol.*, vol. 207, no. 10, pp. 1597–1599, Apr. 2004.

[22] J. Zhu, C. White, D. K. Wainwright, V. Di Santo, G. V. Lauder, and H. Bart-Smith, “Tuna robotics: A high-frequency experimental platform exploring the performance space of swimming fishes,” *Sci. Robot.*, vol. 4, no. 34, Sep. 2019.

[23] J.-D. Liu and H. Hu, “Biologically inspired behaviour design for autonomous robotic fish,” *Int. J. Autom. Comput.*, vol. 3, no. 4, pp. 336–347, Oct. 2006.

[24] A. Jusufi, D. M. Vogt, R. J. Wood, and G. V. Lauder, “Undulatory Swimming Performance and Body Stiffness Modulation in a Soft Robotic Fish-Inspired Physical Model,” *Soft Robot.*, vol. 4, no. 3, pp. 202–210, Sep. 2017.

[25] C. J. Esposito, J. L. Tangorra, B. E. Flammang, and G. V. Lauder, “A robotic fish caudal fin: effects of stiffness and motor program on locomotor performance,” *J. Exp. Biol.*, vol. 215, no. 1, pp. 56–67, Jan. 2012.

[26] N. Gravish and G. V. Lauder, “Robotics-inspired biology,” *J. Exp. Biol.*, vol. 221, no. 7, Apr. 2018.

[27] M. H. Dickinson, F.-O. Lehmann, and S. P. Sane, “Wing Rotation and the Aerodynamic Basis of Insect Flight,” *Science*, vol. 284, no. 5422, pp. 1954–1960, Jun. 1999.

[28] K. Jayaram and R. J. Full, “Cockroaches traverse crevices, crawl rapidly in confined spaces, and inspire a soft, legged robot,” *Proc. Natl. Acad. Sci.*, vol. 113, no. 8, pp. E950–E957, Feb. 2016.

[29] R. Bale, I. D. Neveln, A. P. S. Bhalla, M. A. MacIver, and N. A. Patankar, "Convergent Evolution of Mechanically Optimal Locomotion in Aquatic Invertebrates and Vertebrates," *PLOS Biol.*, vol. 13, no. 4, Apr. 2015.

[30] O. Akanyeti, J. Putney, Y. R. Yanagitsuru, G. V. Lauder, W. J. Stewart, and J. C. Liao, "Accelerating fishes increase propulsive efficiency by modulating vortex ring geometry," *Proc. Natl. Acad. Sci.*, vol. 114, no. 52, pp. 13828–13833, Dec. 2017.

[31] K. Hirata, "Development of Experimental Fish Robot," *Sixth Int. Symp. Mar. Eng.*, no. 650, pp. 711–714, 2000.

[32] Z. Wang, G. Hang, J. Li, Y. Wang, and K. Xiao, "A micro-robot fish with embedded SMA wire actuated flexible biomimetic fin," *Sensors Actuators, A Phys.*, vol. 144, no. 2, pp. 354–360, 2008.

[33] J. Yu, L. Wang, and M. Tan, "Geometric Optimization of Relative Link Lengths for Biomimetic Robotic Fish," *IEEE Trans. Robot.*, vol. 23, no. 2, pp. 382–386, Apr. 2007.

[34] Y. Zhong, Z. Li, and R. Du, "A Novel Robot Fish With Wire-Driven Active Body and Compliant Tail," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 4, pp. 1633–1643, Aug. 2017.

[35] J. J. Videler and F. Hess, "Fast Continuous Swimming of Two Pelagic Predators, Saithe (Pollachius Virens) and Mackerel (Scomber Scombrus): a

Kinematic Analysis," *J. Exp. Biol.*, vol. 109, no. 1, pp. 209–228, Mar. 1984.

[36] O. Akanyeti and J. C. Liao, "A kinematic model of Karman gaiting in rainbow trout," *J. Exp. Biol.*, vol. 216, no. 24, pp. 4666–4677, Dec. 2013.

[37] G. Ozmen Koca *et al.*, "Three-Dimensional Modeling of a Robotic Fish Based on Real Carp Locomotion," *Appl. Sci.*, vol. 8, no. 2, p. 180, Jan. 2018.

[38] C. Fiazza *et al.*, "Biomimetic mechanical design for soft-bodied underwater vehicles," in *OCEANS'10 IEEE SYDNEY*, 2010, pp. 1–7.

[39] P. Domenici and R. Blake, "The kinematics and performance of fish fast-start swimming," *J. Exp. Biol.*, vol. 200, no. 8, pp. 1165–1178, 1997.

[40] C. C. Lindsey, "Form, Function, and Locomotory Habits in Fish," *Fish Physiol.*, vol. 7, pp. 1–100, 1978.

[41] E. D. Tytell and G. V. Lauder, "The hydrodynamics of eel swimming: I. Wake structure," *J. Exp. Biol.*, vol. 207, no. 11, pp. 1825–1841, May 2004.

[42] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Comput. Graph. Image Process.*, vol. 1, no. 3, pp. 244–256, Nov. 1972.

[43] M. Visvalingam and J. D. Whyatt, "Line generalisation by repeated elimination of points," *Cartogr. J.*, vol. 30, no. 1, pp. 46–51, Jun. 1993.

[44] C. Bal, D. Korkmaz, G. O. Koca, M. Ay, and Z. H. Akpolat, “Link length optimization of a biomimetic robotic fish based on Big Bang — Big Crunch algorithm,” in *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016, pp. 189–193.

[45] S. E. A. W. Fetherstonhaugh, Q. Shen, and O. Akanyeti, “Automatic segmentation of fish midlines for optimizing robot design,” *Bioinspir. Biomim.*, vol. 16, no. 4, Jul. 2021.

[46] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Second Edi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[47] N. M. Razali and J. Geraghty, “Genetic algorithm performance with different selection strategiesin solving TSP,” in *Proceedings of the World Congress on Engineering 2011, WCE 2011*, 2011, vol. 2, pp. 1134–1139.

[48] M. J. McHenry, “The mechanical scaling of coasting in zebrafish (Danio rerio),” *J. Exp. Biol.*, vol. 208, no. 12, pp. 2289–2301, Jun. 2005.

[49] B. Y. P. W. Webb, “Speed, Acceleration and Manoeuvrability of Two Teleost Fishes,” *J. Exp. Biol.*, vol. 102, no. 1, pp. 115–122, 1983.

[50] Z. Michalewicz, “A Survey of Constraint Handling Techniques in Evolutionary Computation Methods,” in *Evolutionary Programming*, 1995, pp. 135–155.

[51] T. Ray, K. Tai, and S. Chye, “An Evolutionary Algorithm for Constrained

Optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann*, 2000, pp. 771–777.

[52] J. Xiao, Y. Huang, Z. Cheng, J. He, and Y. Niu, "A hybrid membrane evolutionary algorithm for solving constrained optimization problems," *Optik (Stuttg).*, vol. 125, no. 2, pp. 897–902, Jan. 2014.

[53] Q. Fan and X. Yan, "Self-Adaptive Differential Evolution Algorithm With Zoning Evolution of Control Parameters and Adaptive Mutation Strategies," *IEEE Trans. Cybern.*, vol. 46, no. 1, pp. 219–232, Jan. 2016.

[54] C. M. Breder, "The locomotion of fishes," *Zool. Sci. Contrib. New York Zool. Soc.*, vol. 4, no. 5, pp. 159–297, 1926.

[55] Wardle, Videler, and Altringham, "Tuning in to fish swimming waves: body form, swimming mode and muscle function," *J. Exp. Biol.*, vol. 198, no. 8, pp. 1629–1636, 1995.

[56] H. T. Porter and P. J. Motta, "A comparison of strike and prey capture kinematics of three species of piscivorous fishes: Florida gar (Lepisosteus platyrhincus), redfin needlefish (Strongylura notata), and great barracuda (Sphyraena barracuda)," *Mar. Biol.*, vol. 145, no. 5, pp. 989–1000, Oct. 2004.

[57] P. W. Webb, "'Steady' Swimming Kinematics of Tiger Musky, an Esociform Accelerator, and Rainbow Trout, a Generalist Cruiser," *J. Exp. Biol.*, vol. 138, no. 1, pp. 51–69, Sep. 1988.

[58] R. J. Wootton, *Ecology of Teleost Fishes*. Dordrecht: Springer Netherlands, 1989.

[59] P. W. Webb, P. T. Kostecki, and E. D. O. N. Stevens, “The Effect of Size and Swimming Speed on Locomotor Kinematics of Rainbow Trout,” *J. Exp. Biol.*, vol. 109, no. 1, pp. 77–95, Mar. 1984.

[60] C. Zhou, Z. Cao, S. Wang, and M. Tan, “The Posture Control and 3-D Locomotion Implementation of Biomimetic Robot Fish,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5406–5411.

[61] B. C. Jayne and G. V. Lauder, “New Data on Axial Locomotion in Fishes: How Speed Affects Diversity of Kinematics and Motor Patterns,” *Am. Zool.*, vol. 36, no. 6, pp. 642–655, Dec. 1996.

[62] L. Rome, D. Swank, and D. Corda, “How fish power swimming,” *Science*, vol. 261, no. 5119, pp. 340–343, Jul. 1993.

[63] M. J. McHenry, C. A. Pell, and J. H. Long, “Mechanical control of swimming speed: stiffness and axial wave form in undulating fish models,” *J. Exp. Biol.*, vol. 198, no. 11, pp. 2293–2305, 1995.

[64] E. D. Tytell, C.-Y. Hsu, T. L. Williams, A. H. Cohen, and L. J. Fauci, “Interactions between internal forces, body stiffness, and fluid environment in a neuromechanical model of lamprey swimming,” *Proc. Natl. Acad. Sci.*, vol.

107, no. 46, pp. 19832–19837, Nov. 2010.

[65] M. W. Westneat and S. A. Wainwright, “7. Mechanical design for swimming: muscle, tendon, and bone,” *Fish Physiol.*, vol. 19, pp. 271–311, 2001.

[66] G. V. Lauder and E. D. Tytell, “Hydrodynamics of Undulatory Propulsion,” *Fish Physiol.*, vol. 23, pp. 425–468, 2005.

[67] W. M. van Rees, M. Gazzola, and P. Koumoutsakos, “Optimal morphokinematics for undulatory swimmers at intermediate Reynolds numbers,” *J. Fluid Mech.*, vol. 775, pp. 178–188, Jul. 2015.

[68] M. J. Lighthill, “Large-amplitude elongated-body theory of fish locomotion,” *Proc. R. Soc. London. Ser. B. Biol. Sci.*, vol. 179, no. 1055, pp. 125–138, Nov. 1971.

[69] G. I. Taylor, “Analysis of the swimming of long and narrow animals,” *Proc. R. Soc. London. Ser. A. Math. Phys. Sci.*, vol. 214, no. 1117, pp. 158–183, Aug. 1952.