4.1 Introduction

Previous chapters have introduced the theoretical basis and justifications behind the model, as well as some of the limitations and considerations required to develop the model into a practical solution. This chapter will introduce the software components produced to implement the model, as well as illustrating their usage and the general processes required to run a simulation.

4.2 Components

The software implementation of this model has largely been written in C, with some Python scripts and modules and a small number of supplementary scripts written for use with Bash. The C code used has been compiled and tested using various versions of GCC (up to 5.3.1), and has been written to C99 standards [1].

The software makes use of the following third party components:

- inih - © 2009 Brush Technology
  http://code.google.com/p/inih/
  Used to read input files in INI format [New BSD license]
- libxml2 - from the GNOME project
  http://www.xmlsoft.org
  Used by post-processors to write VTK compatible files [MIT licensed]
- libb64 - Chris Venter
  Base64 Encoding/Decoding routines required to write VTK format outputs [Public Domain]
- zlib - © 1995-2013 Jean-loup Gailly and Mark Adler
  http://www.zlib.net/
  Used to (optionally) compress some output files

Copies of the required code for inih and libb64 are included verbatim with the software, while libxml2 and zlib must be obtained and installed separately.

[1] Although code can be compiled with the `-std=c99` option to gcc, it has mainly been compiled and tested using `-std=gnu99`

For the purposes of this chapter a "component" of this implementation is any generalised executable or script. Libraries, shared code and most debugging and utility scripts are not discussed here. The components can be split into 4 broad groups: Pre-processing tools, the main model and post-processing and analysis tools.

There are various file formats used by the different components of this project. Where appropriate these will be discussed in the text below, with full details and examples can be found in Appendix A along with details of any files not discussed in this Chapter.

4.2.1 Case files

The vast majority of the components described below use a common settings file to ensure consistent transfer and treatment of data. This file defines the input and output paths, assigns meaning to different variables and sets the values of various simulation parameters. These files are plain text (ASCII) files, using the INI format. An excerpt from one of these files is shown below:

```
[paths]
basename = /home/tom/NorthSea/processed/r3snsNk001Cor.slf
output = /home/tom/NorthSea/outputs/
particles = /home/tom/NorthSea/NSPorp1.txt

[settings]
dimensions = 3
particle_steps = 800

[indexes]
fish = 8
```

The file is split into several sections, delimited by a section name enclosed in square brackets, e.g. `[paths]`. Within each section, parameters are specified in the format `name = value`. Parameters names are only recognised and interpreted when present in the correct section of the file. Valid sections, parameter names and their meanings are listed in Table 4.1. Values listed as "float" or "integer" should be positive, and variable numbers should match those listed in the corresponding `basename.vars.txt` file or the output of `varinfo`.

Name	Values	Meaning
Section: paths		
basename	string	Path and prefix for input files, typically generated with `telemac-parse`
particles	string	Path to porpoise definition file
output	string	Path to output folder
Section: settings		
dimensions	2, 3	Define a 2D or 3D simulation
particle_steps	integer	Number of simulation timesteps per mesh timestep
prefix	string	Gradient variable prefix - for auto-detection of pre-computed gradients
threshold	float	Minimum acceptable depth
noise_threshold	float	Maximum acceptable additional noise threshold
food_weight	float	Weighting applied for food weighting
longoutput	true, false	Specify long/full output format. Defaults to false
foodrule	0, 1	Specify food rule variant. Defaults to 0
Section: indexes		
z	variable	Variable number for Z coordinates
u	variable	Variable number for velocity component in x
v	variable	Variable number for velocity component in y
w	variable	Variable number for velocity component in z
noise	variable	Variable number for additional environmental noise
fish	variable	Variable number for food availability data
depth	variable	Variable number for precomputed depth data
Section: scaling		
fish	float, float[3]	Scaling value for food availability gradient. Either a single value or a vector of scaling factors for each Cartesian component
Section: speeds		
default	float	Default mean swimming speed, in ms^{-1}
defaultrange	float	Standard deviation of noise applied to default speed
ydamping	float	Y velocity damping factor
noise	float	Mean swimming speed for noise avoidance behaviour
noiserange	float	Standard deviation of noise applied during noise avoidance
depth	float	Mean swimming speed for shallow water avoidance behaviour
depthrange	float	Standard deviation of noise applied during shallow water avoidance

Table 4.1: Simulation case file parameter descriptions

4.2.2 Pre-processing

The pre-processing tools are responsible for converting and preparing input data for use with the main model. Not all of the components in this section are required in all cases.

telemac-parse (Part of tawe-telemac-utils)

`telemac-parse` is responsible for converting 2D or 3D TELEMAC data into the flat file formats used by this software. `telemac-parse` is part of a set of related utilities that have been made publicly available as `tawe-telemac-utils` on GitHub[2]. The folder of files produced by `telemac-parse` are prefixed with the name of the input TELEMAC file, and this path and prefix become the `basename` property of a simulation case file. The format and naming convention for these files is given in section A.1 of Appendix A.

Importing a 2D mesh: `convert_to_3D`

When a 2D TELEMAC mesh is used as input, it is necessary to "upgrade" the mesh to 3D based on the depth averaged velocity and water depth. This is done using `convert_to_3D`, which stacks a 2D triangular mesh to form a number of depth conforming layers of prismatic elements. The 2D depth averaged velocity components u and v are scaled to provide a vertical profile fitting a $\frac{1}{7}^{\text{th}}$ power law with no vertical motion.

Cached mesh data

In order to improve performance of the model, the mesh connectivity and generated data structures are stored on disk for access by the various other processes. This file is generated by the `prepmesh` utility, which must be run before any tools other than initial data import (such as `telemac-parse` and `convert_to_3D`).

Precomputing water depth: `calcdepth`

In order to improve the run-time performance of the model, the water depth at each point in the domain is calculated and stored as an additional variable. This removes the need to traverse the mesh at each timestep to calculate the depth when determining which behaviour rule to follow. Some input data may include a suitable variable already, or as a consequence of being converted to 3D using `convert_to_3D` - in these cases there is no need to calculate another separate depth value.

[2] https://github.com/tswsl1989/tawe-telemac-utils

In all cases, the variable number corresponding to the calculated or imported depth need to be supplied in the simulation case file.

Adding new information

In order for the animals within the model to react to food availability and additional noise, it is necessary to add that information to the simulation. There are currently three tools provided to create additional variables based on discrete sources and a spatially varying strength.

- `datafill.py`
  `datafill.py` can produce a new variable based on either random values at each node or as the sum of the effect of discrete point sources. These value of the field around each point source is given by $\frac{a}{\sqrt{\Delta x^2+\Delta y^2}}$ where a is the value of the source and $\Delta x, \Delta y$ are the difference in x and y position respectively. The generated field is constant at all depths (no vertical variation). The point sources are defined by a value and their Cartesian coordinates.

- `lldataconvert.py`
  As with `datafill.py`, `lldataconvert.py` produces a variable that is the sum of a number of discrete point sources. It differs slightly in that the input points can be specified using latitude, longitude and a strength value. The lat,lon position is converted to Universal Transverse Mercator coordinates, then the combined effect of each point source at each node is calculated. The value of each point source is calculated as $\frac{a}{\left(\Delta x^2+\Delta y^2\right)^b}$ where a is the value of the source, $\Delta x, \Delta y$ are the differences in position as above and b can be varied to adjust how rapidly the field drops with respect to difference.

  Both of these python scripts produce static fields with respect to time - the position and strength of the defined sources cannot vary over time.

- `fielddata`
  Unlike the other two options, `fielddata` is able to produce time varying fields. A series of files containing positions and values need to be prepared, with one file for each mesh timestep. These files are then used to calculate the value of the field at each node at each mesh timestep. The field strength at each point is

given by $\frac{k.a}{\left(\Delta x^2+\Delta y^2\right)^b}$, with k being an additional multiplier value specified when creating the field and all other terms defined as above. Unlike the previous two scripts, the sources defined here have a finite radius and any node lying within the radius of a source has it's contribution from that source set to the source value, independent of where within the radius it falls.

Defining porpoise

In addition to defining the simulation environment, it is necessary to define the properties and initial states of the porpoise to be simulated. The full list of properties can be found in Table 3.6, although only some of these can be defined from input.

Similar to the use of case files described in subsection 4.2.1, the properties of the porpoise are defined in an INI format file. This is the file referenced in the `particles` property of the case file. The file starts with an `[info]` section containing the number of porpoise and an optional comment, then optional `[defaults]` section, followed by a section for each porpoise, sequentially numbered starting from zero - i.e. `[0]`, `[1]`, `[2]`. The properties that can be set in each section of the file are listed in Table 4.2.

Name	Values	Meaning
Section: info		
comment	string	Descriptive text displayed at run time
np	integer	Number of porpoise to include in the simulation
Section: defaults or **id**		
position	float[3]	Porpoise release coordinates
velocity	float[3]	Initial release velocity (ms^{-1})
orientation	float[3]	Initial orientation angles (α, β, γ)
drag	float[3]	Drag coefficients
area	float[3]	Reference areas

Table 4.2: Porpoise definition file parameters

The `generate_particles.py` script is an interactive utility which prompts for values and allowable ranges for position, orientation and velocity and values for mass, drag coefficients and reference areas. A population is then generated with randomised parameters based on the values and ranges provided. No checks are made to determine whether the generated positions are valid, so some adjustment is typically needed after the file is generated.

4.2.3 Main model

The main model, `simulation`, uses the information provided in the case file to locate the particle definition file, mesh cache and input and output folders. Optional arguments allow a simulation to be resumed from a checkpoint file, early termination of a simulation (rather than using the time defined in the input data) and control over logging output and the number of processes used.

The simulation output is placed into a text file containing the position and full or partial state of each porpoise at each simulation timestep, depending on the options supplied in the case file.

4.2.4 Post-processing

Post-processing tools developed alongside the implementation can be split into 3 main categories: Extraction, Rendering and Analysis

Extraction tools

The extraction tools are designed to partition output data, reducing the files to a more manageable size.

- `track_converter.py`
  Take a track file and convert it to a series of CSV files - one file per simulation timestep.
- `extractparticle`
  Extracts position and state of a specified porpoise at each simulation timestep into a single text file.

Render tools

The rendering tools provided allow results to be viewed and inspected graphically, either as 2D plots/graphs or as 3D graphics files using Paraview[49] - an open source visualisation tool that can import and export data from a variety of formats.

The main tools for export to Paraview[3] are `vtkparticles` and `vtkexport`. These tools

[3] Strictly speaking the tools output to VTK format files, which are compatible with a range of software including Paraview. Only Paraview was used or tested.

take a case file as input, and allow the export of one or more timesteps at specified intervals. In order to provide information regarding the clock time of each simulation timestep, a wrapper PVD file is created, referencing the individual VTK format files. This allows Paraview to (optionally) display the correct time as an annotation and as part of the interface.

In addition to Paraview, a number of Python scripts have been used to plot smaller meshes and extracted data. These scripts make use of the following 3rd party libraries:

- Matplotlib[50]
  http://www.matplotlib.org
- NumPy
  http://www.numpy.org/
- Seaborn
  https://stanford.edu/~mwaskom/software/seaborn/, and
- Pandas
  http://pandas.pydata.org/

These libraries are use to plot, visualise and analyse the available data, and have been used to generate many of the figures in this thesis.

There are three main Python scripts used to visualise results:

- `plot_particle_set.py` - Plots a 2D overview of multiple porpoise tracks
- `plot_combined.py` - Plots 2D or 3D mesh data, optionally including particle start and end points.
- `plot_extracted_particle.py` - Takes data for a single porpoise extracted by `extractparticle` and plots state variables against time.

Analysis

In addition to rendering the results graphically for visual inspection and interpretation, numerical and statistical results can also be extracted from the simulations. These are generated by two tools, `particlestats` and `trackstats`.

- `particlestats` - generates summary statistics (min/max/mean/standard deviation) for position and velocity of all porpoise at each timestep.
- `trackstats` - generates summary statistics for position and velocity of all porpoise over all timesteps.

Both tools can also operate on subsets of the simulated population, allowing the population to be sampled for comparisons. This is used and discussed further in chapter 5.

The scripts and programs described above provide the building blocks for carrying out a simulation based investigation into Harbour Porpoise behaviours in tidal sites, and allow the results to be examined in a number of ways. To illustrate their use, an example case is illustrated below.

4.3 Process Overview

The general procedure for creating and running a simulation is shown in Figure 3.4, with the data preparation steps broken out into Figure 3.5. These processes are typically only required once for any given environment, unless changes need to be made to the additional noise sources or prey locations. There is no explicit limit to the number of fields that can be added to a given simulation, although each additional variable does require additional processing time when the simulation is run.

Taking a 2D TELEMAC file as the basis of a simulation, the steps in Figure 3.4 and Figure 3.5 become the following commands:

```
telemac-parse -o ../RS2D -b ../RS_basecase.slf
convert_to_3D -bBv -n 5 -d 2 -U 0 -V 1 ../RS2D/RS_basecase.slf
prepmesh -b RS3D.ini
calcdepth -v -o 7 RS3D.ini
fielddata -vfo 8 RS3D.ini fish.fp.ini
fielddata -v -o 9 RS3D.ini noise.fp.ini
precomputegrads -i 7 -o 10 -v RS3D.ini
precomputegrads -i 8 -o 13 -v RS3D.ini
precomputegrads -i 9 -o 16 -v RS3D.ini
```

These commands, in order, import data from TELEMAC, convert it from 2D depth averaged to 3D, cache the mesh information and calculate the water depth throughout

the domain. The fields representing fish/food and noise are calculated and the gradients of the depth, food and noise variables are computed and stored in order to improve the speed of the main simulation.

This set of commands would typically only be required once for a given scenario, with multiple simulations being run from this prepared data set.

4.4 Speed and efficiency

4.5 Initial tests

4.6 Model Outputs

4.7 Comparing results