



Defence Research and
Development Canada

Recherche et développement
pour la défense Canada



Improved Simulation of Ship Mechanical Systems

*A. Roy
D. Steinke
R. Nicoll
Dynamic Systems Analysis Limited*

*Prepared by:
Dynamic Systems Analysis Limited
101-19 Dallas Road, Victoria, BC, Canada V8V 5A6*

*Project Manager: Dean Steinke, 902-407-3722
Contract Number: W7707-115188/001/HAL
Contract Scientific Authority: Kevin McTaggart, 902-426-3100 ext 253*

The scientific or technical validity of this Contract Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2012-093
May 2012

This page intentionally left blank.

Improved Simulation of Ship Mechanical Systems

A. Roy
D. Steinke
R. Nicoll

Prepared by:

Dynamic Systems Analysis Limited
101-19 Dallas Road, Victoria, BC, Canada, V8V 5A6

Project Manager: Dean Steinke 902-407-3722

Contract Number: W7707-115188/001/HAL

Contract Scientific Authority: Kevin McTaggart 902-426-3100 ext. 253

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

Defence Research and Development Canada – Atlantic

Contract Report

DRDC Atlantic CR 2012-093

May 2012

- © Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 2012
- © Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2012

Abstract

Previous work developed simulation components for ship mechanical systems, including cables, winches, cranes, and payloads. The payload modelling includes treatment of collisions, such as those that can occur when cargo being lifted by a crane strikes the side of a ship. The payload modelling also includes treatment of hydrodynamic forces in calm water and in waves. This report describes improvements made to simulation components to achieve increases in both fidelity and computational speed. Verification and validation of simulation components were performed using test cases of varying complexity. A launch and recovery example demonstrates the usage of crane and payload models when deploying a small boat from a large ship in waves. A towing example demonstrates the usage of the cable model when a naval frigate is towing a tuna clipper.

Résumé

Des travaux antérieurs ont permis de développer des composants de simulation pour les systèmes mécaniques des navires, y compris les câbles, treuils, grues et charges utiles. La modélisation de la charge utile comprend le traitement des collisions, comme celles qui peuvent se produire lorsqu'une cargaison levée par une grue frappe le côté d'un navire. La modélisation de la charge utile comprend également le traitement des forces hydrodynamiques en eau calme et dans les vagues. Le présent rapport décrit les améliorations apportées aux composants de simulation en vue d'augmenter la fidélité et la vitesse computationnelle. La vérification et la validation des composants de simulation ont été effectuées en utilisant des scénarios d'essai de diverses complexités. Un exemple de lancement et récupération démontre l'utilisation de modèles avec grue et charge utile pendant le déploiement d'un petit bateau à partir d'un gros navire dans les vagues. Un exemple de remorquage démontre l'utilisation du modèle de câble pendant qu'une frégate remorque un thonier.

This page intentionally left blank.

Executive summary

Improved Simulation of Ship Mechanical Systems

A. Roy, D. Steinke, R. Nicoll; DRDC Atlantic CR 2012-093; Defence Research and Development Canada – Atlantic; May 2012.

Introduction: Simulation of multi-body dynamics is required when simulating many naval operations, including replenishment at sea, launch and recovery, and towing. Under a previous contract, a software library was developed for simulation of cranes, winches, cables, and payloads.

Principal Results: This report describes improvements made to simulation components for ship mechanical systems to achieve increases in both fidelity and computational speed. Verification and validation of simulation components were performed using test cases of varying complexity. A launch and recovery example demonstrates the usage of crane and payload models when deploying a small boat from a large ship in waves. A towing example demonstrates the usage of the cable model when a naval frigate is towing a tuna clipper. The simulation components typically run somewhat slower than real-time.

Significance of Results: Simulation components are now available for representing a range of naval platform systems. The simulation components can be easily configured to model different systems, and can be used to explore proposed new systems. The fidelity of the developed components and relatively fast execution speeds make the simulation components suitable for a variety of applications.

Future Plans: Future optimization of simulation components will enable them to run in real-time, making them suitable for training applications.

Sommaire

Improved Simulation of Ship Mechanical Systems

A. Roy, D. Steinke, R. Nicoll ; DRDC Atlantic CR 2012-093 ; Recherche et développement pour la défense Canada – Atlantique ; mai 2012.

Introduction : La simulation de dynamique multicorps est requise pour la simulation de plusieurs opérations navales, y compris le ravitaillement en mer, le lancement et la récupération et le remorquage. Dans le cadre d'un précédent contrat, une bibliothèque de logiciels a été élaborée pour la simulation de grues, de treuils, de câbles et de charge utiles.

Résultats principaux : Le présent rapport décrit les améliorations apportées aux composants de simulation en vue d'augmenter la fidélité et la vitesse computationnelle. La vérification et la validation des composants de simulation ont été effectuées en utilisant des scénarios d'essai de diverses complexités. Un exemple de lancement et récupération démontre l'utilisation de modèles avec grue et charge utile pendant le déploiement d'un petit bateau à partir d'un gros navire dans les vagues. Un exemple de remorquage démontre l'utilisation du modèle de câble pendant qu'une frégate remorque un thonier. Les composants de simulation fonctionnent généralement plus lentement qu'en temps réel.

Importance des résultats : Les composants de simulation sont maintenant disponibles pour la représentation d'une gamme de systèmes de plate-forme navale. Les composants de simulation peuvent être facilement configurés pour modéliser divers systèmes, et peuvent servir à l'exploration des nouveaux systèmes proposés. La fidélité des composants développés ainsi que les vitesses d'exécution relativement rapides rendent les composants de simulation convenables pour une variété d'applications.

Travaux ultérieurs prévus : L'optimisation future des composants de simulation permettra leur utilisation en temps réel, les rendant convenables à des applications d'instruction.

Table of contents

Abstract	i
Résumé	i
Executive summary	iii
Sommaire	iv
Table of contents	v
List of figures	xi
List of tables	xix
1 Introduction	1
1.1 Background	1
1.2 Report overview	1
1.2.1 Payload hydrodynamics improvements overview	2
1.2.2 Director class overview	2
1.2.3 Contact dynamics improvements overview	3
1.2.4 Launch and recovery and towing simulation overview	4
2 SMS API development	5
2.1 Payload hydrodynamics	5
2.1.1 Newton-Euler equation of motion of an SMS::Payload	5
2.1.2 Object surface discretisation	6
2.1.3 Morison equation	7
2.1.4 Description of the fluid domain	8
2.1.5 The Froude-Krylov force	9
2.1.6 Hydrodynamic drag forces	10
2.1.7 Added Mass forces	14

2.1.8	Using ShipMo3D to determine Added Mass for an SMS::Payload	16
2.1.9	GPGPU parallelization of the hydrodynamic force calculation	18
2.2	SMS::Director class	21
2.2.1	Using the SMS::Director	22
2.2.2	SMS::Payload commands	23
2.2.3	SMS::Boomcrane commands	24
2.2.4	SMS::Winch commands	24
2.2.5	SMS::Cable commands	24
2.2.6	Elapsing time between commands	24
2.2.7	Example code	25
2.3	Contact dynamics	26
2.3.1	Improved collision detection code	26
2.3.2	Expanded Polytope Algorithm	28
2.3.3	Exploiting temporal coherence with GJK	30
2.3.4	Contact dynamics code performance	30
2.4	Generalised- α implicit integrator	34
2.4.1	Implicit integration	35
2.4.2	The Generalised- α integrator	35
2.4.3	Implementation for nonlinear systems	37
3	Validation	40
3.1	Hydrodynamics validation	40
3.1.1	Partially submerged body buoyancy test in calm water . . .	40
3.1.2	Fully submerged body pendulum test in calm water	42
3.1.3	Hydrodynamic drag test	43

3.1.4	Fully submerged body added mass test	44
3.1.5	Additional proposed verification tests	46
3.2	Contact dynamics validation for convex decomposed objects	48
3.2.1	Subdivided and non-subdivided box collision tests	48
3.2.2	Oriented box collision test	52
3.2.3	Multiple contact point collision test	57
3.2.4	Boat and cradle collision test	60
3.3	Generalised- α validation	63
3.3.1	MSD	63
3.3.2	Pendulum	64
4	Launch and Recovery simulation	69
4.1	Simplified Launch and Recovery simulation	69
4.1.1	Simulation setup	69
4.1.2	Simulation results	74
4.2	Improving execution speeds	75
4.2.1	The integration timestep size	75
4.2.2	Improved launch and recovery simulation execution performance	82
4.2.3	Profiling	82
4.3	Full Launch and Recovery simulation	93
4.3.1	Simulation results	97
5	Tuna Clipper Towing simulation	101
5.1	Simplified towing simulation	101
5.1.1	Simulation Setup	101
5.1.2	Simulation results	102

5.2	Full Tuna Clipper Towing simulation	102
5.2.1	Results	104
6	Future work	106
6.1	General	106
6.2	Winch class	106
6.3	Contact dynamics	106
6.3.1	Resolving N-body collisions	106
6.3.2	Continuous collision detection	106
6.3.3	Rolling resistance implementation	107
6.3.4	Friction model improvements	107
6.3.5	Volume of interference research	107
6.3.6	Volume of interference limitations	107
6.3.7	Winkler elastic bed depth	108
6.3.8	Improving normal contact model fidelity	108
6.4	Cable improvements	108
6.4.1	Clamped terminations	108
6.4.2	Implicit numerical integration	108
6.5	Launch and Recovery simulation	109
6.6	Tuna Clipper Towing simulation	109
7	Conclusion	110
	References	111
	Annex A: Sample initialisation files	113
A.1	Cable class	113
A.2	Winch class	115

A.3	Payload class	115
A.4	Boomcrane class	116
Annex B:	Descriptions of important class methods	121
B.1	Payload class methods	121
B.1.1	Creating a payload object	121
B.1.2	Overriding the payload position and velocity	121
B.1.3	Overriding gravitational constant	121
B.2	Adding a hydrodynamics mesh for non-linear hydrodynamics	122
B.2.1	Defining wave conditions of the environment	122
B.2.2	Adding contact dynamics geometries	123
B.2.3	Setting the material properties for contact dynamics	123
B.2.4	Changing the position and orientation of the external contact objects	124
B.2.5	Advancing simulation through time	124
B.2.6	Outputting simulation results to disk	124
B.3	Cable class methods	124
B.3.1	Creating a Cable object	124
B.3.2	Defining wave conditions of the environment	125
B.3.3	Attaching a Winch to the cable	125
B.3.4	Setting the Cable's end node condition	125
B.3.5	Advancing simulation through time	126
B.3.6	Outputting simulation results to disk	126
B.4	Winch class methods	126
B.4.1	Creating a Winch object	126
B.4.2	Setting the winch's controller mode	126

B.4.3	Setting the velocity control set point	126
B.4.4	Setting the tension control set point	127
B.5	Boomcrane class methods	127
B.5.1	Creating a Boomcrane object	127
B.5.2	Attaching a Cable to one of the Boomcrane's links .	127
B.5.3	Adjusting the boomcrane base position and orientation	127
B.5.4	Advancing simulation through time	128
B.5.5	Outputting simulation results to disk	128
B.6	Director class methods	128
B.6.1	Creation a Director object	128
B.6.2	Adding actors	128
B.6.3	Adding acting command to a script	128
B.6.4	Loading a script from file	129
B.6.5	Managing a scene	129
Annex C:	Supplementary plots	131
C.1	Plots of the simplified Launch and Recovery simulation from Section 4.1	131
C.2	Plots from the process of improving execution speeds of the Launch and Recovery simulation from Section 4.2.1	133
C.3	Plots from the process of Identifying source of small timesteps from Section 4.2.1.2	135
C.3.1	Plots of the optimised Launch and Recovery simulation of Section 4.2.2	137

List of figures

Figure 1:	a) The fluid pressure distribution over the wetted surface of a cylinder from buoyancy in calm water. b) The polygonal mesh discretisation used for the surface integral of the pressure field over the body surface.	7
Figure 2:	Evaluation of the constant for Bernoulli's equation for a deep seaway by sampling the fluid state at a point \mathbf{P}_{sample} on the surface where pressure and velocities are known.	9
Figure 3:	Experimental setup for the variable buoyancy validation test . . .	11
Figure 4:	Normal and tangential drag coefficients for a cylinder	12
Figure 5:	The path of the flow of fluid across a) a slender rotating object and b) a non-slender rotating object.	14
Figure 6:	The hull lines as defined for ShipMo3D panelling method of a a) $6 \times 6 \times 2$ box b) $100 \times 6 \times 12$ box.	17
Figure 7:	The resulting panelled hull created by ShipMo3D panelling method of a a) $6 \times 6 \times 2$ box b) $100 \times 6 \times 12$ box.	18
Figure 8:	NVIDIA GPU with an array of multi-threaded simultaneous multi-processors	20
Figure 9:	Performance results of a simple buoyancy calculation implemented in C, in C++ using <code>GeomLib</code> , in CUDA and in CUDA with shared memory. CUDA calculations were performed on an EVGA GeForce GTX 465 graphics card.	22
Figure 10:	The use of the MTD to obtain a point within the interference volume of two colliding objects.	28

Figure 11:	A step by step description of the Expanded Polytope Algorithm for interfering convex polyhedra: a) The desired point on the surface of the Minkowski Difference surface that is closest to the origin \mathbf{P}_{min} , b) the original simplex returned from GJK containing the origin showing the search direction vector $\hat{\mathbf{n}}$ from the normal of the closest face along with the furthest vertex in that direction, c) The new geometry with the furthest point from the previous step including the new search direction and furthest point, d) finally no further point can be identified in the search direction, thus the closest face contains \mathbf{P}_{min}	29
Figure 12:	The execution times for the GJK algorithm for pairs of identical spheres of varying mesh resolutions.	32
Figure 13:	The execution times for the EPA algorithm for pairs of identical spheres of varying mesh resolutions.	33
Figure 14:	The execution times for the EPA algorithm for a pair of identical spheres of 16128 polygons each for varying penetration depths. . .	33
Figure 15:	The execution times for the Muller-Preparata algorithm for pairs of identical spheres of varying mesh resolutions.	34
Figure 16:	The experimental setup for the variable buoyancy validation test.	40
Figure 17:	The $\hat{\mathbf{Z}}$ position of an <code>SMS::Payload</code> floating on water for the variable buoyancy validation test.	41
Figure 18:	The orientation of an <code>SMS::Payload</code> floating on water for the variable buoyancy validation test.	41
Figure 19:	The experimental setup for the fully submerged body buoyancy pendulum validation test.	42
Figure 20:	The $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ position of a <code>SMS::Payload</code> attached to a 10 m cable pendulating by a buoyancy force larger than its weight. . . .	43
Figure 21:	Experimental setup for the hydrodynamic drag test	44
Figure 22:	The simulated position of the <code>SMS::Payload</code> with a 128 face mesh	45
Figure 23:	The simulated position of the <code>SMS::Payload</code> with a 512 face mesh	45
Figure 24:	Experimental setup for the variable buoyancy validation test with added mass	46

Figure 25:	The $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ position of a SMS::Payload attached to a 10 m cable pendulating by a buoyancy force larger than its weight incorporating added mass	47
Figure 26:	The simulation setup of the convex decomposed box impacting another convex decomposed box.	49
Figure 27:	The simulations results of position of the one-piece and convex decomposed SMS::Payloads over time.	50
Figure 28:	The simulations results of orientation of the one-piece and convex decomposed SMS::Payloads over time.	50
Figure 29:	The simulations results of stiffness and damping components of the one-piece and convex decomposed SMS::Payloads over time.	51
Figure 30:	The simulations results of MSD/MTD and interference geometry of the one-piece and convex decomposed SMS::Payloads over time.	51
Figure 31:	The simulation setup for the oriented convex decomposed box impacting another convex decomposed box.	52
Figure 32:	The simulated position for the pre-oriented convex decomposed SMS::Payloads over time.	53
Figure 33:	The simulated orientation for the pre-oriented convex decomposed SMS::Payloads over time.	53
Figure 34:	The simulated stiffness and damping components of the pre-oriented convex decomposed SMS::Payloads over time.	54
Figure 35:	A description of the effect of angular velocity, ω , on the contact force without rolling resistance for a one-piece convex box and a subdivided box. The initial state of the a) one-piece and c) subdivided box, showing the contact force cause by the individual volumes of interference, and the next time step showing no change in the volume of interference for the one-piece box in b) and the changes in the volumes of interference of the sub pieces of the convex decomposed box d).	55
Figure 36:	A comparison of the position of the 1 piece and 4 piece subdivided pre-oriented box SMS::Payloads over time.	56
Figure 37:	A comparison of the position of the 4 piece and 16 piece subdivided pre-oriented box SMS::Payloads over time.	56

Figure 38:	The simulation setup of the multiple contact point simulation example.	57
Figure 39:	The position of the <code>SMS::Payload</code> over time.	58
Figure 40:	The Orientation of the <code>SMS::Payload</code> over time.	58
Figure 41:	The contact force components of the <code>SMS::Payload</code> over time. . .	59
Figure 42:	the volume of interference and MSD/MTD of the <code>SMS::Payload</code> over time.	59
Figure 43:	The decomposition of the rescue boat hydrodynamic hull mesh into 19 convex sub-pieces.	61
Figure 44:	The boat and cradle simulation setup.	61
Figure 45:	The position of the <code>SMS::Payload</code> for the boat through time. . . .	62
Figure 46:	The orientation of the <code>SMS::Payload</code> for boat through time. . . .	62
Figure 47:	The number of collision pairs between the <code>SMS::Payload</code> and the cradle through time.	63
Figure 48:	The setup for the Generalised- α mass-spring-damper test case. . .	64
Figure 49:	The time history of position of the mass for the mass spring damper using the Generalized- α integrator and the RK45 integrator. . .	65
Figure 50:	The setup for the Generalised- α mass-spring-damper test case. . .	66
Figure 51:	The $\hat{\mathbf{X}}$ position of the mass over time for both the RK45 and Generalised- α integrator simulations.	67
Figure 52:	The cable tensions for the Generalised- α integrator simulation. . .	67
Figure 53:	The cable tensions for the RK45 integrator simulation.	68
Figure 54:	The simulation scenario for the Simplified Launch and Recovery simulation. The Palfinger-like boomcrane is shown folded (dotted), with a zero joint configuration (solid) and in an unfolded pose (dashed).	70
Figure 55:	The rescue boat a) visualization mesh and b) hydrodynamic polyhedral mesh.	71

Figure 56:	The Launch and Recovery simulation setup with the boomcrane in a folded configuration (dotted outline and light color) and fully unfolded configuration (solid outline and darker color).	77
Figure 57:	The integration timestep size over the course of the Launch and Recovery simulation.	78
Figure 58:	The integration timestep size over the course of the improved launch and recovery example simulation.	82
Figure 59:	A screenshot from a visualisation of the Launch and Recovery simulation, showing the rescue boat, cradle, frigate, boomcrane, and cable.	94
Figure 60:	A snapshot of the Launch and Recovery simulation as the Palfinger-like boomcrane begins to unfold.	98
Figure 61:	A snapshot of the Launch and Recovery simulation as the rescue boat is lifted out of its cradle.	98
Figure 62:	A snapshot of the Launch and Recovery simulation after the rescue boat was dropped in the water.	99
Figure 63:	A snapshot of the Launch and Recovery simulation as the rescue boat is being lifted out of the water showing the tag lines used to prevent the boat from yawing.	99
Figure 64:	A snapshot of the Launch and Recovery simulations with the rescue boat back in its cradle.	100
Figure 65:	Simplified simulation of a small vessel tow.	101
Figure 66:	The rescue boat a) visualization mesh and b) hydrodynamic polyhedral mesh.	102
Figure 67:	The simulated rescue boat vessel position over time.	103
Figure 68:	The simulated cable node N position over time.	103
Figure 69:	A visualisation of the Tuna Clipper Towing simulation.	104
Figure 70:	The tensions in the tow cable for the Tuna Clipper Towing simulation.	105

Figure C.1: The simulated position of the rescue boat for the Simplified Launch and Recovery simulation (the $\hat{\mathbf{X}}$ position indicated in red, the $\hat{\mathbf{Y}}$ position indicated in green and the $\hat{\mathbf{Z}}$ position indicated in blue).	131
Figure C.2: The simulated joint positions of the boomcrane for the Simplified Launch and Recovery simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).	132
Figure C.3: The simulated position of the cable end nodes for the Simplified Launch and Recovery simulation (the cable's node 0 $\hat{\mathbf{X}}$ position is indicated in red, the cable's node 0 $\hat{\mathbf{Y}}$ position is indicated in green, the cable's node 0 $\hat{\mathbf{Z}}$ position is indicated in blue, the cable's node N $\hat{\mathbf{X}}$ position is indicated in yellow, the cable's node N $\hat{\mathbf{Y}}$ position is indicated in magenta, the cable's node N $\hat{\mathbf{Z}}$ position is indicated in cyan).	132
Figure C.4: The simulated tension of the cable's first element for the Simplified Launch and Recovery simulation.	133
Figure C.5: The position of the rescue boat over the course of the Launch and Recovery simulation (the $\hat{\mathbf{X}}$ position indicated in red, the $\hat{\mathbf{Y}}$ position indicated in green and the $\hat{\mathbf{Z}}$ position indicated in blue).	133
Figure C.6: The position of the Palfinger-like boomcrane joints over the course of the Launch and Recovery simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).	134
Figure C.7: The position of the cable's end node 0 over the course of the Launch and Recovery simulation (the cable's node 0 $\hat{\mathbf{X}}$ position is indicated in red, the cable's node 0 $\hat{\mathbf{Y}}$ position is indicated in green, the cable's node 0 $\hat{\mathbf{Z}}$ position is indicated in blue, the cable's node N $\hat{\mathbf{X}}$ position is indicated in yellow, the cable's node N $\hat{\mathbf{Y}}$ position is indicated in magenta, the cable's node N $\hat{\mathbf{Z}}$ position is indicated in cyan).	134

Figure C.8: The tension in the cable's first element over the course of the Launch and Recovery simulation.	135
Figure C.9: The integration timestep size over the course of the simulation for the 4 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).	135
Figure C.10: The integration execution speed time ratio over the course of the simulation for the 4 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).	136
Figure C.11: The integration timestep size over the course of the simulation for the 3 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).	136
Figure C.12: The time ratio over the course of the simulation for the 3 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).	137
Figure C.13: The position of the rescue boat over the course of the improved launch and recovery example simulation (the $\hat{\mathbf{X}}$ position indicated in red, the $\hat{\mathbf{Y}}$ position indicated in green and the $\hat{\mathbf{Z}}$ position indicated in blue).	137
Figure C.14: The position of the boomcrane joints over the course of the improved launch and recovery example simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).	138

Figure C.15: The position of the cable's end node 0 over the course of the improved launch and recovery example simulation (the cable's node 0 $\hat{\mathbf{X}}$ position is indicated in red, the cable's node 0 $\hat{\mathbf{Y}}$ position is indicated in green, the cable's node 0 $\hat{\mathbf{Z}}$ position is indicated in blue, the cable's node N $\hat{\mathbf{X}}$ position is indicated in yellow, the cable's node N $\hat{\mathbf{Y}}$ position is indicated in magenta, the cable's node N $\hat{\mathbf{Z}}$ position is indicated in cyan). 138

Figure C.16: The tension in the cable's first element over the course of the improved launch and recovery example simulation. 139

List of tables

Table 1:	The added mass coefficients found by ShipMo3D for the $6 \times 6 \times 2$ and the expected added mass coefficients in roll and heave.	19
Table 2:	The added mass coefficients found by ShipMo3D for the $100 \times 6 \times 12$ and the expected added mass coefficients in roll and heave.	19
Table 3:	The time ratios for the boat and cradle simulation before and after optimisation, without temporal coherence and with the original brute force MTD method.	31
Table 4:	The expected and simulated $\hat{\mathbf{Z}}$ position of the <code>SMS::Payload</code> . . .	40
Table 5:	The expected and simulated period of oscillation of the <code>SMS::Payload</code>	43
Table 6:	The expected and simulated <code>SMS::Cable</code> angle with the vertical $\hat{\mathbf{Z}}$ axis for a 128 face mesh and 512 face mesh, respectively.	45
Table 7:	The expected and simulated period of oscillation of the <code>SMS::Payload</code>	47
Table 8:	The <code>SMS::BoomCrane</code> 's initial D-H Parameters in a folded configuration for the Simplified Launch and Recovery capabilities test. Entries with a $()^*$ represent the joint variable.	71
Table 9:	Table of untuned PID controller gains for the Palfinger-like boomcrane, where only P gain and viscous damping are used. Here ξ is a placeholder for <i>rad</i> for joints 1-3 and <i>m</i> for joints 4-7.	80
Table 10:	Table of tuned PID controller gains, tuned to maximize integration timestep. No D gain is used in the presence of joint viscous damping. Here ξ is a placeholder for <i>rad</i> for joints 1-3 and <i>m</i> for joints 4-7.	81
Table 11:	The execution time of SMS API <code>CalcDynamics()</code> functions for 1.0 second of simulation time with hydrodynamics calculations (SMS API v0.1.261).	83

Table 12:	The execution time of <code>SMS::Payload::CalcDynamics()</code> and <code>SMS::RigidBody::CalcDynamics()</code> functions for 1.0 second of simulation time without hydrodynamics calculations (SMS API v0.1.261).	83
Table 13:	The execution times of <code>SMS::BoomCrane::CalcDynamics()</code> sub-functions for 1.0 second of simulation time (SMS API v0.1.261).	84
Table 14:	The execution times of <code>SMS::Cable::CalcDynamics()</code> sub-functions for 1.0 second of simulation time (SMS API v0.1.261).	85
Table 15:	The execution times of <code>SMS::Payload::CalcDynamics()</code> sub-functions for 1.0 second of simulation time with hydrodynamics calculations enabled (SMS API v0.1.261).	86
Table 16:	The execution times of <code>SMS::RigidBody::CalcDynamics()</code> sub-functions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.261).	86
Table 17:	The execution times of <code>SMS::RigidBody::CalcForces()</code> sub-functions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.261).	86
Table 18:	The execution times of <code>SMS::Payload::CalcDynamics()</code> sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).	88
Table 19:	The execution times of <code>SMS::RigidBody::CalcDynamics()</code> sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).	88
Table 20:	The execution times of <code>SMS::RigidBody::CalcForces()</code> sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).	88
Table 21:	The execution time of SMS API <code>CalcDynamics()</code> functions for 1.0 second of simulation time, with hydrodynamics calculations (SMS API v0.1.263).	89
Table 22:	The execution time of <code>SMS::Payload::CalcDynamics()</code> and <code>SMS::RigidBody::CalcDynamics()</code> functions for 1.0 second of simulation time, without a hydrodynamics (SMS API v0.1.263).	89
Table 23:	The execution times of <code>SMS::BoomCrane::CalcDynamics()</code> subfunctions for 1.0 second of simulation time (SMS API v0.1.263).	90

Table 24:	The execution times of <code>SMS::Cable::CalcDynamics()</code> subfunctions for 1.0 second of simulation time (SMS API v0.1.263).	90
Table 25:	The execution times of the <code>SMS::Payload::CalcDynamics()</code> subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).	91
Table 26:	The execution times of <code>SMS::RigidBody::CalcDynamics()</code> subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).	91
Table 27:	The execution times of <code>SMS::RigidBody::CalcForces()</code> subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).	91
Table 28:	The execution times of <code>SMS::Payload::CalcDynamics()</code> subfunctions for 1.0 second of simulation time (SMS API v0.1.263).	92
Table 29:	The execution times of <code>SMS::RigidBody::CalcDynamics()</code> subfunctions for 1.0 second of simulation time (SMS API v0.1.263).	92
Table 30:	The execution times of <code>SMS::RigidBody::CalcForces()</code> subfunctions for 1.0 second of simulation time (SMS API v0.1.263).	92

This page intentionally left blank.

1 Introduction

Dynamic Systems Analysis Ltd. (DSA) has worked with DRDC Atlantic to develop the Ship Mechanical Systems Application Programming Interface (SMS API). The software has been developed to provide the capability to simulate ship-based mechanical systems. Such systems could include shipborne boomcranes, cables, winches, and general payloads (AUVs, small rescue crafts, etc.). Fidelity and accuracy of the algorithms developed has been prioritized over execution speed. Contact dynamics between objects such as the payload and the naval frigate are also of primary importance. More information on the development of the SMS API can be found in [1].

The following report reviews tasks the authors have completed to improve and augment the capabilities of the SMS API. The SMS API developments reviewed herein have been focused by the need to assess operations such as the launch and recovery of a small vessel from a naval frigate as well as the towing of an unpowered vessel through a seaway by a naval frigate. This report summarizes advances made to the SMS API, and demonstrates the use of the SMS API with the aforementioned simulation scenarios.

1.1 Background

The SMS API is a C++ simulation library that provides a user with the ability to create high fidelity simulations of ship mechanical simulations. There are four main pieces of equipment the SMS API is able to simulate: boomcranes, slack/taut cables, general rigid bodies (i.e. payloads), and winches (to pay in/out cable). The SMS API consists of four main classes:

- `SMS::BoomCrane`
- `SMS::Cable`
- `SMS::Payload`
- `SMS::Winch`

The `SMS::Payload` class incorporates a collision resolution system to detect and resolve collisions between itself and some external object, such as a naval frigate hull or deck.

1.2 Report overview

As mentioned above, this report in part presents improvements to the SMS API that enable the simulation of two scenarios:

- the launch and recovery of a small vessel from a naval frigate, and,
- the towing of a small vessel in a seaway by a naval frigate.

In order to simulate these complex scenarios, further development of the SMS API was required. First, the `SMS::Payload` required a hydrodynamics model to capture the interactions between itself and the seaway. Second, a facility was required to enable the scripting of simulation events such as defining boomcrane motions, attach/detaching cables, winching cables in and out, etc. Third, improvements to the contact dynamics capabilities were required. The contact modeling capabilities in the SMS API lacked robustness and needed an improved software architecture to manage the simulation of convex decomposed concave objects. Lastly, the use of an implicit numerical integrator was pursued to mitigate undesirable high frequency effects in the cable model which causes explicit solvers to run at small time steps and thus slower simulation speeds.

1.2.1 Payload hydrodynamics improvements overview

To determine the hydrodynamic and buoyancy forces that are to be applied to a payload, it is necessary to describe the motion of the fluid relative to the payload. For this work, this information is obtained using ShipMo3D's `ShipMo3D::DeepSeaway` library [2]. The seaway model describes the fluid domain used by ShipMo3D's ship and the `SMS::Payload`. Fixed and translating seaways as well as regular and irregular wave models are modelled by the `ShipMo3D::DeepSeaway` DLL. The seaway model is accurate for deep water conditions, which requires a depth greater than half the wavelength of the oncoming ocean waves. The SMS API interacts with the `ShipMo3D::DeepSeaway` library via a C# DLL. For the SMS API to call the C# DLL, it was necessary to compile the SMS API using Microsoft .Net's Common Language Runtime (CLR).

Section 2.1 outlines how the `SMS::Payload` uses the information provided from the `ShipMo3D::DeepSeaway` library to evaluate the hydrodynamics forces acting on it and provides the theoretical background. Section 3.1 reviews the validation cases that were simulated to test the payload hydrodynamics functionality in the SMS API.

1.2.2 Director class overview

To create and simulate complex scenarios efficiently, a new class called the `SMS::Director` class was added to the SMS API. This enhancement of the SMS API was built to facilitate the rapid creation and management of simulation scenarios. The `SMS::Director` class was designed to manage a queue of commands, which each SMS API `SMS::SimObject` will execute at various points throughout the simulation. The

`SMS::Director` is tasked with ensuring each `SMS::SimObject` carries out the set of instructions defined in the script. Section 2.2 discusses its implementation as well as provides instructions on how to interact with it.

1.2.3 Contact dynamics improvements overview

Ship-borne crane operations, like maneuvering a payload suspended on a cable while in rough sea states, could potentially see the payload impacting or making contact with the ship deck, hull, transom, or other obstacles. To mitigate and manage the risk of these operations, numerical simulations are often used to evaluate the safety of multiple potential scenarios and operating procedures. To realistically simulate these scenarios in the SMS API, it is necessary to:

- determine the minimum separation distance (MSD) and minimum translational distance (MTD) between the `SMS::Payload` and external objects in its path,
- to detect collisions when they occur, and
- to determine the contact forces with a reasonable level of accuracy.

These capabilities had been previously implemented in the SMS API. Documentation and discussion of the methods and models used to accomplish these tasks have been reported in [1]. However, the existing code used for MTD/MSD determination and volume of interference determination to determine the contact force would frequently cause simulations to destabilize. In addition, simulations executed prohibitively slowly for practical use. The contact dynamics code responsible for these issues was re-visited to improve execution speed and simulation stability.

Some of the highlights of the work completed are:

- a new collision detection infrastructure to efficiently and cleanly handle convex decomposed geometries,
- Expanded Polytope Algorithm (EPA) was implemented to determine the minimum translational distance (MTD) or penetration depth,
- exploitation of temporal coherence for GJK to speed up collision detection,
- creation of a new set of contact dynamics simulations to test the convex decomposed collision detection capabilities and identify bugs,
- bug fixes, exception handling and general improvements to increase simulation stability and robustness,
- code profiling to identify inefficiencies and make performance improvements.

A description of the improvements made to the code over the time period covered by this report can be found in Section 2.3. This includes a discussion of the new collision detection infrastructure, the EPA, and how temporal coherence is exploited with GJK. A set of four simulation cases which ensures the proper functioning of the new collision detection infrastructure can be found in Section 3.2. Discussion on the scalability of the algorithms used for contact dynamics as well as the results of code optimisation efforts can be found in Section 2.3.4.

1.2.4 Launch and recovery and towing simulation overview

As discussed above, the required outcome of the SMS API software developments presented here was the creation of a simulation of the launch and recovery of a small vessel from a naval frigate and a simulation of a small vessel being towed by a naval frigate. These simulations were developed progressively over the course of the work. As new functionality was added or improved, more complexity was added to these simulations. Section 4 discusses a simplified version of the Launch and Recovery simulation, the work done to improve the execution speed of the simulation and the final version for which an animated visualisation was produced and submitted to DRDC along with this report. Section 5 discusses a simplified version of the towing of a small vessel simulation, followed by the final version of the towing simulation, titled Tuna Clipper Towing, for which an animated visualisation was produced and submitted to DRDC along with this report.

2 SMS API development

This section describes the theory and technical details of the SMS API improvements covered in this report. Section 2.1 discusses the theory behind the `SMS::Payload` hydrodynamic model. Section 2.2 describes the design and usage of the `SMS::Director` scripting class. Section 2.3 describes the improvements made to the `SMS::Payload` contact resolution system. Finally, section 2.4 discusses the theory behind the Generalised- α implicit numerical integrator for the `SMS::Cable`.

2.1 Payload hydrodynamics

2.1.1 Newton-Euler equation of motion of an `SMS::Payload`

The Newton-Euler equations of motion of an `SMS::Payload` as described in [1] state that the relationship between its acceleration and the force applied to it is:

$${}^B\mathbf{f} = {}^B\mathbf{M} {}^B\mathbf{a}_{cg} \quad (1)$$

where ${}^B\mathbf{f}$ is a 6 degree of freedom (DOF) spatial vector describing the force and moment applied about each body-fixed frame axis of the `SMS::Payload`, ${}^B\mathbf{M}$ is the `SMS::Payload`'s rigid body mass matrix described about the body-fixed frame, which is located at the center of mass, and ${}^B\mathbf{a}_{cg}$ is the spatial acceleration vector that is the time derivative of the body velocity ${}^B\mathbf{v}_{cg}$, describing the linear and angular acceleration of the `SMS::Payload` about each body-fixed frame axes. Because the frame of reference is fixed to the body and is moving with respect to time, ${}^B\mathbf{a}_{cg}$ takes the form:

$${}^B\mathbf{a}_{cg} = {}^B\dot{\mathbf{v}}_{cg} + {}^B\omega_B {}^B\mathbf{v}_{cg} \quad (2)$$

where ${}^B\omega_B$ is the angular velocity about the body-fixed frame. More information on rigid body dynamics can be seen in [1].

The mass matrix generally takes the form of:

$${}^B\mathbf{M} = \begin{bmatrix} {}^B\mathbf{m} & 0 \\ 0 & {}^B\mathbf{I} \end{bmatrix} \quad (3)$$

where

$${}^B\mathbf{m} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}, \quad (4)$$

$${}^B\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}, \quad (5)$$

and m is the mass of the object, I_{xx} , I_{yy} , and I_{zz} are the mass moments of inertia about the inertial frame axis, and I_{xy} , I_{xz} , and I_{yz} are the products of inertia.

The force applied to the `SMS::Payload` is the sum of all forces applied to it. This includes cable forces, contact forces, gravitational, and hydrodynamics forces:

$${}^B\mathbf{f} = \sum_{i=0}^{N_{cab}} {}^B\mathbf{f}_{CAB}^i + \sum_{i=0}^{N_{con}} {}^B\mathbf{f}_{CON}^i + {}^B\mathbf{f}_G + {}^B\mathbf{f}_M \quad (6)$$

where ${}^B\mathbf{f}_{CAB}^i$ is the force from the i^{th} attached cable, ${}^B\mathbf{f}_{CON}^i$ is the force from the i^{th} contact, ${}^B\mathbf{f}_G$ is the force due to gravity, ${}^B\mathbf{f}_M$ is the hydrodynamic force, and N_{cab} and N_{con} are the number of attached cables and contacts, respectively.

This work addresses the modelling of the hydrodynamic forces only. The entire `SMS::Payload` object surface is discretised using a convex closed polygon mesh (polyhedron) representation. Individual pressures acting on each polygon panel are resolved and accrued over the entire polyhedron, which is a discrete form of a surface integral over the surface of the object.

2.1.2 Object surface discretisation

The fluid force on the `SMS::Payload` is determined as the integral of the fluid pressures acting on the payload. The integration of fluid pressures acting on the immersed `SMS::Payload` surface is accomplished by discretising the surface using a polygon mesh. The fluid forces acting on each polygon of the mesh are accumulated to obtain the total fluid force acting on the `SMS::Payload`. For example, a 3×1 linear force \mathbf{f} acting on the object is:

$$\mathbf{f} = \sum_{j=0}^{N_{poly}} \mathbf{f}_j = \sum_{p=0}^{N_{poly}} -p(\mathbf{C}_{p,j}) A_j \hat{\mathbf{f}}_j \quad (7)$$

where N_{poly} is the number of polygons on the polyhedron, \mathbf{f}_j is a 3×1 vector describing the force applied to the polygon face j , the pressure, $p(\mathbf{C}_{p,j})$, is a function of the location of the polygon face centroid $\mathbf{C}_{p,j}$, A_j is the area of polygon j , and $\hat{\mathbf{f}}_j$ is the force direction at the centroid of polygon j .

The forces applied to each polygon face also produce a moment on the object:

$$\boldsymbol{\tau} = \sum_{j=0}^{N_{poly}} \boldsymbol{\tau}_j = \sum_{j=0}^{N_{poly}} \mathbf{C}_{p,j} \times \mathbf{f}_j \quad (8)$$

where $\boldsymbol{\tau}$ is the 3×1 vector describing the total moment applied to the object, and $\boldsymbol{\tau}_j$ is the torque applied to the object from the force \mathbf{f}_j applied to polygon j .

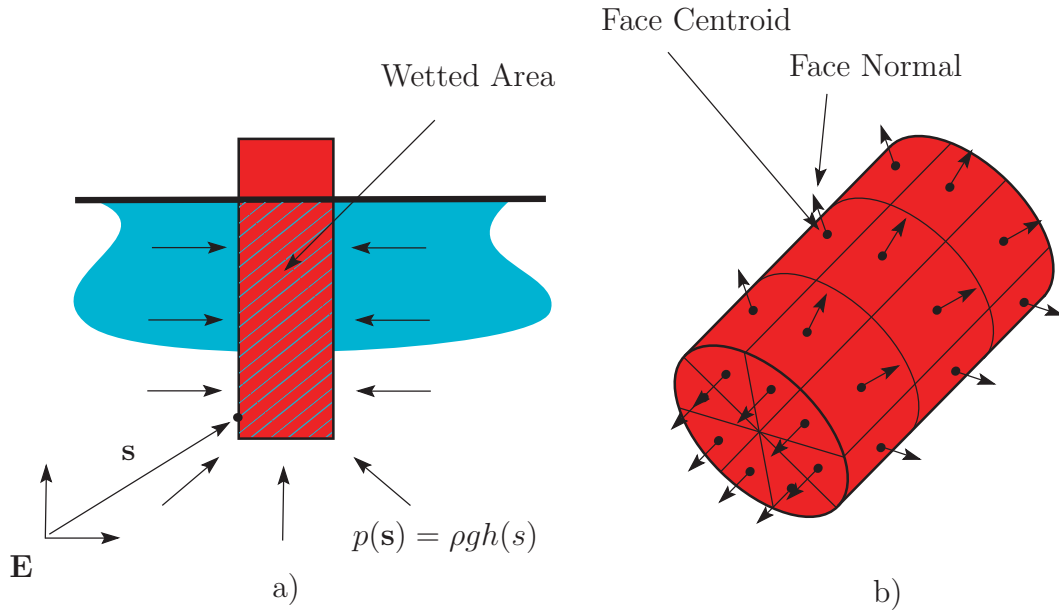


Figure 1: a) The fluid pressure distribution over the wetted surface of a cylinder from buoyancy in calm water. b) The polygonal mesh discretisation used for the surface integral of the pressure field over the body surface.

For floating objects in a seaway, some polygons will be exposed to water while others are exposed to air. If a significant amount of the object area is exposed to the air, the resulting forces from wind loading may need to be considered for accurate simulation. However, in many cases hydrodynamic loading will dominate the response of a payload and wind loading can safely be neglected. It was decided to neglect wind loading on the payload in order to concentrate on first accurately estimating the dominant hydrodynamic loading on a payload. An advantage of using the polyhedral representation of the payload is that only the hydrodynamic forces acting on the wetted polygons are considered. In a similar way, fluid forces from the air could certainly be taken into account. A polygon is considered to be wet if its centroid is below the water surface. The wetted area of a polyhedron is approximated as the sum of the areas of the wetted polygons. The finer the polygon mesh, the more accurate the wetted area calculations. It should be noted that as the number of polygons in a mesh grows, the number fluid force calculations required grows linearly. As such, the time required to determine the fluid forces can quickly become a significant bottleneck to any potential benefits that could be gained from increased accuracy.

2.1.3 Morison equation

Morison *et. al* proposed that the forces acting on a submerged body by a flowing fluid can be reasonably represented as the sum of drag and inertial forces [3, 4]. The

Morison equation describes the hydrodynamic force as:

$$\mathbf{f}_m = \mathbf{f}_d + \mathbf{f}_{f-k} + \mathbf{f}_{inertial} \quad (9)$$

where \mathbf{f}_d , \mathbf{f}_{f-k} , $\mathbf{f}_{inertial}$ are the 3×1 drag force, Froude-Krylov force, and inertial or added mass force respectively. The objective of the proceeding is to examine how the SMS API determines the Morison load acting on each panel of a discretized payload.

2.1.4 Description of the fluid domain

To calculate the Morison loading on a discretized payload, the flow field must be described. The SMS API relies on knowing the velocity of the flow at any point in the simulation space. The velocity at a given point, can be described in space and time as [5]:

$$\mathbf{V}(\mathbf{r}, t) = \hat{\mathbf{i}}u(\mathbf{r}, t) + \hat{\mathbf{j}}v(\mathbf{r}, t) + \hat{\mathbf{k}}w(\mathbf{r}, t) \quad (10)$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$ are the direction unit vectors about the 3 Cartesian frame axes and $u(\mathbf{r}, t)$, $v(\mathbf{r}, t)$, $w(\mathbf{r}, t)$ are the absolute fluid vector components about the $\hat{\mathbf{X}}$, $\hat{\mathbf{Y}}$, $\hat{\mathbf{Z}}$ axes, respectively, at a point \mathbf{r} in space and time, t . The acceleration of the fluid at a location \mathbf{r} in space and time, t , is described as the rate of change of $\mathbf{V}(\mathbf{r}, t)$ in time:

$$\mathbf{a}(\mathbf{r}, t) = \frac{d\mathbf{V}(\mathbf{r}, t)}{dt} = \hat{\mathbf{i}}\frac{du(\mathbf{r}, t)}{dt} + \hat{\mathbf{j}}\frac{dv(\mathbf{r}, t)}{dt} + \hat{\mathbf{k}}\frac{dw(\mathbf{r}, t)}{dt} \quad (11)$$

Assuming an irrotational fluid and an inviscid flow, the fluid field can be described using the velocity potential function $\phi(\mathbf{r}, t)$ where $\mathbf{V}(\mathbf{r}, t) = \nabla\phi(\mathbf{r}, t)$ leading to [5]:

$$u(\mathbf{r}, t) = \frac{\partial\phi(\mathbf{r}, t)}{\partial x}, v(\mathbf{r}, t) = \frac{\partial\phi(\mathbf{r}, t)}{\partial y}, w(\mathbf{r}, t) = \frac{\partial\phi(\mathbf{r}, t)}{\partial z}. \quad (12)$$

Bernoulli's equation is described as [5]:

$$\rho\frac{\partial\phi(\mathbf{r}, t)}{\partial t} + p(\mathbf{r}, t) + \frac{1}{2}\rho V(\mathbf{r}, t)^2 + \rho gz(\mathbf{r}, t) = C \quad (13)$$

where $V(\mathbf{r}, t) = |\nabla\phi(\mathbf{r}, t)|$ is the magnitude of the fluid velocity at a point \mathbf{r} in space, ρ is the fluid density, $p(\mathbf{r}, t)$ is the pressure at a point \mathbf{r} in space and time t , $z(\mathbf{r}, t)$ is the elevation of a point \mathbf{r} in space, $\frac{\partial\phi(\mathbf{r}, t)}{\partial t}$ is the partial derivative with respect to time of $\phi(\mathbf{r}, t)$ at a point \mathbf{r} in space, g is gravitational acceleration and C is a constant [5].

The constant C for Bernoulli's equation is found by sampling the fluid at some point in the fluid where the pressure, fluid velocity, acceleration and depth from surface is known, such as at the ocean surface, as illustrated in Figure 2.

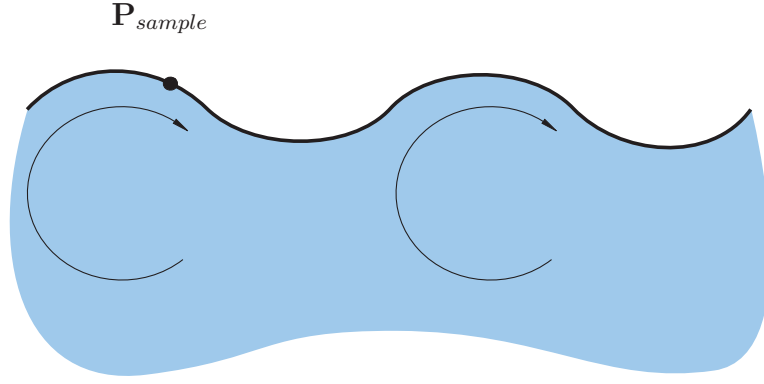


Figure 2: Evaluation of the constant for Bernoulli's equation for a deep seaway by sampling the fluid state at a point \mathbf{P}_{sample} on the surface where pressure and velocities are known.

2.1.5 The Froude-Krylov force

The 3×1 Froude-Krylov force is the result of the pressure field of the fluid acting on the body:

$$\mathbf{f}_{f-k} = - \int_{S_w} p(\mathbf{s}) \hat{\mathbf{n}}(\mathbf{s}) ds \quad (14)$$

where S_w is the wetted surface, $p(\mathbf{s})$ is the pressure of the fluid at some point \mathbf{s} on the surface, ds is the differential area and $\hat{\mathbf{n}}(\mathbf{s})$ is the surface normal at a point \mathbf{s} on the surface of the object. The Froude-Krylov force is the surface integral of the pressure field of the undisturbed seaway acting on the body [4]. The Froude-Krylov force does not include diffraction effects; it assumes that the fluid is not disturbed by the presence of the body. Diffraction forces can be assumed to be zero if the body length is small relative to the incident wavelength. When no waves are present, the Froude-Krylov force is equal to the buoyancy force acting on the body.

As mentioned above, the SMS API uses a discretized mesh to represent the payload hull in the seaway. To calculate the Froude-Krylov force acting on this payload equation 14 becomes:

$$\mathbf{f}_{f-k} = - \sum_{j=0}^{N_{poly}} p(\mathbf{C}_{p,j}) A_j \hat{\mathbf{n}}_j \quad (15)$$

where A_j is the area of polygon j and $\hat{\mathbf{n}}_j$ is the polygon face normal direction. For the case where the fluid is not still, such as in a `ShipMo3D::DeepSeaway`, the pressure field is also a function of the fluid motion, which can be described using the velocity potential function for irrotational inviscid flows. The pressure of the fluid at a point

$C_{p,j}$ in space is derived from Bernoulli's equation:

$$p(\mathbf{C}_{p,j}) = \rho C - \rho \frac{\partial \phi(\mathbf{C}_{p,j})}{\partial t} - \frac{1}{2} \rho U(\mathbf{s})^2 - \rho g h(\mathbf{C}_{p,j}) \quad (16)$$

where C is a constant determined by evaluating equation 13 at some point in the Seaway where all other terms are known, $U(\mathbf{C}_{p,j})$ is the scalar velocity of the fluid at the centroid of polygon j , $\mathbf{C}_{p,j}$, $h(\mathbf{C}_{p,j})$ is the depth under the water surface at the centroid of polygon j .

For a still object in still water, the pressure is simply equal to equation 13 where the velocity terms $V(\mathbf{C}_{p,j})$ and $\frac{\partial \phi(\mathbf{C}_{p,j})}{\partial t}$ are zero. Therefore, the pressure of the fluid is a function of water depth alone, $p(\mathbf{C}_{p,j}) = \rho g h(\mathbf{C}_{p,j})$. Figure 1 illustrates the case where the pressure is a function of fluid depth alone.

2.1.6 Hydrodynamic drag forces

The drag force arises when a body moves relative to the fluid in which it is immersed as indicated in Figure 3. Hydrodynamic drag is a complicated nonlinear phenomenon that is not easily generalized. Approaches such as computational fluid dynamics (CFD) resolve the fluid domain and the drag forces reasonably consistently for arbitrary geometries, but simulation or analysis using these techniques can be a complex process and is also computationally expensive. Rather than relying on generalized techniques for determination of drag, empirical data exists that report drag coefficients for various shapes. The drag force acting on an object is based typically on the frontal area of the object in the direction of the flow:

$$f_d = \frac{1}{2} \rho C_d A_{proj} v^2. \quad (17)$$

where C_d is the drag coefficient, A_{proj} is the frontal projected area in the direction of relative fluid flow and v is the relative fluid velocity.

For the purposes of the SMS API it is necessary to estimate the drag forces acting on a given payload such that many types of operations can be simulated rapidly. Since CFD is not appropriate for this, an approach relying on empirical drag coefficients was implemented. Drag coefficient data for various shapes with fluid flowing across them in many different directions are available in literature. Here, objects are assumed to have the same drag coefficient in the positive and negative axes directions (e.g. the object has the same drag coefficient in the positive X and negative X directions), which simplifies the relationship between the drag force and relative velocity of the fluid to:

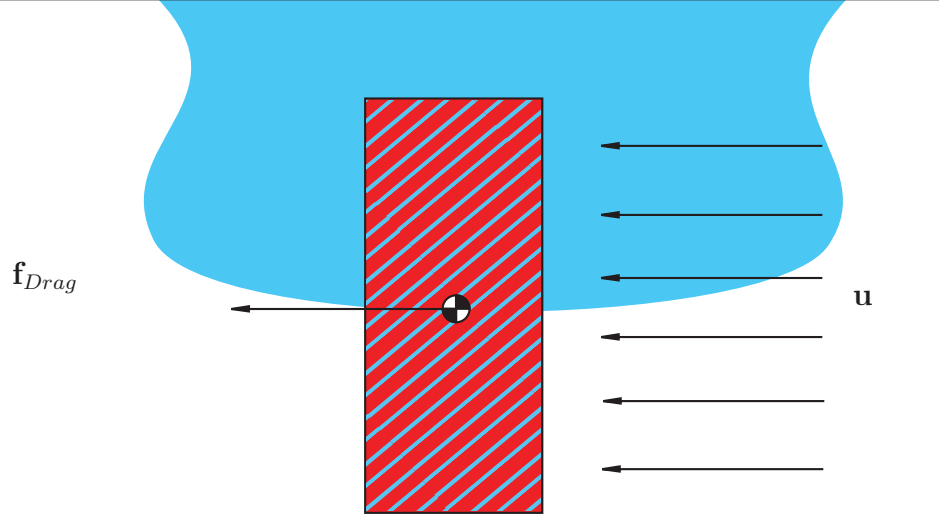


Figure 3: Experimental setup for the variable buoyancy validation test

$$\mathbf{f}_d = \sum_{j=0}^{N_{poly}} \left\{ \begin{array}{l} \frac{1}{2}\rho A_{j,\hat{\mathbf{x}}} C_{d,\hat{\mathbf{x}}} (\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})) |\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})| \cdot \hat{\mathbf{i}} \\ \frac{1}{2}\rho A_{j,\hat{\mathbf{y}}} C_{d,\hat{\mathbf{y}}} (\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})) |\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})| \cdot \hat{\mathbf{j}} \\ \frac{1}{2}\rho A_{j,\hat{\mathbf{z}}} C_{d,\hat{\mathbf{z}}} (\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})) |\mathbf{u}(\mathbf{C}_{p,j}) - \mathbf{v}(\mathbf{C}_{p,j})| \cdot \hat{\mathbf{k}} \end{array} \right\} \quad (18)$$

where $\mathbf{u}(\mathbf{C}_{p,j})$ is the absolute fluid relative velocity at the centroid of polygon j , $\mathbf{v}(\mathbf{C}_{p,j})$ is the absolute velocity of the object at the centroid of polygon j , A_j is the area of polygon j , $C_{d,\hat{\mathbf{x}}}, C_{d,\hat{\mathbf{y}}}, C_{d,\hat{\mathbf{z}}}$ are the overall drag coefficients for each of the 3 Cartesian flow directions and $A_{j,\hat{\mathbf{x}}}$, $A_{j,\hat{\mathbf{y}}}$ and $A_{j,\hat{\mathbf{z}}}$ are the projected areas of polygon i in the YZ plane, ZX plane and XY plane respectively:

$$A_{j,\hat{\mathbf{x}}} = A_j \hat{\mathbf{n}}_j \cdot \hat{\mathbf{i}} \quad (19)$$

$$A_{j,\hat{\mathbf{y}}} = A_j \hat{\mathbf{n}}_j \cdot \hat{\mathbf{j}} \quad (20)$$

$$A_{j,\hat{\mathbf{z}}} = A_j \hat{\mathbf{n}}_j \cdot \hat{\mathbf{k}} \quad (21)$$

The drag coefficient is found by experiment for a single geometry and flow direction, since objects have different drag coefficients acting on them depending on geometry and relative flow direction. For example, the drag coefficient for fluid flow across a cylinder would be different than the flow of fluid along it as in Figure 4. Aspect ratios of the geometry influence the drag coefficient as well. For the case of a circular cylinder whose central axis is aligned with the $\hat{\mathbf{z}}$ axis, the coefficients of drag in the $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ directions would be identical and different from the coefficient of drag in the $\hat{\mathbf{z}}$ axis.

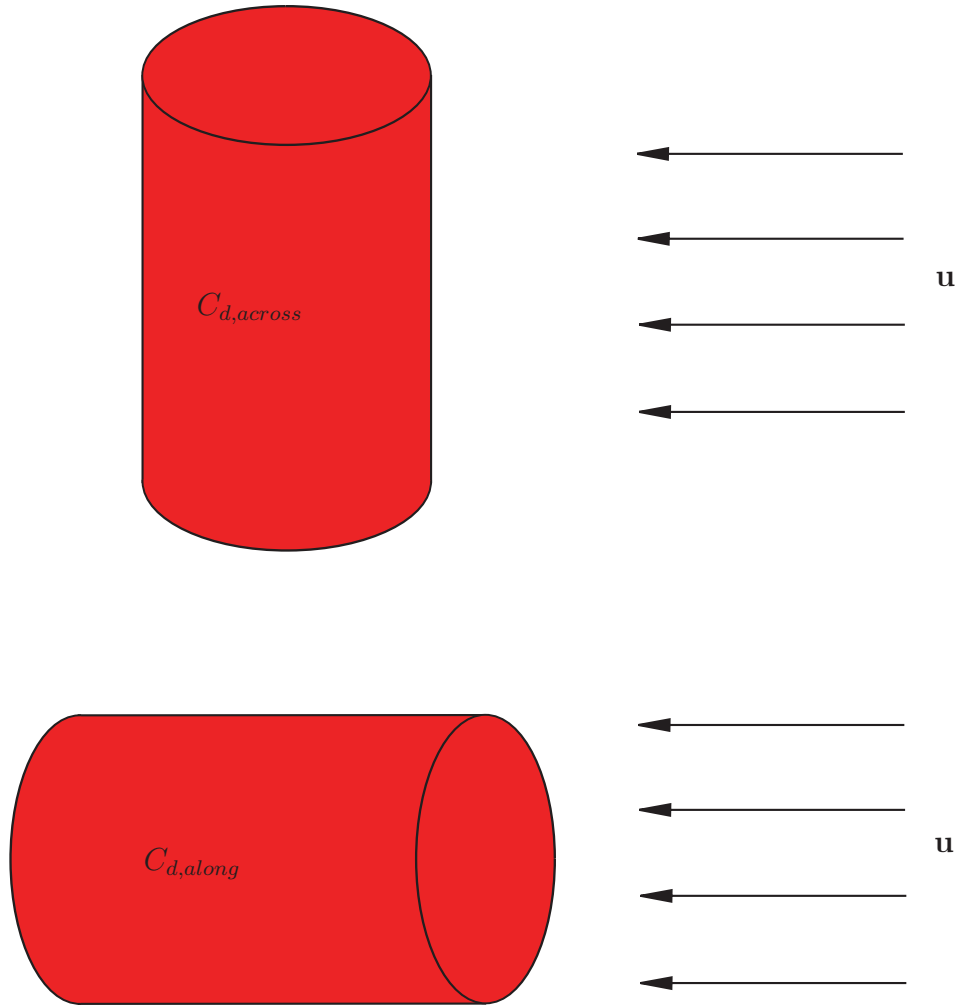


Figure 4: Normal and tangential drag coefficients for a cylinder

The absolute velocity of the object, $\mathbf{v}(\mathbf{C}_{p,i})$, at the centroid of polygon i can be separated into two components:

$$\mathbf{v}(\mathbf{C}_{p,i}) = \mathbf{v}_{body} + \omega_{body} \times \mathbf{C}_{p,i} \quad (22)$$

where \mathbf{v}_{body} is the linear absolute velocity of the body described about the body-fixed frame, and the ω_{body} angular velocity of the body. The angular velocity component

of the drag force inherent in Equation 18 can then be separated leading to:

$$\mathbf{f}_d = \sum_{i=0}^{N_{poly}} \left\{ \begin{array}{l} \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{x}}}(\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body})|\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body}|\hat{\mathbf{i}} \\ \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{y}}}(\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body})|\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body}|\hat{\mathbf{j}} \\ \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{z}}}(\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body})|\mathbf{u}(\mathbf{C}_{p,i}) - \mathbf{v}_{body}|\hat{\mathbf{k}} \end{array} \right\} \quad (23)$$

$$- \sum_{i=0}^{N_{poly}} \left\{ \begin{array}{l} \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{x}}}(\omega_{body} \times \mathbf{C}_{p,i})|\omega_{body} \times \mathbf{C}_{p,i}|\hat{\mathbf{i}} \\ \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{y}}}(\omega_{body} \times \mathbf{C}_{p,i})|\omega_{body} \times \mathbf{C}_{p,i}|\hat{\mathbf{j}} \\ \frac{1}{2}\rho A_{i,proj} C_{d,\hat{\mathbf{z}}}(\omega_{body} \times \mathbf{C}_{p,i})|\omega_{body} \times \mathbf{C}_{p,i}|\hat{\mathbf{k}} \end{array} \right\}$$

2.1.6.1 Limitations

The above methodology for estimating the drag effect on payloads estimates the drag moments using the translational drag coefficients. This is equivalent to finding the center of pressure on a face of a submerged object, calculating the drag force on that face, and computing the drag moment using the center of pressure location relative to the center of mass and total drag force. This methodology is convenient because arbitrary geometry with known rough translational drag coefficients can be readily simulated in 6 DOF. However, the rotational drag effect is an approximation as translational drag coefficients are developed considering only translational motion. Consider Figure 5; in this case, the discretized drag moment as outlined in equation 23 is applied to the body using equation 8. In this figure it is assumed that the objects have the same depth. Considering object a), the flow across the tip of this narrow object due to the rotation of the object can be considered as linear movement across that object. Now considering object b), the flow of fluid around the wider object takes a longer path than if the body was in pure translation. This example emphasizes the translational drag coefficients are not completely adequate to estimate drag effects in rotation.

The skin friction effect is lumped into the translational drag coefficient with this methodology. A consequence of this is that for bodies of revolution (e.g., cylinders or spheres) that are rotating about their axis of revolution, no resistive force is applied to the object to prevent spinning. In many applications, skin friction or viscous effects are not critical to accurately determining the overall system dynamics and can be safely neglected. However, some skin friction or viscous drag is convenient in numerical simulation to prevent destabilisation of simulations. This phenomenon will be investigated further to find a method of including it in the `SMS::Payload` hydrodynamic loading model.

An alternate method that could be investigated for use in the SMS API payload model, would be the specification of linear and quadratic drag coefficients. This method completely decouples the drag calculations between the rotational and translational degrees of freedom by using a set of independently determined drag coefficients for each degree of freedom. However, a significant disadvantage of this method

is that CFD or physical experiments are required to determine these coefficients. In addition, this method would not apply to objects that are dynamically submerging such as a small craft.

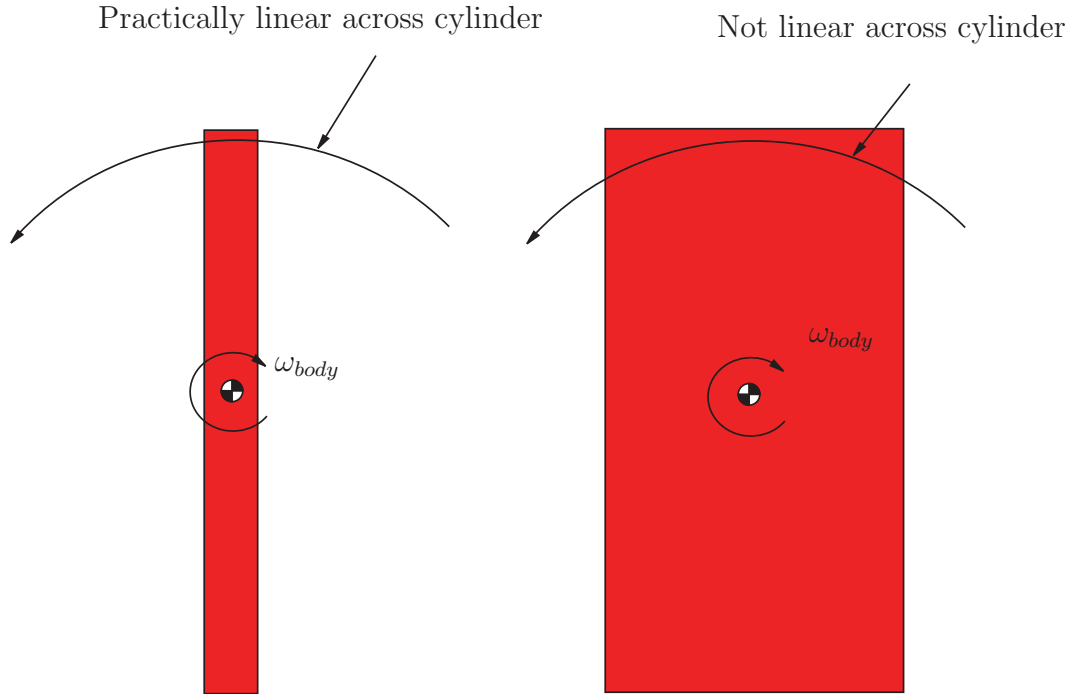


Figure 5: The path of the flow of fluid across a) a slender rotating object and b) a non-slender rotating object.

2.1.7 Added Mass forces

When an immersed body is accelerated relative to a fluid, the object must also accelerate some fluid around it. The pressure force applied to the object as a result of the acceleration of the surrounding fluid affects the dynamics of the object if the density of the fluid is significant. This effect is usually referred to as virtual or added mass, since it has the effect of increasing the apparent mass of the accelerating object. For fully submerged rigid bodies in flows that are not accelerating, the added mass is expressed as a symmetric 6×6 positive definitive added mass matrix that is added to the body mass such that equation 1 becomes:

$${}^B\mathbf{f} = ({}^B\mathbf{M} + {}^B\mathbf{A})\mathbf{a}_{cg} \quad (24)$$

where ${}^B\mathbf{A}$ is the added mass matrix. In flows that are accelerating, such as an object that is in a seaway with waves present, the acceleration of the surrounding must be considered. Morison's approach accounts for acceleration of a slender body relative to

the fluid motion. In addition, the added mass effect is typically non-dimensionalised such that it is expressed in terms of an added mass coefficient that is multiplied by the weight of the displaced fluid. Added mass coefficients for Morison's approach are defined for simple cross-sectional geometries.

In the SMS API, to develop the the linear fluid inertial force acting on a payload, first consider that the fluid inertial force can be expressed as:

$$\mathbf{f}_i = \sum_{j=0}^{N_{poly}} \left\{ \begin{array}{l} C_{a,\hat{\mathbf{x}}} \rho_w V_{disp,j} (\dot{\mathbf{u}}(\mathbf{C}_{t,j}) - \mathbf{a}(\mathbf{C}_{t,j})) \cdot \hat{\mathbf{i}} \\ C_{a,\hat{\mathbf{y}}} \rho_w V_{disp,j} (\dot{\mathbf{u}}(\mathbf{C}_{t,j}) - \mathbf{a}(\mathbf{C}_{t,j})) \cdot \hat{\mathbf{j}} \\ C_{a,\hat{\mathbf{z}}} \rho_w V_{disp,j} (\dot{\mathbf{u}}(\mathbf{C}_{t,j}) - \mathbf{a}(\mathbf{C}_{t,j})) \cdot \hat{\mathbf{k}} \end{array} \right\} \quad (25)$$

where ρ_w is the fluid density, $V_{disp,j}$ is the displaced fluid volume of a tetrahedron (for triangular polygons) created by the surface polygon j and the center of the body-frame, $\dot{\mathbf{u}}(\mathbf{C}_{t,j})$ is the absolute fluid acceleration at the centroid of a tetrahedron created by polygon j and the body frame origin, $\mathbf{a}(\mathbf{C}_{t,j})$ is the body absolute acceleration at the centroid of a tetrahedron created by polygon j and the origin of the body frame, $C_{a,\hat{\mathbf{x}}}$, $C_{a,\hat{\mathbf{y}}}$ and $C_{a,\hat{\mathbf{z}}}$ are the added mass coefficients in the 3 Cartesian directions.

The acceleration of the body $\mathbf{a}(\mathbf{C}_{t,j})$ at the centroid of a tetrahedron created by polygon j and the origin of the body frame is determined from the forces applied to the object. Typically, the component of added mass force proportional to the body absolute acceleration in equation 25 is brought to the other side of equation 1 and lumped in with the physical mass while the component of added mass proportional to absolute fluid acceleration remains lumped in with other forces applied to the body. This facilitates an explicit evaluation for time domain simulation such that equation 1 becomes equation 24, where ${}^B\mathbf{M}_a$ is:

$${}^B\mathbf{A} = \begin{bmatrix} \mathbf{m}_a & 0 \\ 0 & \mathbf{I}_a \end{bmatrix}. \quad (26)$$

and \mathbf{m}_a is mass of the fluid that is accelerated when the body is accelerated and \mathbf{I}_a describes the moment of inertia of the distributed added mass. The added mass matrix takes the form:

$$\mathbf{m}_a = \sum_{i=0}^{N_{poly}} \mathbf{m}_{a,i} = \sum_{i=0}^{N_{poly}} \begin{bmatrix} C_{a,\hat{\mathbf{x}}} \rho_w V_{disp,i} & 0 & 0 \\ 0 & C_{a,\hat{\mathbf{y}}} \rho_w V_{disp,i} & 0 \\ 0 & 0 & C_{a,\hat{\mathbf{z}}} \rho_w V_{disp,i} \end{bmatrix} \quad (27)$$

Moving the inertial force component to the other side of the Newton-Euler equation leaves the inertial force component that is dependent only on the absolute fluid

velocity as:

$$\mathbf{f}_i = \sum_{i=0}^{N_{poly}} \left\{ \begin{array}{l} C_{a,\hat{\mathbf{x}}}\rho_w V_{disp,i} \dot{\mathbf{u}}(\mathbf{C}_{t,i}) \cdot \hat{\mathbf{i}} \\ C_{a,\hat{\mathbf{y}}}\rho_w V_{disp,i} \dot{\mathbf{u}}(\mathbf{C}_{t,i}) \cdot \hat{\mathbf{j}} \\ C_{a,\hat{\mathbf{z}}}\rho_w V_{disp,i} \dot{\mathbf{u}}(\mathbf{C}_{t,i}) \cdot \hat{\mathbf{k}} \end{array} \right\} \quad (28)$$

In the SMS API the added mass moment of inertia matrix \mathbf{I}_a is approximated by summing the mass moments of inertia of the water displaced by a tetrahedral volume formed by every polygon and the center of gravity of the object. To simplify this calculation, the mass of each tetrahedron is lumped at its centroid and using the parallel axis theorem following the relation for the mass moment of inertia about the centroid of the body [6]:

$$\mathbf{I}_a = \sum_{i=0}^{N_{poly}} \mathbf{m}_{a,i} (\mathbf{C}_{t,i} \cdot \mathbf{C}_{t,i} \hat{\mathbf{I}} + \mathbf{C}_{t,i} \otimes \mathbf{C}_{t,i}) \quad (29)$$

2.1.7.1 Limitations

The added mass moment of inertia for bodies of revolution, such as cylinders or spheres, about their axis will likely be negligible. The method discussed here distributes the added mass uniformly over the geometry of the object. This allows for a good representation of the added mass moment of inertia for slender objects such as long cylinders, however it will likely over-estimate the added mass moment of inertia in certain cases such as for bodies of revolution rotating about the axis of revolution. Due to the lack of available experimental data for objects with angular acceleration, accuracy of the method used for calculating added mass moment of inertia is difficult to verify. This limitation should be considered when using the model and more work should be completed for validation.

2.1.8 Using ShipMo3D to determine Added Mass for an SMS::Payload

In order to accurately model generic objects, alternative methods of determining the hydrodynamics have been investigated. Specifically, time was spent researching the use of potential flow theory to model these hydrodynamic forces. ShipMo3D is software developed by DRDC to help simulate the motions of ships in seaways under various environmental conditions [7]. A user manual and theory documentation for the software can be found in [2, 7, 8, 9]. The theory behind how ShipMo3D determines the added mass for different geometries is based on potential flow theory.

ShipMo3D can be used to determine the added mass coefficients for SMS::Payload geometry about 6 degrees of freedom. If the SMS::Payload is considered small compared

to the incident wavelengths, diffraction and radiation effects can be assumed negligible, and the resulting coefficients produced by ShipMo3D can be used. ShipMo3D has the ability to determine the added mass of an object under various wave conditions and constructs a hydrodynamic database. Because the `SMS::Payload` is always assumed to be small relative to the incident wavelengths, those added mass coefficients for zero frequency waves are of interest. One important assumption to note is that the added mass coefficients returned by ShipMo3D are for objects that have a constant waterline.

2.1.8.1 Comparing ShipMo3D results with the literature

In order to understand the procedure of determining the zero frequency added mass coefficients, 2 simple geometries were modelled in ShipMo3D and compared against experimental results from literature. The first object is a fully submerged $6 \times 6 \times 2$ box and the second is a $100 \times 6 \times 12$ also fully submerged box. These objects represent a non-slender and slender object respectively. Figures 6 and 7 show the hull line geometry representation read in by ShipMo3D and the meshed geometry that will be used in radiation and diffraction function to determine the added mass coefficients of restitution.

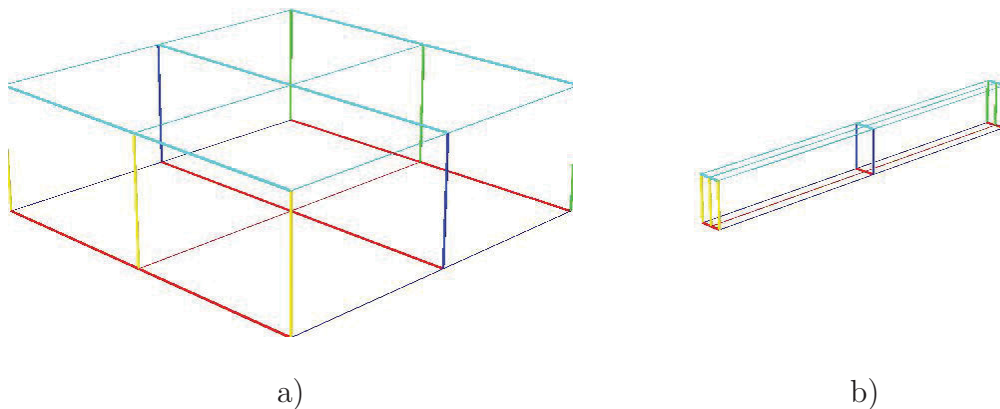


Figure 6: The hull lines as defined for ShipMo3D panelling method of a) $6 \times 6 \times 2$ box b) $100 \times 6 \times 12$ box.

The zero frequency linear added mass of these objects in heave are compared against those found in [10]. The expected added mass in heave of the $6 \times 6 \times 2$ box can be found in Table 2.3 from [10] as a rectangular block. Likewise, the expected added mass in heave of the $100 \times 6 \times 12$ box can also be found in Table 2.3 from [10] as the added mass per unit length of a rectangle.

The added mass moment of inertia about the roll degree of freedom was compared against those found in [4]. The expected added mass in roll for both boxes can be

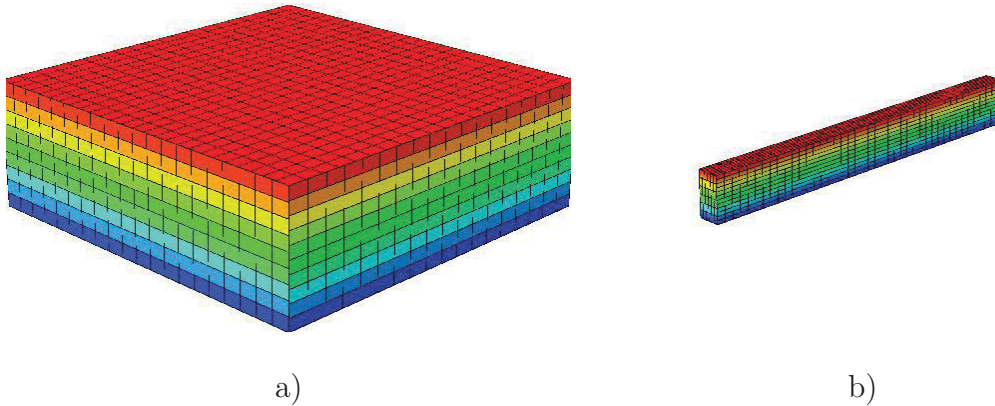


Figure 7: The resulting panelled hull created by ShipMo3D panelling method of a) $6 \times 6 \times 2$ box b) $100 \times 6 \times 12$ box.

found in Table D-1 of [4] for a rectangular cross-section. It should be noted that added mass for the $6 \times 6 \times 2$ box may not match expected closely as it is not slender. Tables 1 and 2 present the ShipMo3D zero frequency added mass coefficient output as well as the expected heave and roll degrees of freedom coefficients of added mass for the $6 \times 6 \times 2$ and $100 \times 6 \times 12$ geometries, respectively.

2.1.9 GPGPU parallelization of the hydrodynamic force calculation

NVIDIA's CUDA was chosen over OpenCL for investigating general purpose problem solving on GPUs (GPGPU) because documentation and hardware drivers were more accessible for it. The choice of CUDA limits the software to CUDA compatible cards made by NVIDIA. Due to a similarity in how both libraries work, a change over should not be difficult in the future should OpenCL become favorable. CUDA facilitates programming NVIDIA's GPUS and enables the parallelization of computational tasks where the same operation must be effectuated many times on data sets much like a traditional single-instruction multiple-data (SIMD) class parallel computer.

Though CUDA's computational architecture is similar to SIMD class parallel computers, it is more accurately classified as a single-instruction multiple-thread (SIMT) infrastructure where a data set is assigned a thread that executes a particular instruction set. These GPUs sacrifice single-thread execution speed in exchange for total computational throughput by executing many threads simultaneously. There are two fundamental measures of processor performance: task latency and total throughput. Task latency is the time required to initiate and complete a task, while throughput is the amount of work (number of tasks) that can be completed per unit time. This highlights the architectural design differences between CPUs and GPUS where CPUs

Table 1: The added mass coefficients found by ShipMo3D for the $6 \times 6 \times 2$ and the expected added mass coefficients in roll and heave.

	<i>ShipMo3D</i>	<i>Expected</i>
Surge	0.347	—
Sway	0.346	—
Heave	1.70	1.66
Roll	0.893	0.95
Pitch	0.893	—
Yaw	0.182	—

Table 2: The added mass coefficients found by ShipMo3D for the $100 \times 6 \times 12$ and the expected added mass coefficients in roll and heave.

	<i>ShipMo3D</i>	<i>Expected</i>
Surge	0.063	—
Sway	2.042	—
Heave	0.66	0.67
Roll	0.55	0.565
Pitch	.578	—
Yaw	1.73	—

have been designed in such a way to minimize the latency (and hence maximize total throughput) of the execution of a single thread at a time while GPUs try to maximize the total throughput by reducing the significance of added thread latency by parallelizing thread execution [11].

A modern NVIDIA GPU is made up of multiple simultaneous multi-processors (SM) each with multiple cores that can execute threads in parallel as seen in Figure 8. Threads executed simultaneously on the same SM can cooperate and share low-latency local memory, which helps reduce high-latency global memory accesses.

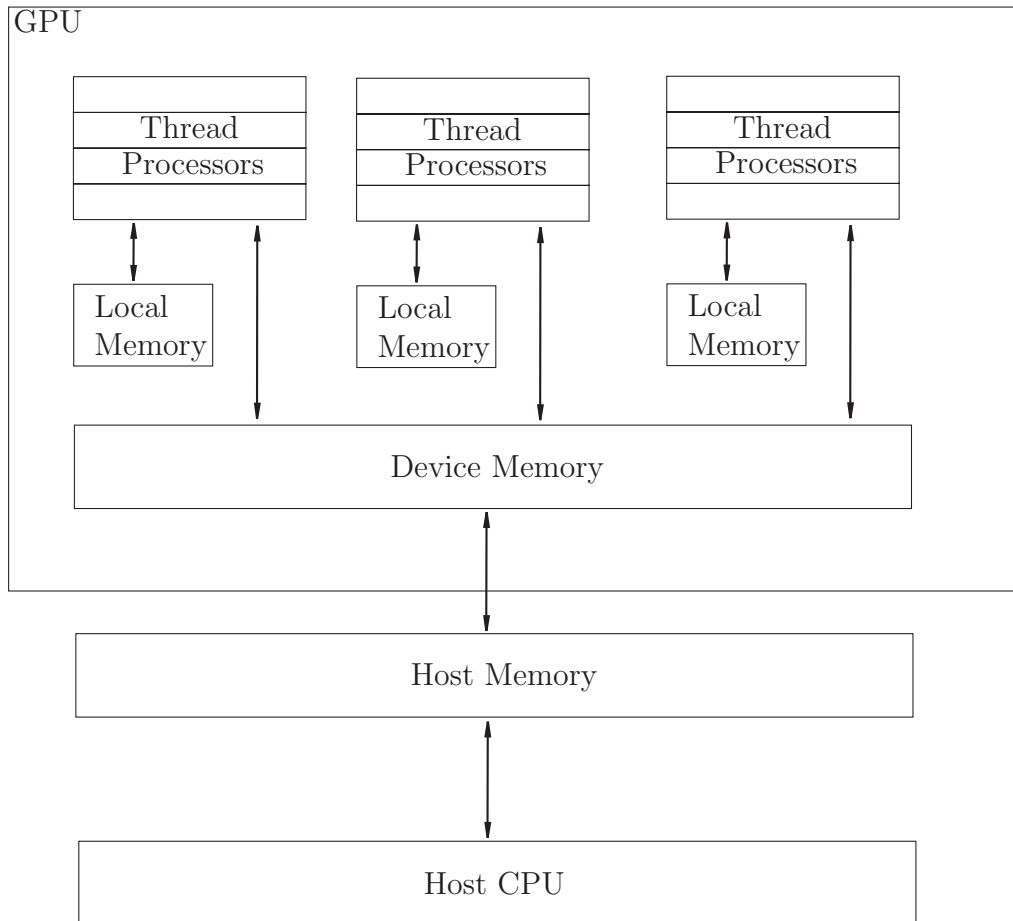


Figure 8: NVIDIA GPU with an array of multi-threaded simultaneous multi-processors

For the SMS API, geometries are discretised into a polyhedron mesh and the hydrodynamic forces acting on the surface of the object are determined for each face. This method of calculating fluid forces is described in more detail in section 2.1.2. The finer the polygon mesh, the more accurate the integration of the fluid forces on the object surface will be. The number of hydrodynamic calculations grows linearly

with the number of polygons. The calculations that must be effectuated for each face consists of the same instructions for all faces and can all be performed in parallel. This makes the problem of resolving the hydrodynamic fluid forces in this manner embarrassingly parallel.

To date, the feasibility of using CUDA to calculate these fluid interaction forces has been investigated for a simple hydrostatic buoyancy calculation, where the pressure applied to each polygon is:

$$p(\mathbf{C}_{\mathbf{p},i}) = \rho g h_i. \quad (30)$$

A thread is created for each polygon that calculates the simple hydrostatic buoyancy force applied on each polygon. All polygon forces are accumulated to determine the total force applied on the object. The threads are spread among the GPU's SMs in such a way that the block thread count does not exceed 512 threads per block.

A CUDA testbench application was created and the buoyancy calculation was implemented 4 times: a functional C implementation, an implementation making use of **GeomLib** like the SMS API, a CUDA conversion of the functional C implementation, and a CUDA conversion of the C implementation that makes use of shared memory. Results of these tests are found in Figure 9. The CUDA implementations were run on an EVGA GeForce GTX 465 OC graphics card which features 11 SMs each with 32 cores. It should be noted that the buoyancy calculation that makes use of **GeomLib** library was 6 times slower than a functional C implementation. CUDA implementations achieved a speedup even with a low resolution polygon mesh with speedups of around 2.5 for higher resolution meshes. It should be noted that the CUDA implementation is about 16 times faster than the **GeomLib** based implementation. The speed up is low for these tests because very few calculations are effectuated on each polygon, this means the ratio of time to acquire the polygon from global memory to the time to effectuate the calculation is large. The shared memory implementation showed no appreciable difference in speedup over the regular CUDA implementation because each polygon has its own data and can not make use of another polygon's data. Very few calculations are made for each polygon, and each thread does not make use of any data used by any other thread. It is anticipated that when moving fluids are considered, greater speedups will be achieved due to increased computation load per polygon and shared memory will play a much more important role in speedup since many polygons could share the same environmental data.

2.2 SMS::Director class

The **SMS::Director** class facilitates a user's interaction with SMS API **SMS::SimObjects**. The class allows a user to readily script commands that should be performed over the course of an SMS API simulation. The class consists of a queue of commands

Speedup of buoyancy calculation [baseline is a Barebone C Implementation]

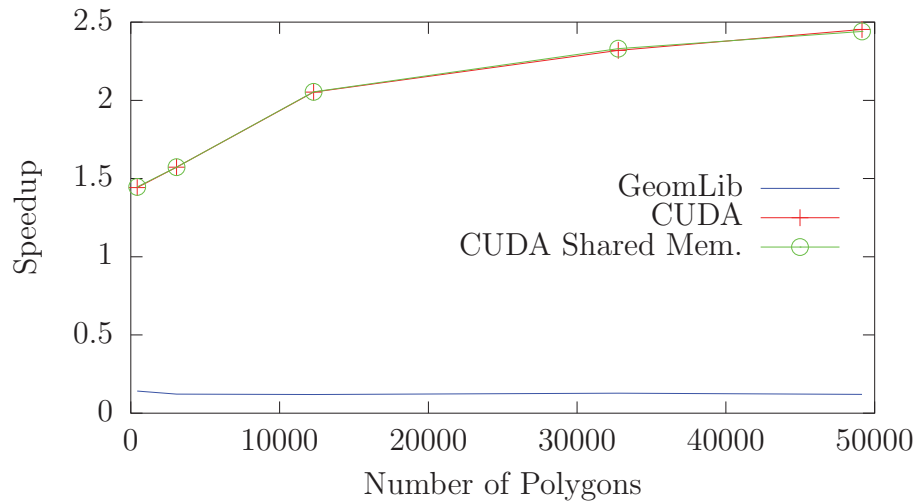


Figure 9: Performance results of a simple buoyancy calculation implemented in C, in C++ using *GeomLib*, in CUDA and in CUDA with shared memory. CUDA calculations were performed on an EVGA GeForce GTX 465 graphics card.

called a script, which the `SMS::Director` executes sequentially on a set of “actors” it is controlling, where `SMS::SimObjects` are the actors. Each `SMS::SimObject` type has it’s own set of commands that enable activities such as moving boomcrane joints to a particular position, paying in or out cable, or attaching/detaching cables.

Each command is placed in a First In First Out queue and are executed sequentially. The command consists of a string that begins with the name of the actor, followed by a sequence of other parameters that help to define the command. The user can add commands to the `SMS::Director` by using the `AddCommand(std::string commandStr)` method which adds `commandStr` to the end of its queue or by reading it from file as `myDirector.ReadScriptFromFile(std::string fileName)` where `fileName` is the name of the text file which contains the scripts.

2.2.1 Using the SMS::Director

Actors can be assigned the `SMS::Director` object using the following method:

```
SMS::Director::AddActor(SimObject *object,
                        std::string actorName)
```

where `object` is a pointer to an `SMS::SimObject` that will be stored and `actorName` is the name assigned to that particular actor. The `SMS::Director` will identify its actor by name only.

Commands are added to the `SMS::Director`’s script using the following command:

`SMS::Director::AddCommand(std::string command)`

where `command` is a string that contains multiple parameters that define the command.

2.2.2 SMS::Payload commands

The `SMS::Payload` has a number of commands available to it. The user can freeze certain degrees of freedom and release them throughout a simulation using a `hold` or `release` command, respectively. `SMS::Cables` can also be attached to or detached from an `SMS::Payload` using the `connect` and `disconnect` commands, respectively.

2.2.2.1 Placing and releasing holds placed on a degree of freedom

In order to lock an `SMS::Payload`'s degree of freedom in space over time, the following command would be provided to the `SMS::Director`:

```
"payloadName hold dof commandName"
```

where `payloadName` is the name given to the `SMS::Payload` when it was assigned as an actor, `hold` is the hold command, `commandName` is the name given to the command which will later identify it for release, and `dof` is one of the 6 degrees of freedom that is being locked and can be any of the following: X, Y, Z, EX,EY,EZ.

To release a lock on an `SMS::Payload`'s degree of freedom, the following command is provided to the `SMS::Director`:

```
"payloadName release commandName"
```

where `release` is the release command.

2.2.2.2 Connecting and disconnecting cables

The attachment and detachment of `SMS::Cables` to an `SMS::Payload` can be scripted using the following commands:

```
"payloadName cableName winchName connect node softConnectTime x y z"
and
```

```
"payloadName cableName winchName disconnect
softDisconnectTime"
```

where `payloadName` is the name of the `SMS::Payload` actor, `cableName` is the name of the `SMS::Cable` actor, `connect` and `disconnect` are the attachment/detachment commands, `x`, `y` and `z` are the cable attachment location with respect to the body-fixed frame of the `SMS::Payload`, `winchName` is the name of an `SMS::Winch` that might be attached to the `SMS::Cable`, `softConnectTime` and `softDisconnectTime` is the period of time over which to perform a connect or disconnect procedure to prevent tension shocks, and `node` is simply which cable node to connect and is either 0 or N. If there is no `SMS::Winch` on the `SMS::Cable`, `winchName` is set to `none`. If a `SMS::Winch` actor is specified, the `SMS::Director` will use the `SMS::Winch` actor to control the tension in the cable during the attachment and detachment process as

part of the soft connect/disconnect feature. Also, during the soft connect procedure, the end of the cable node is moved to the connection point over the elapsed period of time.

2.2.3 SMS::Boomcrane commands

The `SMS::Director` has one command available for an `SMS::Boomcrane` that allows the scripting of its joint actuations. The user can add sequential joint actuation commands as:

```
"boomcraneName jointID jointDesiredPos actuationVelocity"
```

where `boomcraneName` is the name of the `SMS::Boomcrane` actor, `jointID` is the joint ID number, `jointDesiredPos` is the desired joint position which it is actuated to (units of meters for prismatic joints and radians for revolute joints), and `actuationVelocity` is the unsigned magnitude of the velocity the joint will actuate by to reach the desired position.

2.2.4 SMS::Winch commands

The `SMS::Director` has one command available for an `SMS::Winch`. It allows the scripting of the pay in or pay out of cable. The user can add sequential pay in and pay out commands as:

```
"winchName cableName cableLength payoutVelocity"
```

where `winchName` is the actor name for the `SMS::Winch`, `cableName` is the actor name for the `SMS::Cable`, `cableLength` is the desired length of payed out cable, and `payoutVelocity` is the desired rate of change over time of the cable length used to achieve the desired length.

2.2.5 SMS::Cable commands

There are currently no `SMS::Director` commands available for an `SMS::Cable`. However, it must still be added as an actor if it is to be connected to an `SMS::Payload` or controlled by an `SMS::Winch`.

2.2.6 Elapsing time between commands

The timing of actions can be manipulated by adding pauses between commands. This is accomplished by adding the following command:

```
"pause timeToElapse"
```

where `pause` is the pause command and `timeToElapse` is the amount of time to allow to elapse before the next command is executed.

2.2.7 Example code

```
//Creating SimObjects
SMS::Payload testPayload("Payload.ini",0,true);
SMS::Cable testCable("Cable.ini",0,true);
SMS::Winch testWinch("Winch.ini",0,true);
SMS::BoomCrane testBoomCrane("BoomCrane.ini",0,true);

//Creating Director
SMS::Director myDirector;

//Assigning actors
myDirector.AddActor(&testBoomCrane,"testBoomCrane");
myDirector.AddActor(&testWinch,"testWinch");
myDirector.AddActor(&testCable,"testCable");
myDirector.AddActor(&testPayload,"testPayload");

//Hold the payload's Z position at 10m
myDirector.AddCommand("testPayload hold Z 10 holdZ");

//move the boomcrane's second joint to pi/4 trying to maintain
//0.5 radians per second
myDirector.AddCommand("testBoomCrane 1 0.7854 0.5");

//Pay in/out cable until it is 12.4 meters long at a rate of
// 1.5 m/s
myDirector.AddCommand("testWinch testCable 12.4 1.5");

//Release the payload's Z hold
myDirector.AddCommand("testPayload release holdZ");

//Connect the cable to the payload, there's a attached to the
//cable and we want to use it to soften the connection over a
//period of 1 seconds
myDirector.AddCommand("testPayload testCable testWinch
connect 0 1.0 0.0 0.0 1.5 ");

//Pause
myDirector.AddCommand("Pause 10");

//Disconnect the cable using an attached winch to
//soft-disconnect.
myDirector.AddCommand("testPayload testCable testWinch disconnect 0.5");
```

2.3 Contact dynamics

A few major algorithm modifications have been made to the SMS API to handle concave objects (via convex decomposition) and to generally improve the performance of the code. First, a new collision detection infrastructure was put in place to handle the difficulties that arise with convex decomposed collision geometries, such as keeping track of collision pair contact data which must be remembered for future time steps. The new architecture is described in Section 2.3.1. Second, a new penetration depth or minimum translational distance (MTD) determination method called the Expanded Polytope Algorithm (EPA), which is discussed in Section 2.3.2, was implemented to improve performance over the previous brute force method. Finally, GeomLib's GJK implementation has been modified to make use of temporal coherence. A discussion on how temporal coherence is exploited can be found in Section 2.3.3.

2.3.1 Improved collision detection code

In order to handle the complexities that arise when dealing with decomposed collision objects, which are assembly of several connected convex hull objects, it was necessary to introduce a revised collision detection infrastructure in the SMS API. Difficulties in debugging and maintaining the initial infrastructure necessitated the development of a revised infrastructure. At the foundation of the revised infrastructure is an abstract base class called `GeomLib::CollisionObject`. This class forms the foundation for the creation of other types of collision objects (e.g. convex collision objects, Axis Aligned Bounding Boxes (AABB), Oriented Bounding Boxes (OBB), etc.).

The improvements can be summarized as:

- The creation of two child classes of the `CollisionObject` class to handle convex collision object and convex decomposed collision object collisions. The classes are called `GeomLib::CollObjConvex` and `GeomLib::CollObjConvexDecomp`, respectively.
- Every `CollisionObject` type can be tested against any other type of `CollisionObject` for collision by simply passing a pointer to the other `CollisionObject` as a parameter of the `CollisionObject::DetectCollision()` function.
- The new collision infrastructure is compatible for N -body collision resolution.
- Each `CollisionObject` maintains a database of collision information stored in a `std::map` with all other `CollisionObjects` it is tested against, making temporal coherence and time dependent contact phenomena possible.

Details of the improvements are provided in the following sections.

2.3.1.1 Details of improved collision detection code

The key function in the `GeomLib::CollisionObject` class is the pure virtual function `DetectCollision(...)` which is passed a pointer to another `CollisionObject`. Every child class implements its own version of that function and will cast the `CollisionObject` pointer passed in to it to whatever type of `CollisionObject` child class it is; and deal with detecting a collision appropriately. Every type of `CollisionObject` should be able to be passed in any type of `CollisionObject` and know how best to perform the collision detection check between itself and that particular `CollisionObject` type.

Two `CollisionObject` types have been implemented: `GeomLib::CollObjConvex` and `GeomLib::CollObjConvexDecomp`. The `CollObjConvex` object includes an internal `GeomLib::Polyhedron` object that defines its geometry which is passed in during its initialisation process. The `CollObjConvex` guarantees that the `Polyhedron` is convex by processing it through the `GeomTools::QuickHull3D(...)` algorithm. If a `CollObjConvex` is passed in another `CollObjConvex` via the `DetectCollision()` function, the GJK algorithm (`GeomTools::FindMSDGJK(...)`) is used to perform the collision query.

The `CollObjConvexDecomp` object contains a `std::vector` of `CollObjConvex` objects. Because of the use of a `std::vector` of `CollObjConvex`s instead of some sort of Bounding Volume Hierarchy (BVH), every `CollObjConvex` must be tested for collision against the `CollisionObject` passed in. This can lead to a large number of unnecessary collision checks especially if it is tested against another `CollObjConvexDecomp` and both have a large numbers of sub objects. Each `CollObjConvex` will be tested against every `CollObjConvex` of the other `CollObjConvexDecomp` leading to $N \times M$ collision checks, where N and M are the number of `CollObjConvex` sub objects for each `CollObjConvexDecomp` object, respectively. In the future, `CollObjConvexDecomp` collision tests could be sped up using an AABB/OBB tree or some other BVH.

Information about individual collisions, such as the Minimum Translation Distance (MTD), the volume of interference and the final GJK simplex for temporal coherence, are stored inside a `GeomLib::CollisionDataContainer` object. Every `CollisionObject` has a `std::map` of `CollisionDataContainer` objects. Every time a new `CollisionObject` is queried for collision with another `CollisionObject`, a new `CollisionDataContainer` is added to the `std::map` and is mapped using a collision object *id* which is unique to each `CollisionObject`. A copy of the `CollisionDataContainer` is held by each `CollisionObject` in the collision pair. Whenever a `CollisionObject` is queried a second time for collisions with another, the Coll-

isionDataContainer for that collision pair is easily obtainable regardless of whether CollisionObject *A* queried CollisionObject *B* or vice-versa. This leads to some duplicated data though it renders the system more flexible.

This new collision detection infrastructure forms a strong basis around which a future expansion into broad-phase collision detection to handle more general multi-body simulations can occur. This would enable the boomcrane arm segments to collide with each other and the payload.

2.3.2 Expanded Polytope Algorithm

The SMS API determines the contact force acting between two colliding objects using the volume of interference between both objects. Currently, this is done using the Muller-Preparata algorithm [12]. This requires knowledge of a point that is simultaneously located inside both colliding objects or in other words inside the volume of interference. To accomplish this, the SMS API uses the MTD, which is represented as a line segment between two points, \mathbf{P}_1 and \mathbf{P}_2 , one on each object that represent the maximum penetration depth, or the Minimum Translation Distance to achieve separation. With knowledge of \mathbf{P}_1 and \mathbf{P}_2 , a point located within the volume of interference can usually be obtained¹ as $\frac{\mathbf{P}_1 + \mathbf{P}_2}{2}$, as shown in Figure 10.

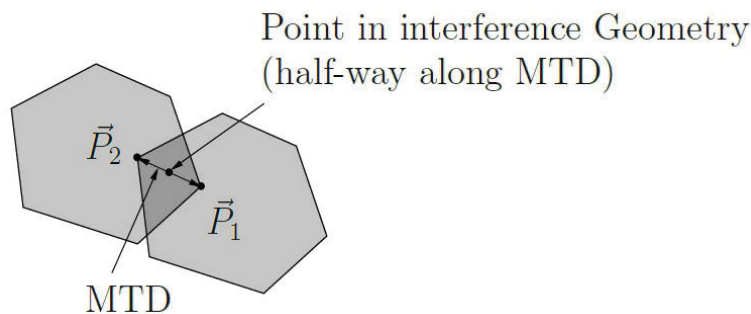


Figure 10: The use of the MTD to obtain a point within the interference volume of two colliding objects.

The original implementation of the MTD determination code in SMS API used a brute force approach with a complexity of $O(N * M)$ where N and M are the number of vertices of each collision object. Essentially, the SMS API constructed the entire Minkowski difference and searched its surface for the closest point to the origin [1]. This resulted in an inefficient algorithm that did not scale well to geometries with a large number of vertices. To improve the MTD code, a new method called the

¹DSA is aware of an exception where the use of the MTD would fail to provide a point within the volume of interference.

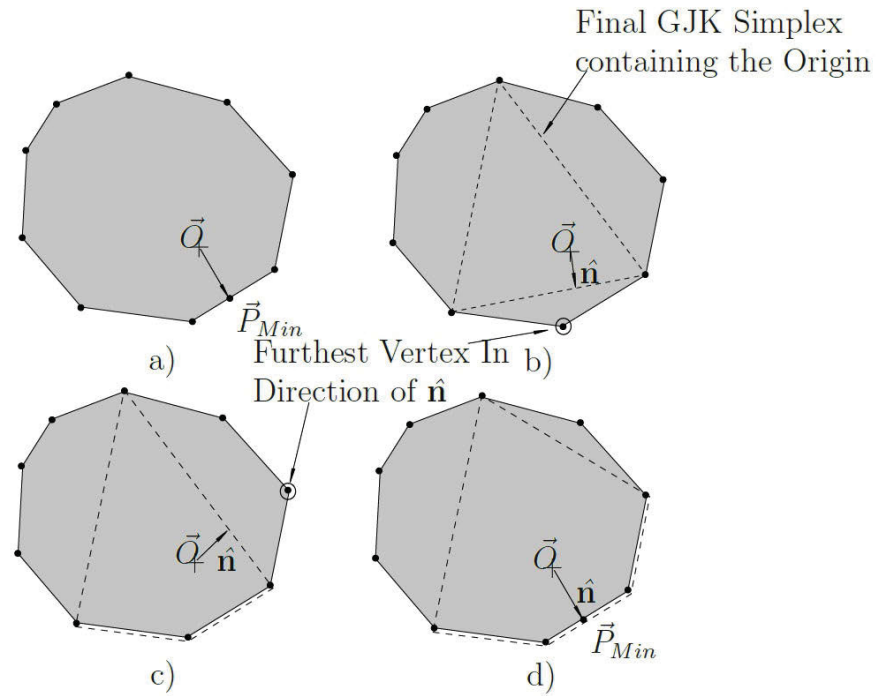


Figure 11: A step by step description of the Expanded Polytope Algorithm for interfering convex polyhedra: a) The desired point on the surface of the Minkowski Difference surface that is closest to the origin \mathbf{P}_{min} , b) the original simplex returned from GJK containing the origin showing the search direction vector \hat{n} from the normal of the closest face along with the furthest vertex in that direction, c) The new geometry with the furthest point from the previous step including the new search direction and furthest point, d) finally no further point can be identified in the search direction, thus the closest face contains \mathbf{P}_{min} .

Expanded Polytope Algorithm (EPA) [13] has been implemented. The EPA is a complementary algorithm to GJK to help obtain the MTD and it has a complexity of $O(\log(N + M))$ so long as the polyhedral geometries make use of the Dobkin-Kirkpatrick hierarchy [14]. The MTD can be obtained from the point \mathbf{P}_{Min} on the surface of the Minkowski difference surface which is closest to the origin [1], as shown in Figure 11 a). Instead of generating the entire Minkowski difference, the EPA starts with a polyhedron created from the final 4-simplex (a tetrahedron) returned from GJK which contains the origin, and identifies the face closest to the origin. From here, a vertex of the Minkowski difference which lies furthest outside that face using the support function is found, as shown in Figure 11 b) and c). If a vertex is found to lie outside the face, it is added to the polyhedron which expands the polytope. `GeomTools::QuickHull3D(...)` is used to effectively and efficiently add it the mesh. When the support function fails to return any vertex which lie outside the closest face, the point \mathbf{P}_{Min} is determined which provides the MTD [1].

2.3.3 Exploiting temporal coherence with GJK

The GJK algorithm [1, 15] determines whether or not two objects are interfering by stepping through the Minkowski difference until a 4-simplex is found which encapsulates the origin. This process has a complexity of $O(\log(N + M))$ where N and M are the number of vertices for each interfering convex polyhedron when Dobkin-Kirkpatrick hierarchies are used. However, unless the objects are moving at large rates, or the numerical integration is taking large time steps, the Minkowski difference will vary little between time steps. The final 4-simplex returned from GJK is likely to be identical or near identical to that returned in the previous time step. This can be exploited to speed up the execution of the GJK algorithm. That is, the final simplex obtained by GJK, whether it has determined that a collision is or is not occurring, is saved for use in the next time step. This simplex is used as the starting point for the next GJK run. In general, GJK will not require any stepping through the Minkowski difference and is likely to immediately return a collision or non-collision state. Thus, by exploiting temporal coherence, GJK can be effectively made to run at $O(1)$ regardless of the complexity of the geometries except for the first run.

2.3.4 Contact dynamics code performance

This section presents the benefits of the contact dynamics code improvements. Section 2.3.4.1 summarizes the results of implementing the EPA, using temporal coherence with the GJK, and improving memory management in the SMS API. Section 2.3.4.2 reviews additional benefits and improvements that were made to the SMS API contact dynamics capabilities by examining the algorithmic complexity of GJK, EPA and the Muller-Preparata algorithms.

2.3.4.1 Code performance increases

Benefits of memory allocation optimisation When the EPA and the temporal coherence algorithm changes were implemented in the SMS API, the ‘boat and cradle’ simulation from Section 3.2.4 had a simulation time ratio of 28:1². To reduce this time ratio function level memory creation and destruction was minimized by:

- removing functions with vectorial return types,
- removing function or loop level memory creation.

In addition, the code was generally cleaned up by removing unnecessary tasks from functions. After these changes were made, the ‘boat and cradle’ simulation had a simulation time ratio of 23:1.

Benefits of algorithmic changes To demonstrate the benefits of the algorithmic improvements made, the boat and cradle simulation was re-run without temporal coherence and again when using the brute force MTD method instead of EPA. These simulations’ time ratios, including those discussed above can be found in Table 3. From this table, one can see that the largest performance increase was due to replacement of the brute force MTD method with EPA. Temporal coherence and memory optimisation also contributed significantly to the overall performance increase.

Table 3: *The time ratios for the boat and cradle simulation before and after optimisation, without temporal coherence and with the original brute force MTD method.*

Variation	Time ratio
Algorithmic changes, no memory optimisation	28:1
Algorithmic changes, with memory optimisation	23:1
Without temporal coherence, with memory optimisation	32:1
Brute force MTD method, with memory optimisation	166:1

2.3.4.2 Algorithmic complexity of GJK, EPA and Muller-Preparata

To demonstrate the scalability and in some cases the improvements of the computational geometry algorithms used to detect collisions and determine the contact forces in the SMS API, the run times of the GJK, EPA and the Muller and Preparata algorithms for pairs of spheres with varying mesh resolutions were determined. The execution times of each of these functions were recorded and averaged over 300 trials for each mesh resolution pair and plotted in the proceeding figures.

²This large time ratio is mainly due to the fact that the boat is subdivided into 19 sub-objects and the cradle is made up of 4 sub-objects; leading to ~80 potential collision pairs.

The GJK algorithm has a complexity of about $O(\log(n + m))$ where n and m are the number of polygons between both objects. This complexity and scalability is already quite desirable, however by exploiting temporal coherence a complexity of $O(1)$ can be, and has been, achieved. The execution times of the GJK algorithm with and without exploiting temporal coherence were obtained for a pair of identical non-interfering spheres of varying mesh resolutions. The results of those runs can be found in Figure 12. The execution times of the GJK algorithm without temporal coherence follows a logarithmic curve while the GJK algorithm with temporal coherence has a constant time complexity.

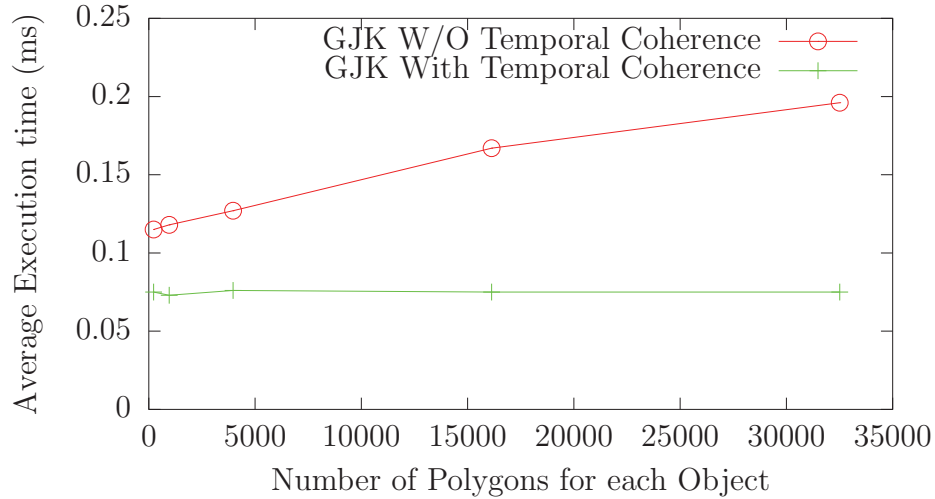


Figure 12: The execution times for the GJK algorithm for pairs of identical spheres of varying mesh resolutions.

The EPA algorithm also has a $O(\log(n + m))$ complexity. The previous algorithm, a brute force approach, used to determine the minimum translational distance (MTD) or penetration depth, constructed the entire Minkowski difference geometry. This costly operation had a time complexity of $O((n + m)^2)$. Though it would have been interesting to compare the execution times of these two methods, the brute force approach took far too long to execute with such high polygon counts thus the comparison was not practical or necessary. The execution times for the EPA algorithm can be found in Figure 13.

Due to the way in which the EPA converges to its solution, it was anticipated that the execution time of the EPA algorithm might be shorter when the MTD was small. To demonstrate this phenomenon, two spheres of 16128 polygons each were processed through GJK and EPA from a state of non-interference to a penetration of $\frac{1}{2}$ radius. The results of these tests can be found in Figure 14. As anticipated, the EPA algorithm performs somewhat faster in cases of small interpenetration. This limitation is fine for most contact situations since large penetration depths are not typical.

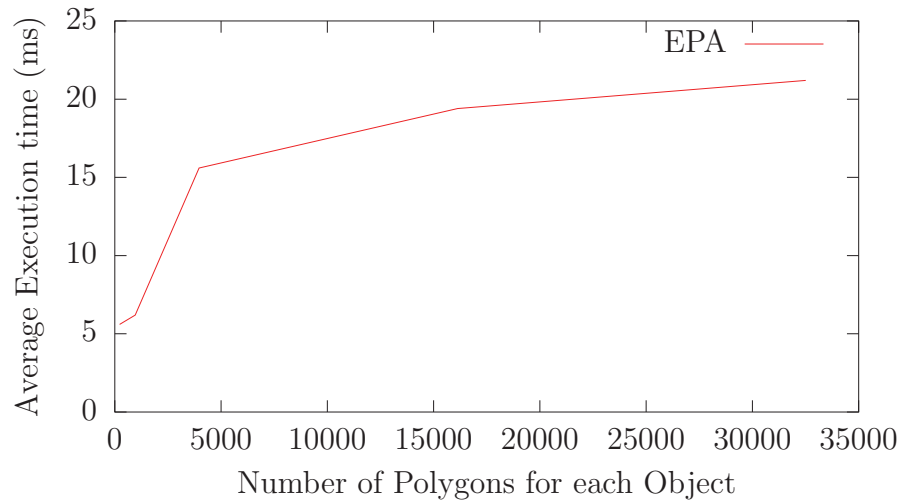


Figure 13: The execution times for the EPA algorithm for pairs of identical spheres of varying mesh resolutions.

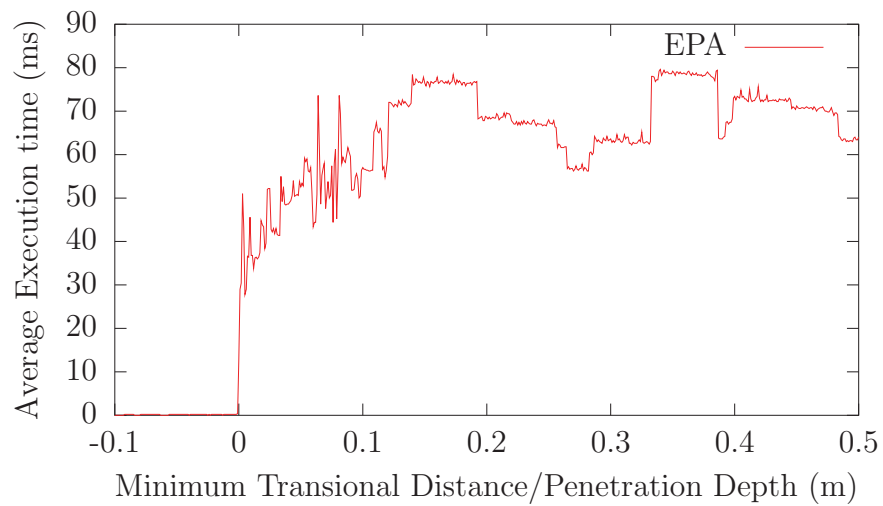


Figure 14: The execution times for the EPA algorithm for a pair of identical spheres of 16128 polygons each for varying penetration depths.

Finally, the Muller-Preparata intersection geometry algorithm has a worst-case complexity of $O((n + m)\log(n + m))$. This complexity is because the Muller-Preparata algorithm is dependent on the use of a convex hull algorithm. The most efficient convex hull algorithms (such as the QuickHull3D algorithm), have a complexity of $O((n + m)\log(n + m))$. Thus, the bottleneck in the Muller-Preparata algorithm is the convex hull algorithm. If a more efficient convex hull algorithm can be found, it would have a significant impact on the Muller-Preparata algorithm. Unfortunately, the authors are currently unaware of a more efficient volume of interference algorithm or 3d convex hull algorithm. There is also a lack of efficient parallel convex hull algorithms, so speeding up the Muller-Preparata algorithm through parallelization is not immediately evident. The execution times of the Muller-Preparata algorithm for two identical spheres of varying mesh resolutions for the Muller Preparata algorithm can be found in Figure 15.

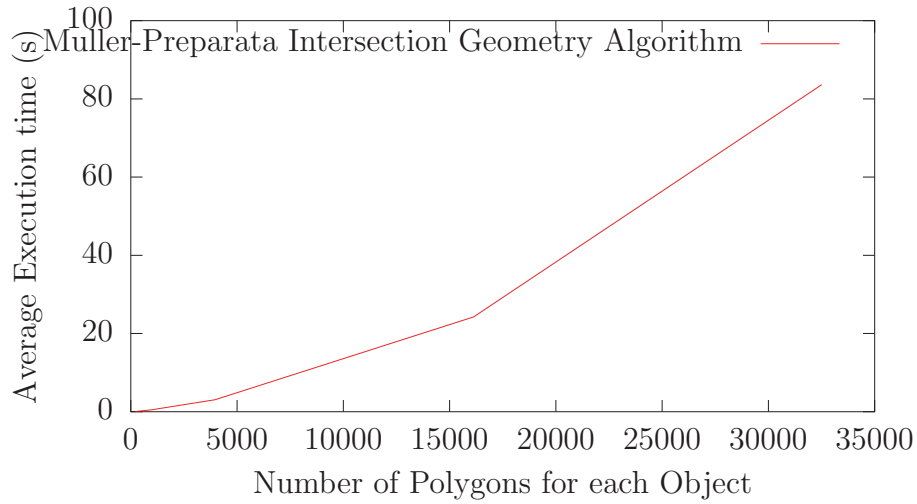


Figure 15: The execution times for the Muller-Preparata algorithm for pairs of identical spheres of varying mesh resolutions.

2.4 Generalised- α implicit integrator

The objective of this section is to describe the Generalised- α numerical integrator. Section 2.4.1 discusses the difference between explicit and implicit numerical integrators in general. Section 2.4.2 discusses the Generalised- α integrator in detail, while Section 2.4.3 discusses issues that arise when using Generalised- α with multiple degree of freedom (DOF) nonlinear systems.

2.4.1 Implicit integration

An explicit numerical integrator evaluates accelerations at a given point in time as a sole function of present time position and velocity or in other words system state information. This allows a numerical integration process to march forward without any iteration as the accelerations are evaluated directly and utilized to predict future positions through some particular explicit algorithm, such as Runge-Kutta regime. In contrast, implicit numerical integrators predict future positions by using an assumed acceleration profile over a span of time on the order of the time step. This results in quantifying the future state via a blend of accelerations known at the present time as well as from some future point in time and requires iteration to converge on the future state as the future accelerations can not normally be known *a priori*. A particular nuance of implicit integrators is their ability to introduce artificial numerical damping. This can help attenuate high frequency effects that otherwise may destabilize a system or require unrealistic or complex physical damping to control. This can afford significant increases in numerical stability in some systems that otherwise require a much smaller time step or are even incapable of execution with explicit numerical integrators. It is very common for large degree of freedom structural or computational fluid dynamics models to use implicit numerical integration when studying dynamic problems. While stability is an advantage in its own right, artificial numerical dissipation may afford larger time steps that also significantly increase simulation execution speed when compared to explicit numerical integration.

2.4.2 The Generalised- α integrator

There are several implicit numerical integrators available that have been developed over several decades. One of the more recent versions referred to as the Generalised- α method is an excellent candidate for implementation [16]. This particular integrator has a single, user-settable parameter that directly controls the amount of artificial numerical dissipation in the integration process. In the original work presenting this integrator, the authors analysed its effect on a linear second-order differential equation:

$$\ddot{u} + 2\zeta\omega\dot{u} + \omega^2u = f \quad (31)$$

which was then reformulated in to a linear algebraic update equation giving the relationship between present and future states via the amplification matrix, \mathbf{A} :

$$\mathbf{X}_{n+1} = \mathbf{A}\mathbf{X}_n \quad (32)$$

The spectral radius, ρ , is defined as the maximum absolute value of the eigenvalues of the amplification matrix. The authors found that the spectral radius varies as a function of the ratio of the numerical time step used divided by the natural period of the system. For values less than 1 (indicating a very small time step), the spectral radius produced approached 1. As this ratio passed a value of 1 and increased, the spectral radius converges to a value between 1 and 0 and this limit is referred to the spectral radius at infinity, ρ_∞ . Through additional analysis and constraints, the authors formulated the coefficients of the integrator as a function of the spectral radius at infinity, giving a single parameter to control the amount of artificial numerical dissipation in the system. When the spectral radius, and therefore the largest eigenvalues of the amplification matrix, is less than 1, the system dynamic response is attenuated, emulating the effect of damping. In general, an analyst can establish the effect of numerical dissipation readily by sensitivity study and comparison with the RK45 integrator. However, based on the experience gained in this work on cable dynamics, it is expected that maximum damping will be typically used. Other than the damping effect, there are no other numerical or accuracy costs to using different values of ρ_∞ .

It should be noted that the Generalised- α parameters can also be easily adjusted to known values to produce identical results of older implicit numerical integrators such as Wilson- θ and Newmark- β . In more recent years, the literature indicates cable dynamics problems can be resolved utilizing these integrators, which indicates it is a feasible approach [17, 18]. One challenge with implicit numerical integrators is a theoretical framework for error control and therefore adaptive time step capability. However, more recent work in the literature indicates that some algorithms exist and are worth exploring in future work to better leverage the capabilities of implicit numerical integration [19].

All implicit numerical integrators afford blending of present and future accelerations in order to predict future positions by utilizing position and velocity update equations as their theoretical foundation. For mechanical systems in particular they also enforce dynamic equilibrium at some future time via Newton's Second Law of motion. The Generalised- α update equations for displacement, d , velocity, v are [16]:

$$\mathbf{d}_{n+1} = \mathbf{d}_n + h\mathbf{v}_n + h^2 ([0.5 - \beta] \mathbf{a}_n + \beta \mathbf{a}_{n+1}) \quad (33)$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h ([1 - \gamma] \mathbf{a}_n + \gamma \mathbf{a}_{n+1}) \quad (34)$$

and equilibrium at a future time point is guaranteed by utilizing Newton's Second Law:

$$\mathbf{M}\mathbf{a}_{n+1-\alpha_m} + \mathbf{C}\mathbf{v}_{n+1-\alpha_f} + \mathbf{K}\mathbf{d}_{n+1-\alpha_f} = \mathbf{F}(t_{n+1-\alpha_f}) \quad (35)$$

where

$$\mathbf{d}_{n+1-\alpha_f} = (1 - \alpha_f)\mathbf{d}_{n+1} + \alpha_f\mathbf{d}_n \quad (36)$$

$$\mathbf{v}_{n+1-\alpha_f} = (1 - \alpha_f)\mathbf{v}_{n+1} + \alpha_f\mathbf{v}_n \quad (37)$$

$$\mathbf{a}_{n+1-\alpha_m} = (1 - \alpha_m)\mathbf{a}_{n+1} + \alpha_m\mathbf{a}_n \quad (38)$$

$$t_{n+1-\alpha_f} = (1 - \alpha_f)t_{n+1} + \alpha_ft_n \quad (39)$$

and h is the time step at time point n , \mathbf{a} is the acceleration and \mathbf{M} , \mathbf{C} and \mathbf{K} are the system mass, damping and stiffness values respectively. In the original work, analysis was completed to determine the appropriate relationships between the constants γ , β , α_f , and α_m . Relationships were derived that enforce second order accuracy and desired level of numerical dissipation [16].

2.4.3 Implementation for nonlinear systems

For linear systems, the structural stiffness, damping, and mass values indicated in equation 35 are by definition independent of the system state. The stiffness and damping terms are a manifestation of the structural response to strain and strain rate of the system and in any case are constant based on the mass spring damper system properties.

However, the structural response of a cable is highly nonlinear with stiffness and damping terms changing depending on the state of the system. A significant source of nonlinearity is that the stiffness term is a function of the tension in the cable, though there are nonlinearities in mass and damping matrices as well. In order to accommodate the nonlinear nature of the system, equation 35 is resolved utilizing a Newton-Raphson approach as indicated in [17]. In this approach, the external forces (including hydrodynamic loading) are assumed to be constant over the time step. The Newton-Raphson approach is implemented by utilizing the update equations to solve for the future acceleration, \mathbf{a}_{n+1} , and future velocity, \mathbf{v}_{n+1} to be solely in terms of a single unknown, the future displacement, \mathbf{d}_{n+1} . Rearranging equations 34 and 34 produces:

$$\mathbf{a}_{n+1} = \frac{1}{\beta h^2} [\mathbf{d}_{n+1} - \mathbf{d}_n] - \frac{1}{\beta h} \mathbf{v}_n - \frac{[0.5 - \beta]}{\beta} \mathbf{a}_n \quad (40)$$

$$\mathbf{v}_{n+1} = \frac{\gamma}{\beta h} [\mathbf{d}_{n+1} - \mathbf{d}_n] + \left[1 - \frac{\gamma}{\beta}\right] \mathbf{v}_n + \left[h - h\gamma - \frac{h\gamma}{\beta} (0.5 - \beta)\right] \mathbf{a}_n \quad (41)$$

Note that equations 40 and 41 show future accelerations, \mathbf{a}_{n+1} , and velocities, \mathbf{v}_{n+1} , expressed in terms of the single unknown parameter of future displacement, \mathbf{d}_{n+1} . Present time displacement and velocity, are known from either initial conditions or from the last numerical integration step. The initial accelerations are known directly from Newton's Second Law or are the product of the last numerical integration step.

The result of substituting equations 36-39 in to equation 35 produces equation 42 [17]:

$$(1 - \alpha_m)\mathbf{M}\mathbf{a}_{n+1} + \alpha_m\mathbf{M}\mathbf{a}_n + (1 - \alpha_f)\mathbf{K}\mathbf{d}_{n+1} + \alpha_f\mathbf{K}\mathbf{d}_n = (1 - \alpha_f)\mathbf{F}_{n+1} + \alpha_f\mathbf{F}_n \quad (42)$$

where the stiffness matrix \mathbf{K} represents the axial, bending, and torsion stiffness of the cable. Note that in this case structural damping is lumped in with the external forces. In addition, unlike the work presented in [17] additional temporal blending of the mass, \mathbf{M} , and stiffness matrices has been neglected for simplicity. Equation 42 represents Newton's Second Law as envisioned through the Generalised- α algorithm, which blends present and future accelerations to establish the future state. If equations 40 and 41 are substituted in to equation 42, the resulting equation is still Newton's Second Law but with a single unknown in the form of the future displacement. Recall that the present displacement, velocity, and accelerations are known quantities. The tangent stiffness matrix is the term used to describe the remaining matrix of terms that are proportional to the future displacement and include components from the structural stiffness as well as mass terms. The tangent stiffness matrix therefore describes inertial resistance to abrupt temporal changes in future position as well as resistive structural forces that oppose changes in future position regardless of the time step.

The iterative Newton-Raphson technique can be used to resolve the future displacement based on a first-order Taylor series approximation to any function as described in [18]:

$$\mathbf{f}(x + \epsilon) = \mathbf{f}(x) + \mathbf{J}(x)\epsilon = 0 \quad (43)$$

The change required in the state, ϵ , can be found with knowledge of the matrix of first derivatives or Jacobian, \mathbf{J} , by solving the system 43:

$$\epsilon = -[\mathbf{J}(x)]^{-1}\mathbf{f}(x) \quad (44)$$

Typically the time steps must be small for stability purposes and this affords an approximation to the Jacobian:

$$\mathbf{J}(x) \approx (1 - \alpha_f)\mathbf{K} + \frac{1 - \alpha_m}{\beta h^2}\mathbf{M} \quad (45)$$

In the context of structural analysis problems that utilize the Newton-Raphson approach, this approximation is referred to as the tangent stiffness matrix. The structural stiffness matrix, \mathbf{K} , and combined lumped structural mass and directional added mass matrix, \mathbf{M} , are computed at each point in time. The tangent stiffness matrix is then computed and an LU decomposition regime is leveraged to solve the system of equations from equation 44. As the full mass and stiffness matrices are required, the number of equations in the cable model grows rapidly with the number of nodes. However, the LU decomposition step is the most computationally costly and is only performed once per time step and multiple iterations in the Newton-Raphson process do not significantly increase the computational burden.

3 Validation

3.1 Hydrodynamics validation

3.1.1 Partially submerged body buoyancy test in calm water

To verify accuracy of the buoyancy force created by the fluid pressure field for a partially submerged body, a buoyant box shaped SMS::Payload is placed in a calm seaway as shown in Figure 16. The SMS::Payload has dimensions of 10 m in length and width and a height of 1 m which makes it a stable low-profile floating platform. The SMS::Payload has a mass of 25625 kg and a density of 256.25 kg/m³, which is $\frac{1}{4}$ that of ocean water at 1025 kg/m³. It has the mass moment of inertia of a solid box where $I_{xx} = I_{yy} = 210,000 \text{ kg} \cdot \text{m}^2$ and $I_{zz} = 430,000 \text{ kg} \cdot \text{m}^2$. Each of the 6 sides of the box is discretised into 10×10 sets of 4 sided polygons, for a total of 600 polygons.

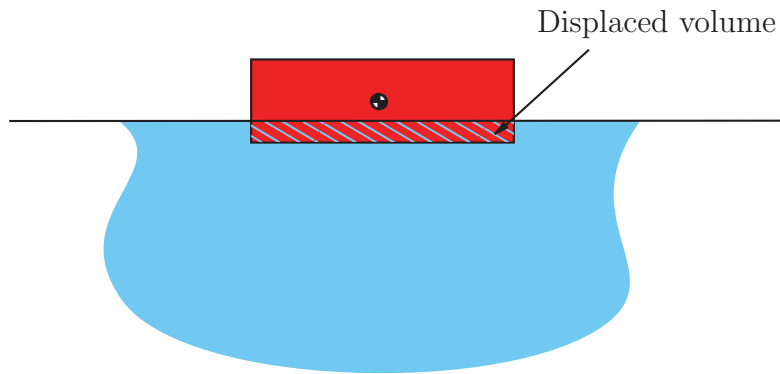


Figure 16: The experimental setup for the variable buoyancy validation test.

The SMS::Payload is expected to displace $\frac{1}{4}$ of the volume of the box of water, the $\hat{\mathbf{Z}}$ position of the SMS::Payload should average 0.25 m above the ocean surface ($Z = 0 \text{ m}$).

Results

Table 4 shows both the expected and the simulated average position for the floating SMS::Payload. Figure 17 indicates its position over time, and Figure 18 indicates orientation over time.

Table 4: The expected and simulated $\hat{\mathbf{Z}}$ position of the SMS::Payload

Variable	Expected $\hat{\mathbf{Z}}$ position	Simulated $\hat{\mathbf{Z}}$ position
$\mathbf{Z}_{\text{Payload}}$	0.25 m	0.250133 m

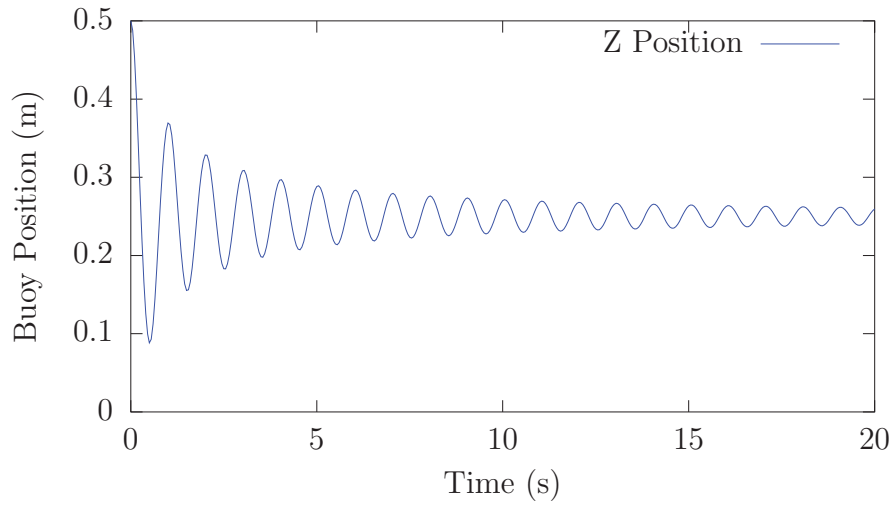


Figure 17: The \hat{Z} position of an *SMS::Payload* floating on water for the variable buoyancy validation test.

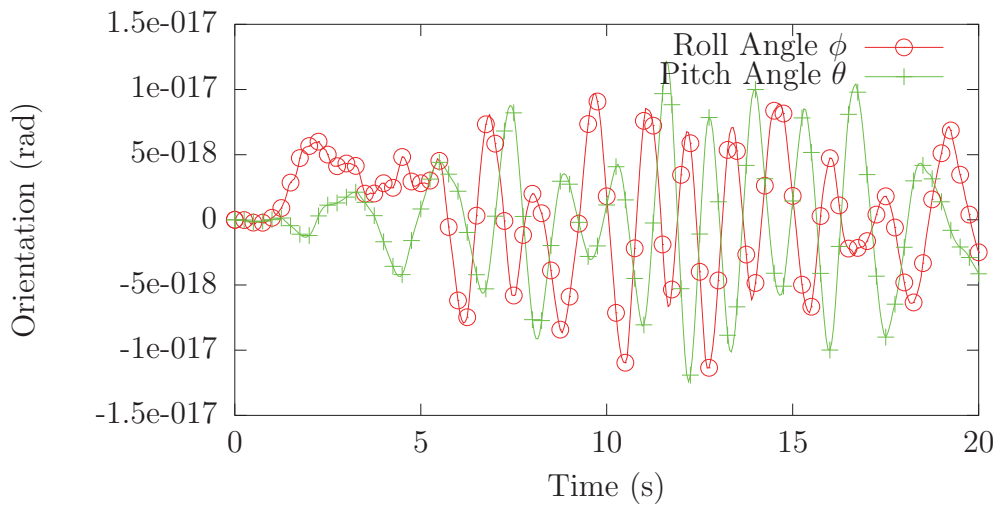


Figure 18: The orientation of an *SMS::Payload* floating on water for the variable buoyancy validation test.

3.1.2 Fully submerged body pendulum test in calm water

To verify accuracy of the buoyancy force created by the fluid pressure field for a fully submerged body, a buoyant sphere shaped `SMS::Payload` is placed in a calm seaway at $\mathbf{P}_{E \rightarrow B} = \{0.0, 0.01, -10.0\}$ attached to a 10 m long `SMS::Cable` whose bottom node is fixed in space, as shown in Figure 19. The `SMS::Payload` has a radius of 1 m, a volume of 4.1888 m^3 , a mass of 1000 kg, and the mass moment of inertia of a solid sphere where $I_{xx} = I_{yy} = I_{zz} = 400 \text{ kg} \cdot \text{m}^2$. The fluid drag and inertial loading effects on the `SMS::Payload` are turned off. The spherical `SMS::Payload` surface is represented using a spherical polyhedron with 128 triangular polygons.

The `SMS::Cable`'s node 0 is attached to the `SMS::Payload` at $\{0.0, 0.0, 0.0\}$ from its body-fixed frame and has its node N located at the global $\{0.0, 0.0, -20.0\}$.

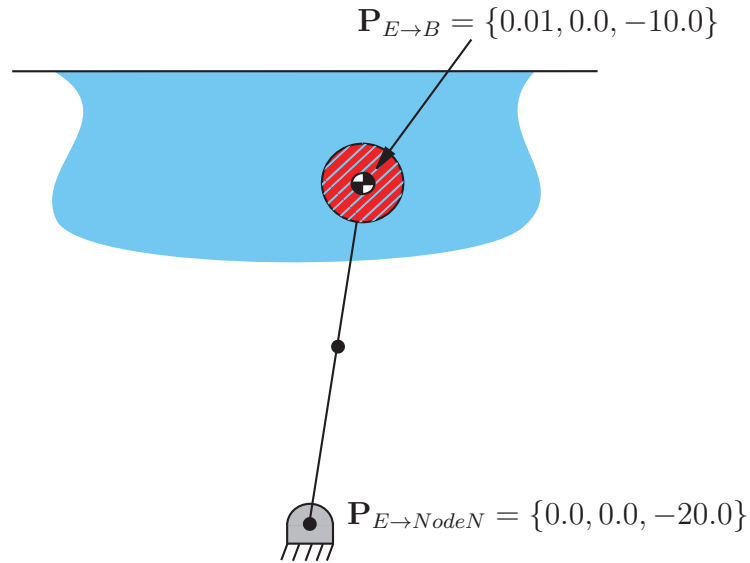


Figure 19: The experimental setup for the fully submerged body buoyancy pendulum validation test.

The expected period of oscillation for the pendulum is:

$$\tau_{pend} = \frac{2\pi}{\sqrt{\frac{F_b - mg}{mL}}} \quad (46)$$

where F_b is the buoyant force:

$$F_b = \rho_w g V_{Payload} \quad (47)$$

m is the mass, g is the gravitational acceleration, L is the length of the cable, ρ_w is the density of sea water and $V_{Payload}$ is the volume of water the payload displaces.

Results

Table 5 indicates the expected and simulated periods of oscillation for the `SMS::Payload` and Figure 20 illustrates its position over time.

Table 5: The expected and simulated period of oscillation of the `SMS::Payload`.

Variable	Expected period of oscillation (s)	Simulated period of oscillation (s)
τ_{pend}	3.6516	3.6409

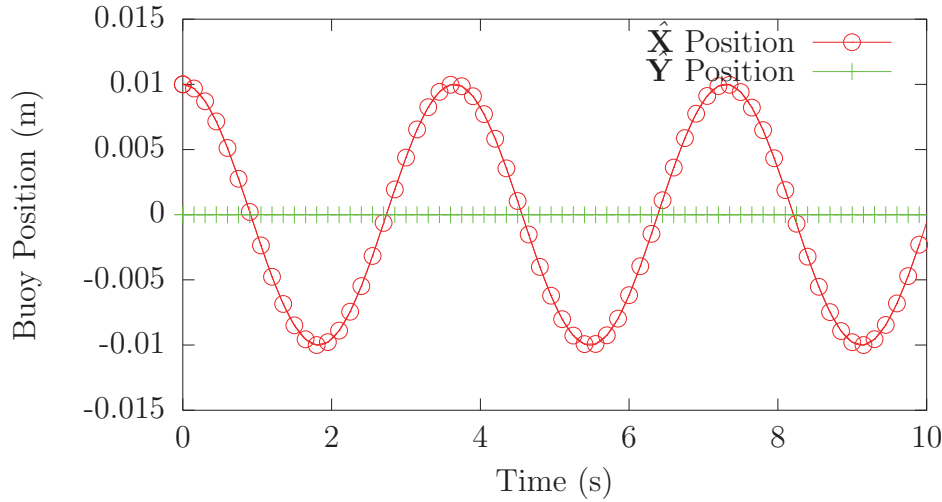


Figure 20: The \hat{X} and \hat{Y} position of a `SMS::Payload` attached to a 10 m cable pendulating by a buoyancy force larger than its weight.

3.1.3 Hydrodynamic drag test

To verify that the polygon mesh method of determining the drag force in combination with the buoyancy or Froude-Krylov force is producing expected results, a buoyant sphere-shaped `SMS::Payload` is placed in a translating uniform seaway moving at a speed of 7 m/s along the \hat{X} axis. The `SMS::Payload` is located at $\mathbf{P}_{E \rightarrow B} = \{0.0, 0.0, -10.0\}$, which is attached to a `SMS::Cable` whose bottom node is fixed in space, as shown in Figure 21. The `SMS::Cable`'s node 0 is attached to the `SMS::Payload` at $\{0.0, 0.0, 0.0\}$ from its body-fixed frame. The `SMS::Cable` is 10m long which means its node N is located at $\{0.0, 0.0, -20.0\}$. The spherical `SMS::Payload` surface is represented using a spherical polyhedron with 128 triangular polygons and the simulation is re-executed with a mesh of 512 polygons.

The `SMS::Payload` has a radius of 1m which gives it a volume of 4.1888 m^3 . The `SMS::Payload` has a mass of 3000kg. It has the mass moment of inertia of a solid

sphere where $I_{xx} = I_{yy} = I_{zz} = 1200 \text{ kg} \cdot \text{m}^2$. The SMS::Payload was given a drag coefficient and added mass coefficient of 0.5 in all directions.

The expected angle θ between the cable and the $\hat{\mathbf{Z}}$ axis is:

$$\theta = \text{atan}((F_d)/(F_b - mg)); \quad (48)$$

where F_d is the drag force, F_b is the buoyancy force, m is the mass and g is the gravitational acceleration.

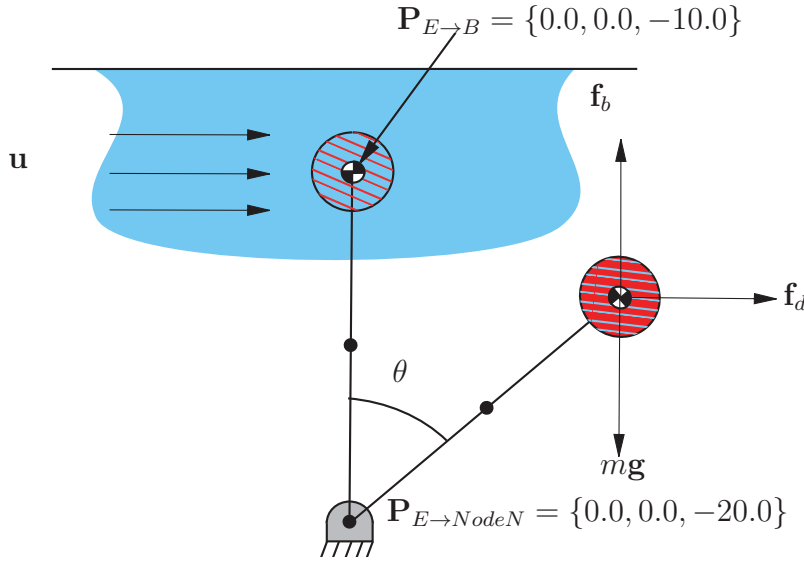


Figure 21: Experimental setup for the hydrodynamic drag test

Results

Table 6 shows the expected and the simulated final angle between the SMS::Cable and the vertical $\hat{\mathbf{Z}}$ axis when using a 128 face polyhedron well as a 512 polyhedron for the SMS::Payload. Figures 22 and 23 indicate the SMS::Payload's position over time for the 128 face polyhedron and 512 face polyhedron, respectively.

3.1.4 Fully submerged body added mass test

To verify that the fluid inertial loading is functioning as expected, the buoyancy pendulum test is reproduced with the addition of the added mass effect (Figure 24). The added mass coefficient for a sphere of 0.5 about all three degrees of freedom was chosen.

The expected period of oscillation for the pendulum with added mass is:

Table 6: The expected and simulated *SMS::Cable* angle with the vertical $\hat{\mathbf{Z}}$ axis for a 128 face mesh and 512 face mesh, respectively.

Variable	Expected cable angle (rad, relative to vertical)	Simulated cable angle (rad, relative to vertical)
τ_{pend} (128 faces)	1.2411	1.2999
τ_{pend} (512 faces)	1.2411	1.2588

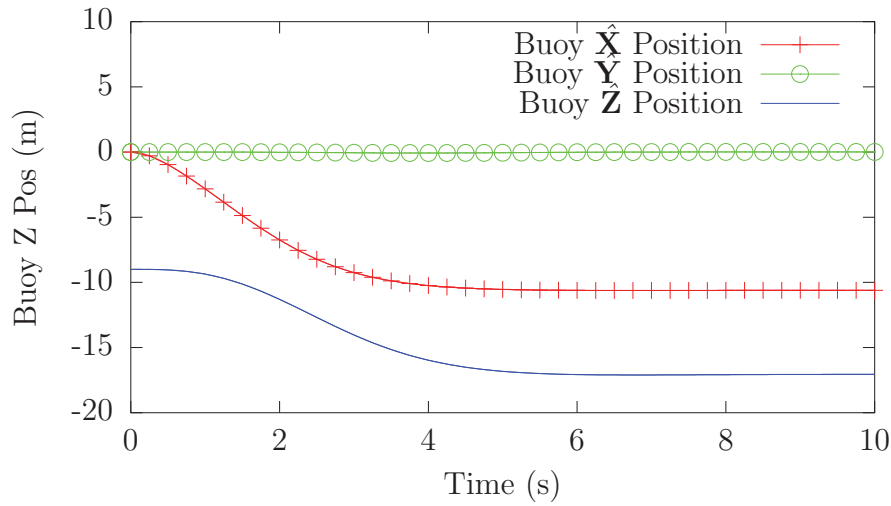


Figure 22: The simulated position of the *SMS::Payload* with a 128 face mesh

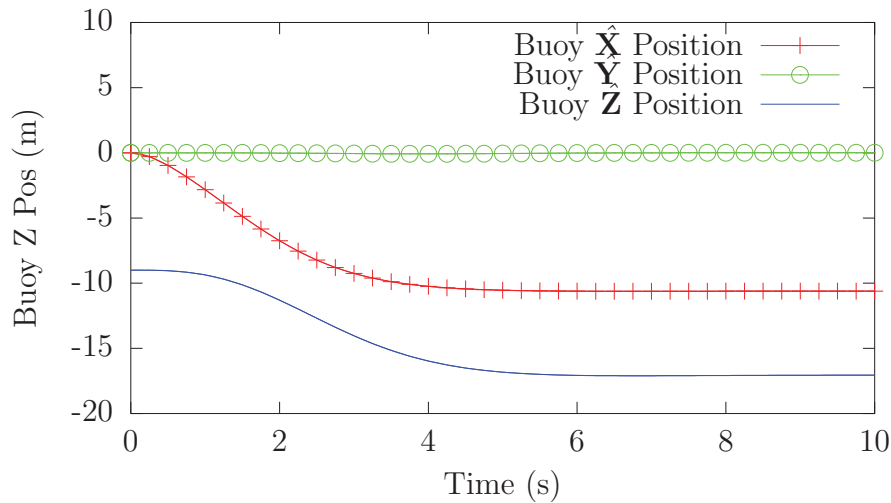


Figure 23: The simulated position of the *SMS::Payload* with a 512 face mesh

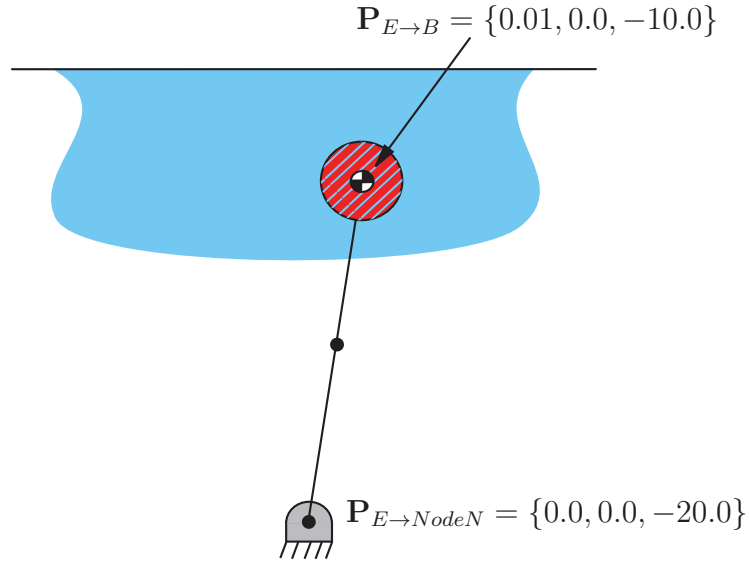


Figure 24: Experimental setup for the variable buoyancy validation test with added mass

$$\tau_{pend} = \frac{2\pi}{\sqrt{\frac{F_b - mg}{(m + m_a)L}}} \quad (49)$$

where F_b is the buoyant force:

$$F_b = \rho_w g V_{Payload} \quad (50)$$

m is the mass, m_a is the added mass, g is the gravitational acceleration, L is the length of the cable, ρ_w is the density of sea water and $V_{Payload}$ is the volume of water the payload displaces. The added mass for a fully submerged 2 m diameter sphere with an added mass coefficient of 0.5 is 2147 kg.

Results

Table 7 shows the expected and the simulated periods of oscillation for the SMS: :Payload and Figure 20 indicates its position over time.

3.1.5 Additional proposed verification tests

- **Cylinder at 45 degree inclination to a current**

This test would verify the effect of drag forces acting on an object with non-uniform drag coefficients due to flow moving past it along more than one of its axes.

Table 7: The expected and simulated period of oscillation of the *SMS::Payload*

Variable	Expected period of oscillation (s)	Simulated period of oscillation (s)
τ_{pend}	6.2008	6.3015

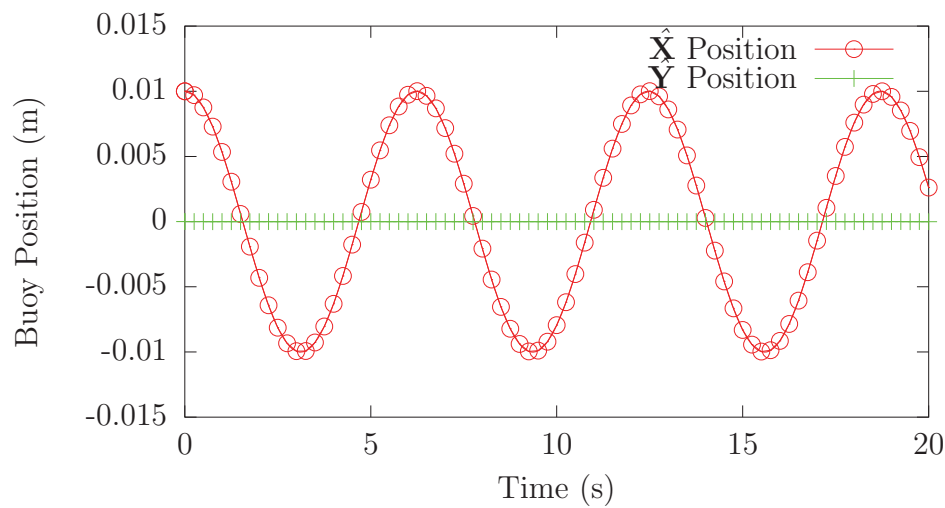


Figure 25: The \hat{X} and \hat{Y} position of a *SMS::Payload* attached to a 10 m cable pendulating by a buoyancy force larger than its weight incorporating added mass

- **Articulated body cylinder link in wave with added mass test**
This test would compare the wave forces acting on a fixed cylinder against expected analytical results.
- **Variable submergence drag test**
The body would be partially submerged, the linear drag force and results drag moment would be compared against expected analytical results.
- **Variable submergence added mass test**
The body would be partially submerged, accelerated and/or subjected to an accelerating flow. The linear added mass force, added mass, added mass moment and added mass moment of inertia would be compared against expected analytical results.
- **Axial motion added mass effect**
Objects whose geometry is symmetrical about an axis would exhibit negligible added mass when it is rotationally acceleration about that axis. The added mass moment of inertia and added mass moment would be compared against expected analytical result.

3.2 Contact dynamics validation for convex decomposed objects

The following four tests have been developed to demonstrate the proper functioning of the revised collision detection architecture. The first two tests described in sections 3.2.1 and 3.2.2 verify the convex decomposition method by comparing the results from convex decomposed objects to the same systems created without convex decomposition.

The third test described in section 3.2.3 consists of a simple multi-contact situation where the `SMS::Payload` collides with two separate collision objects. Finally, section 3.2.4 takes a rescue boat hydrodynamics hull, decomposes it into convex sub-pieces, and simulates the contact scenario between the hull and a cradle.

3.2.1 Subdivided and non-subdivided box collision tests

Two scenarios were executed using the SMS API that simulated a $1m$ cube colliding with a larger $100m \times 100m$ surface. In the first scenario, the cube and surface were subdivided into four separate sub-pieces, and in the second scenario the cube and surface were modeled as one object each. These scenarios were built to qualitatively demonstrate the ability of the SMS API to represent complex contact geometry by several sub-pieces, such that complex shapes can be simulated easily.

This simulation example consists of a $1\text{ m} \times 1\text{ m} \times 1\text{ m}$ box as the **SMS::Payload** hull impacting a $100\text{ m} \times 100\text{ m} \times 10\text{ m}$ box that is the static external contact geometry. For the decomposed (subdivided) object version of the simulation, both the **SMS::Payload** box and the external contact geometry box are subdivided into four equal convex sub pieces, as shown in Figure 26. The **SMS::Payload** has a mass of 7000 kg, and moments of inertia of $583\text{ kg}\cdot\text{m}^2$ about three degrees of freedom. Both object's material properties consist of a Young's modulus of 209 GPa and a damping factor of 10 MPa·s. The **SMS::Payload** starts at 0.05 m above the external box and falls under gravity.

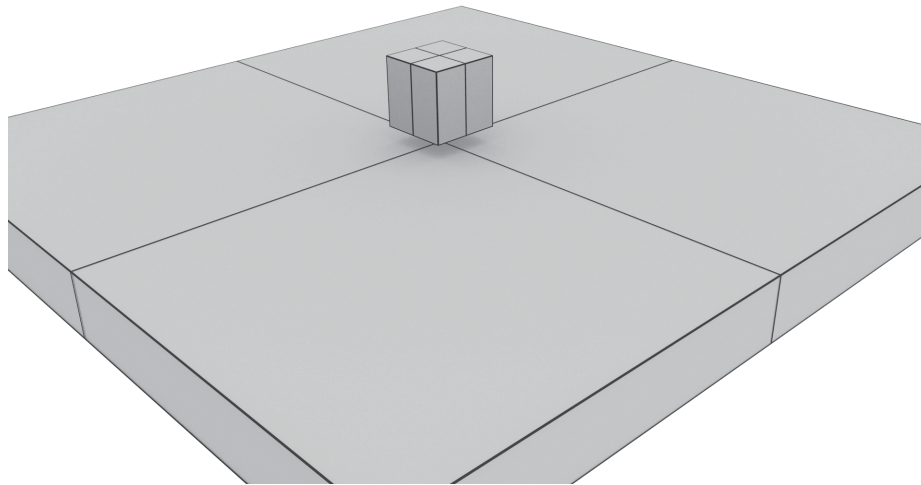


Figure 26: The simulation setup of the convex decomposed box impacting another convex decomposed box.

3.2.1.1 Results

The simulation was run until a steady state was reached. Figure 27 shows the position of the **SMS::Payload** over time. The payload starts at a \hat{Z} position of 5.55 m and settles at a position of slightly less than 5.5 m. Initial collision occurs when the **SMS::Payload** is at a height of 5.5 m. The difference in the position of the one-piece **SMS::Payload** and the convex decomposed **SMS::Payload** is negligible, as shown in Figure 27. Note that the plot lines of the convex decomposed object are plotted last, which obscures the plot lines of the one-piece **SMS::Payload**).

As shown in Figure 28, the convex decomposed **SMS::Payload**'s orientation over time is different than the one-piece **SMS::Payload**, which oscillates on the order of $1.0\text{e-}16$ radians. However, even though the orientation oscillations are larger for the convex decomposed **SMS::Payload**, they are negligible like the one-piece **SMS::Payload**. One possible reason for this is that for the one-piece object there is only one force

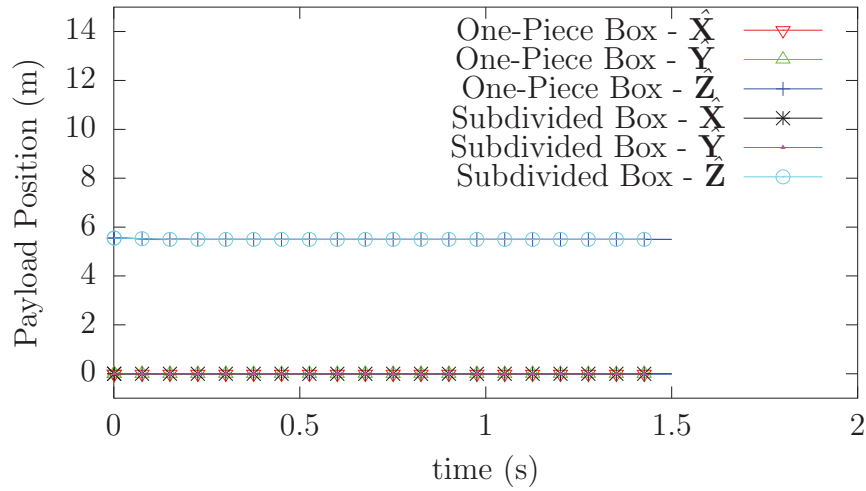


Figure 27: The simulations results of position of the one-piece and convex decomposed SMS::Payloads over time.

application point which varies rapidly to ensure self-righting moments are applied to the SMS::Payload, while for the convex-decomposed object there are four contact points whose application points vary less. This means the four individual contact forces must work to achieve a balance in the self-righting moment.

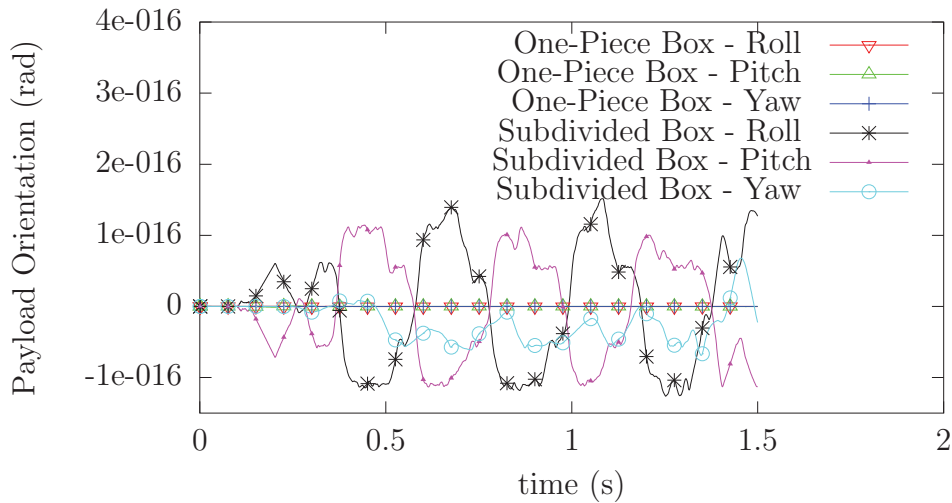


Figure 28: The simulations results of orientation of the one-piece and convex decomposed SMS::Payloads over time.

Figure 29 shows the stiffness and damping components of the normal contact force over time and Figure 30 shows the MSD/MTD and volume of interference for both the one-piece and convex decomposed SMS::Payloads. The forces, volumes and distances

are practically identical between the one-piece and convex decomposed objects. This test shows that decomposing the objects into convex sub pieces and evaluating the contact force acting on each individual sub pieces has no significant impact on the final results of the simulation.

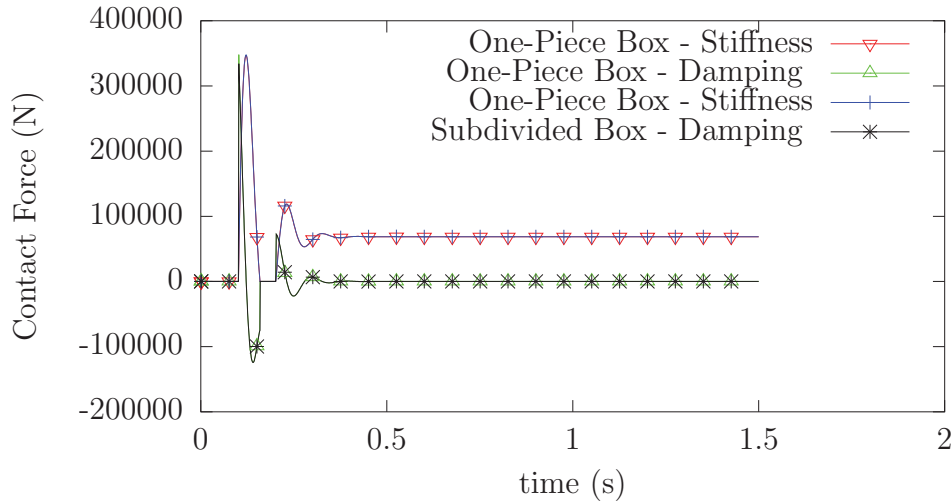


Figure 29: The simulations results of stiffness and damping components of the one-piece and convex decomposed *SMS::Payloads* over time.

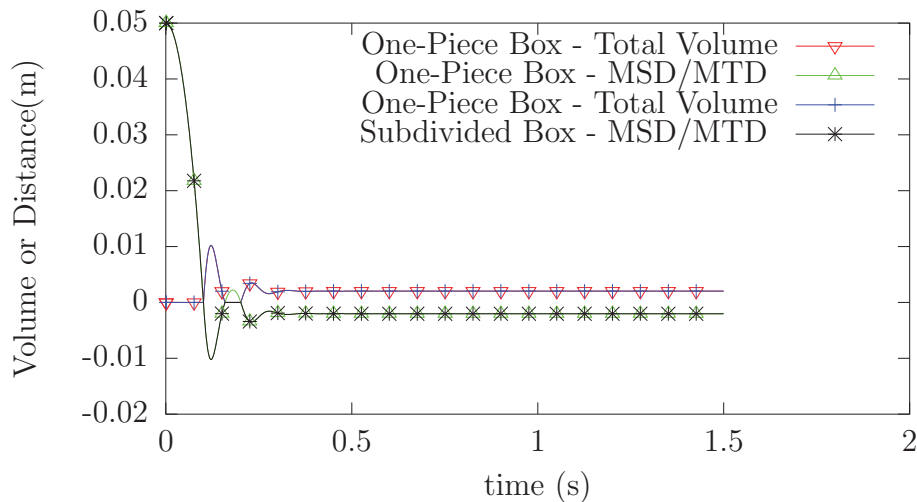


Figure 30: The simulations results of MSD/MTD and interference geometry of the one-piece and convex decomposed *SMS::Payloads* over time.

3.2.2 Oriented box collision test

This simulation example is a modified version of the subdivided box test from Section 3.2.1. The only difference is the `SMS::Payload` is given an initial orientation of 0.1 radians in roll and pitch and an initial separation of the box centers of 10 m as seen in Figure 31. All other properties are identical.

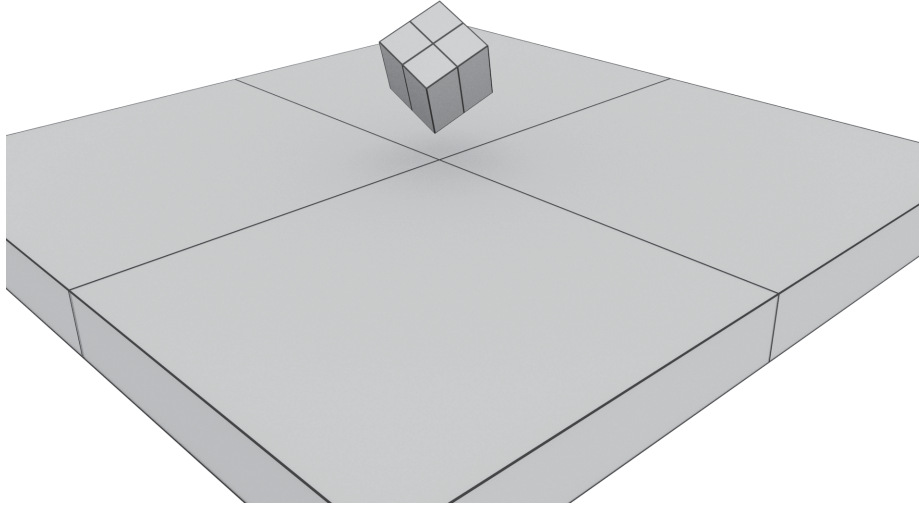


Figure 31: *The simulation setup for the oriented convex decomposed box impacting another convex decomposed box.*

3.2.2.1 Results

The simulation was run until a steady state was reached. Figures 32, 33, 34 show the position, orientation and contact force components of the `SMS::Payload` over time. The `SMS::Payload` drops onto the large flat box and due to the high damping properties of the materials, the `SMS::Payload` comes to a rest quickly. The `SMS::Payload` ends with a constant rate of rotation about the $\hat{\mathbf{Z}}$ axis due to a lack of spinning friction.

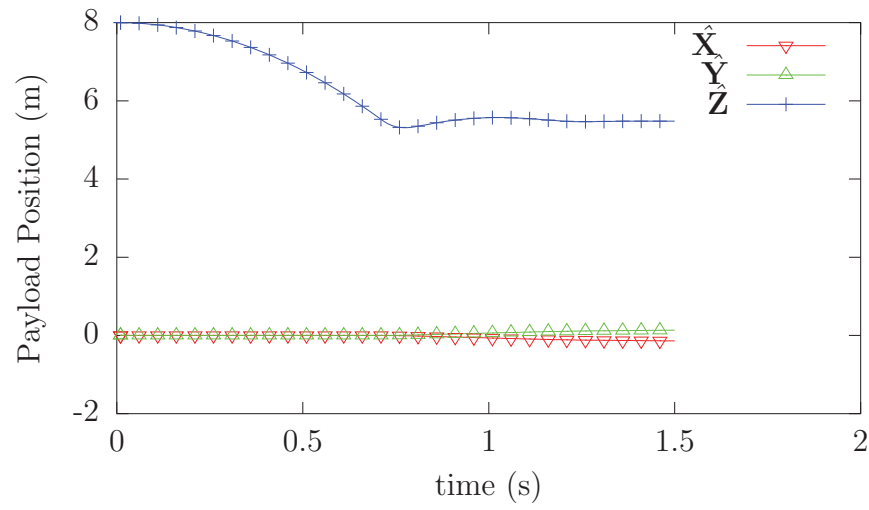


Figure 32: The simulated position for the pre-oriented convex decomposed SMS : - Payloads over time.

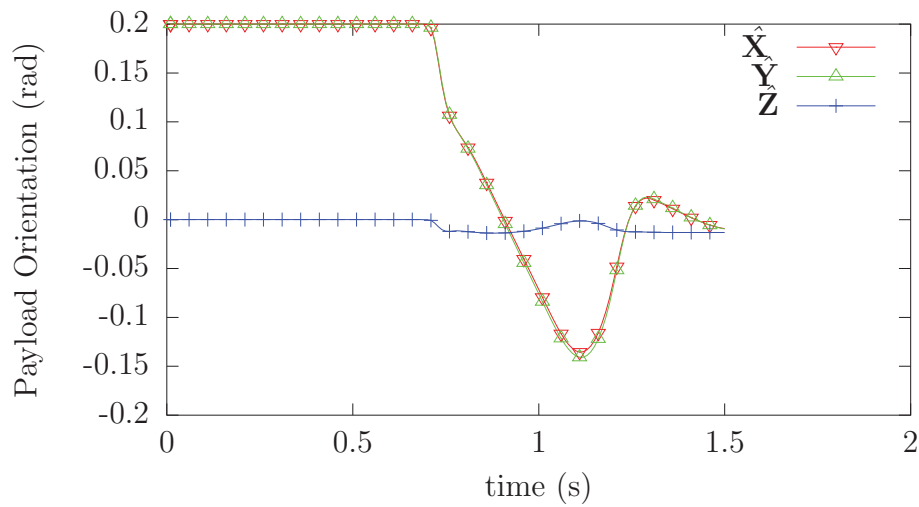


Figure 33: The simulated orientation for the pre-oriented convex decomposed SMS : - Payloads over time.

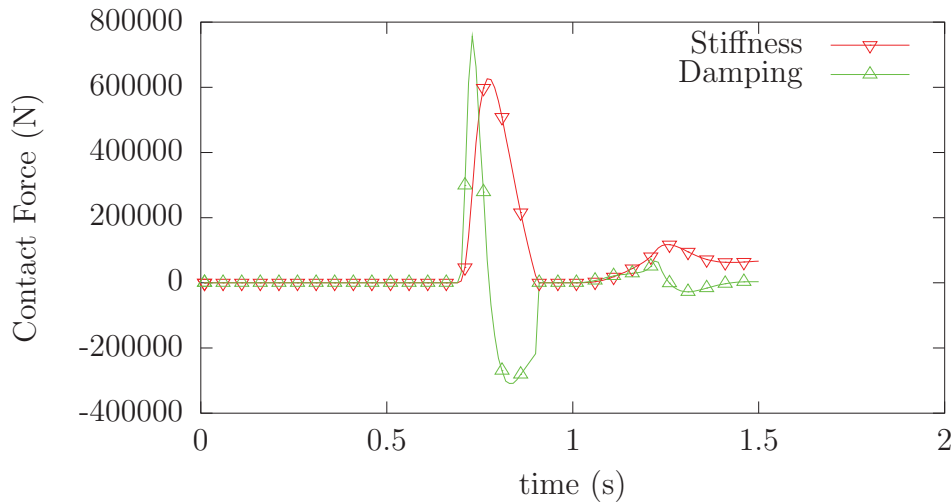


Figure 34: The simulated stiffness and damping components of the pre-oriented convex decomposed SMS::Payloads over time.

3.2.2.2 A discussion on rolling resistance

It is important to note that convex decomposition of the box into four pieces has the added benefit of distributing the volumes of interferences into four spatially spread contact volumes. When the box has an angular velocity, the individual volumes of interference will experience different rates of change, thus creating some rotational damping resistance. This effect helps to approximate the effect of rolling resistance. If the box consisted of single convex pieces, the relative angular velocity of the objects would cause no change in the volume of interference and little energy would be dissipated as indicated in Figure 35 a) and b). The individual volumes of interference for the sub-pieces of a subdivided box for the same contact situation would however see volume rates of change and thus dissipate energy due to rolling resistance, see Figure 35 c) and d). Rolling resistance has not yet been implemented in the SMS API and can only be approximated by decomposing the objects into sub-pieces.

To demonstrate this phenomena through simulation, the first 1.5 seconds of the simulation presented in this section is reproduced with a one piece box, a 4 piece box and a 16 piece box. The one piece box reacts significantly differently to the contact in Figure 36 than the 4 piece box, while the reactions of the 4 piece and 16 piece box in Figure 37 are quite similar since they both handle rolling resistance similarly. An alternative to relying on many sub-pieces to model rolling resistance that may be inaccurate and increase computational demand is the inclusion of a rolling resistance model [20].

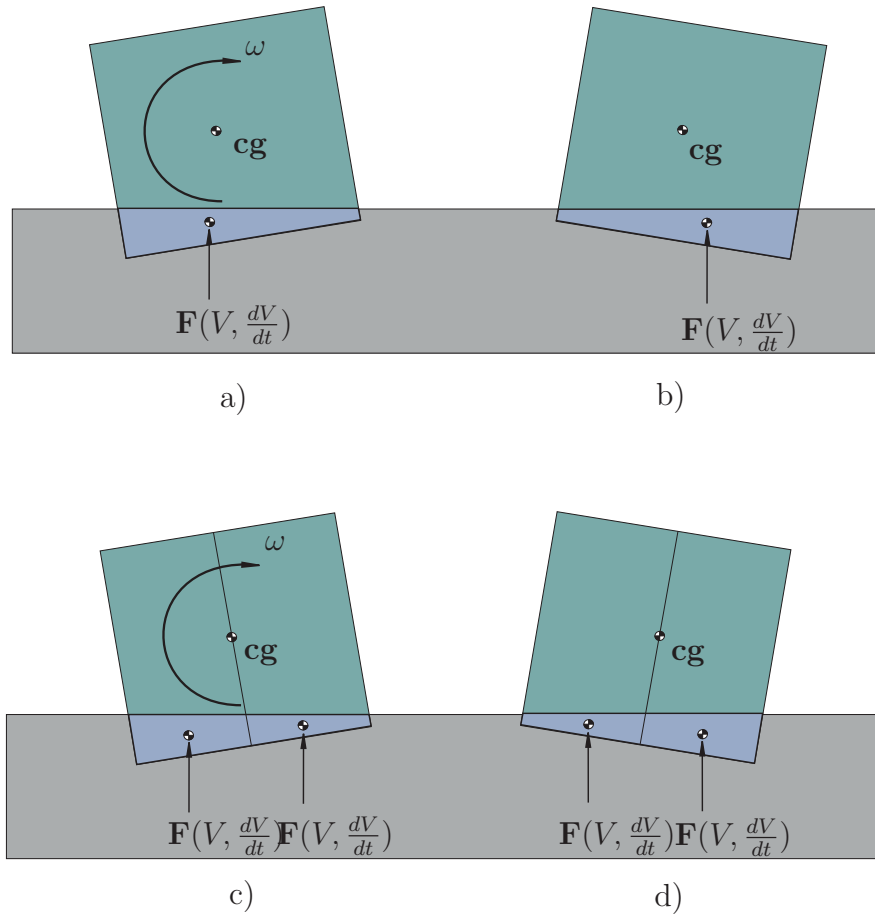


Figure 35: A description of the effect of angular velocity, ω , on the contact force without rolling resistance for a one-piece convex box and a subdivided box. The initial state of the a) one-piece and c) subdivided box, showing the contact force cause by the individual volumes of interference, and the next time step showing no change in the volume of interference for the one-piece box in b) and the changes in the volumes of interference of the sub pieces of the convex decomposed box d).

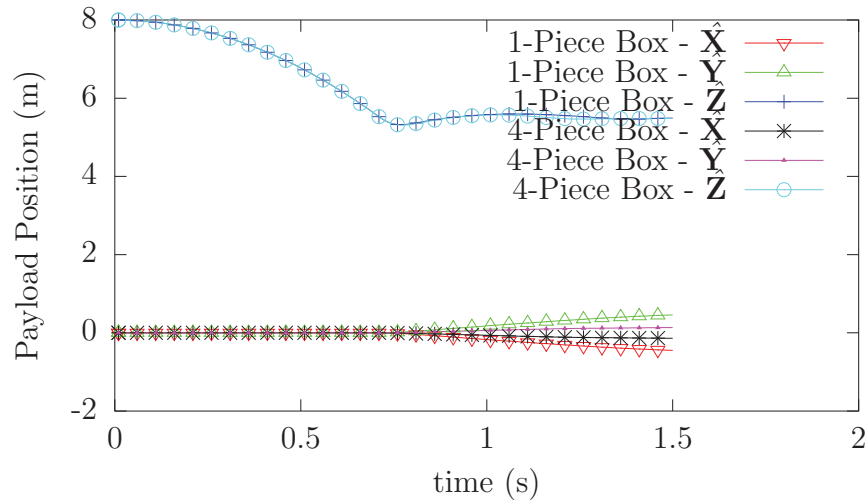


Figure 36: A comparison of the position of the 1 piece and 4 piece subdivided pre-oriented box SMS::Payloads over time.

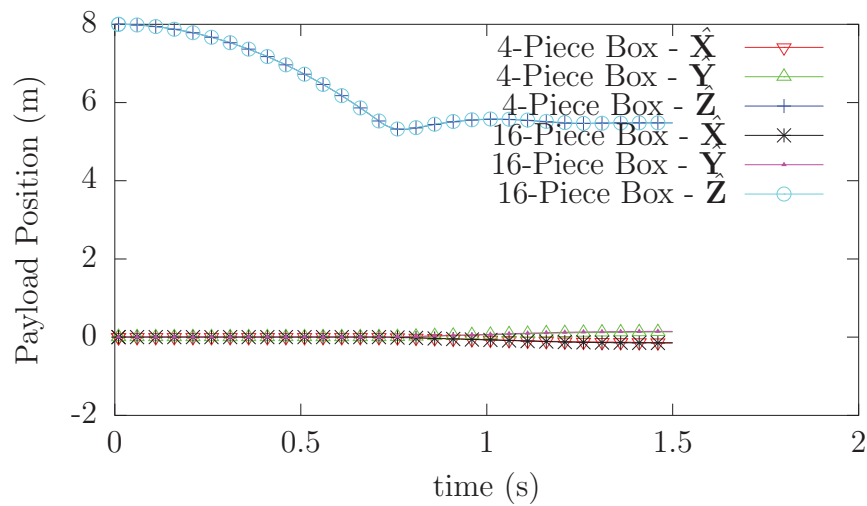


Figure 37: A comparison of the position of the 4 piece and 16 piece subdivided pre-oriented box SMS::Payloads over time.

3.2.3 Multiple contact point collision test

A scenario was created to test that the `SMS::Payload` can make contact with two separate external objects. The simulation consists of an `SMS::Payload` described geometrically with an elongated box with dimensions of $1\text{m} \times 8.25\text{m} \times 1\text{m}$ dropping onto two small $1\text{m} \times 0.9\text{m} \times 1.45\text{m}$ boxes whose centers are separated by 6.75m as shown in Figure 38. The `SMS::Payload` has a mass of $40,000\text{kg}$ and the mass moments of Inertia about the $\hat{\mathbf{X}}$, $\hat{\mathbf{Y}}$, and $\hat{\mathbf{Z}}$ axes are $3,333\text{ kg}\cdot\text{m}^2$, $233,000\text{ kg}\cdot\text{m}^2$, and $233,000\text{ kg}\cdot\text{m}^2$, respectively. Both object material properties consist of a Young's modulus of 20 GPa and a damping factor of $1\text{ MPa}\cdot\text{s}$. The `SMS::Payload` starts the simulation with a small air gap between itself and the boxes which it will come to rest on.

The collision detection system will detect two collisions, resolve two volumes of interference, and apply two contact forces to the `SMS::Payload`. The system is considered to be functioning properly if the `SMS::Payload` comes to rest on the two external boxes.



Figure 38: *The simulation setup of the multiple contact point simulation example.*

3.2.3.1 Results

Figure 39 shows that the `SMS::Payload` comes in contact with external boxes and due to the high damping of the material settles to a steady state rest. Figure 40 shows that the `SMS::Payload` oscillates back and forth about the $\hat{\mathbf{X}}$ axis until a balance is achieved as the contact forces imparts opposing moments. The contact force stiffness component shown in Figure 41 oscillates about $3.9\text{e}6\text{ N}$ to match the weight of the `SMS::Payload`, while the volume of interference in Figure 42 is negligible for the duration of the simulation due of the high stiffness of the material.

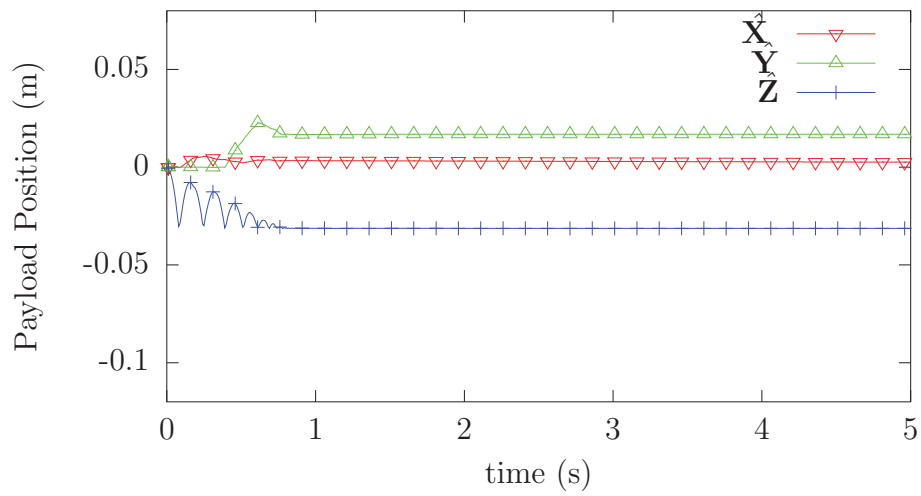


Figure 39: The position of the SMS::Payload over time.

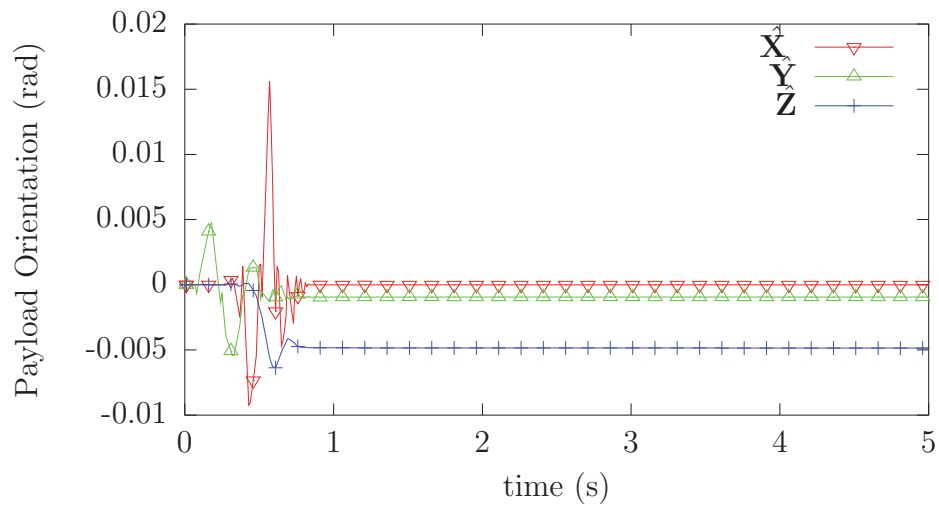


Figure 40: The Orientation of the SMS::Payload over time.

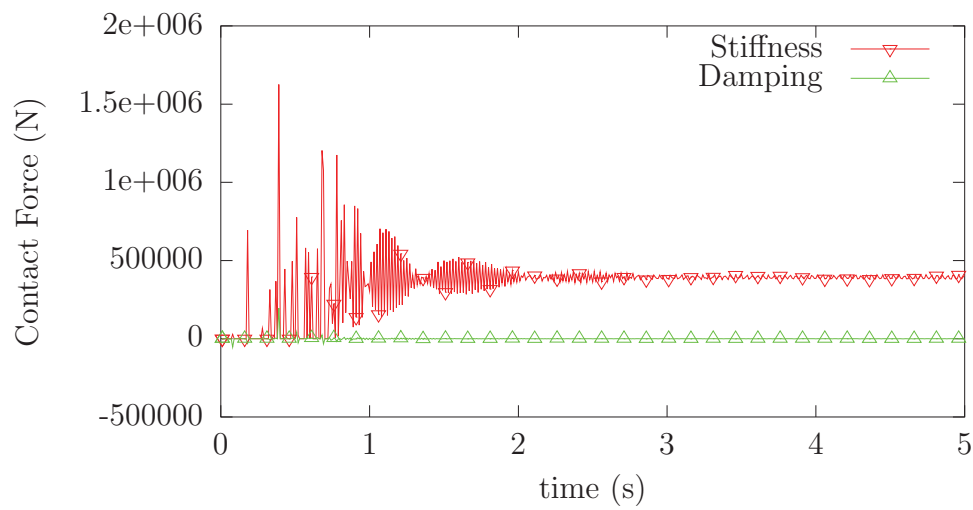


Figure 41: The contact force components of the *SMS::Payload* over time.

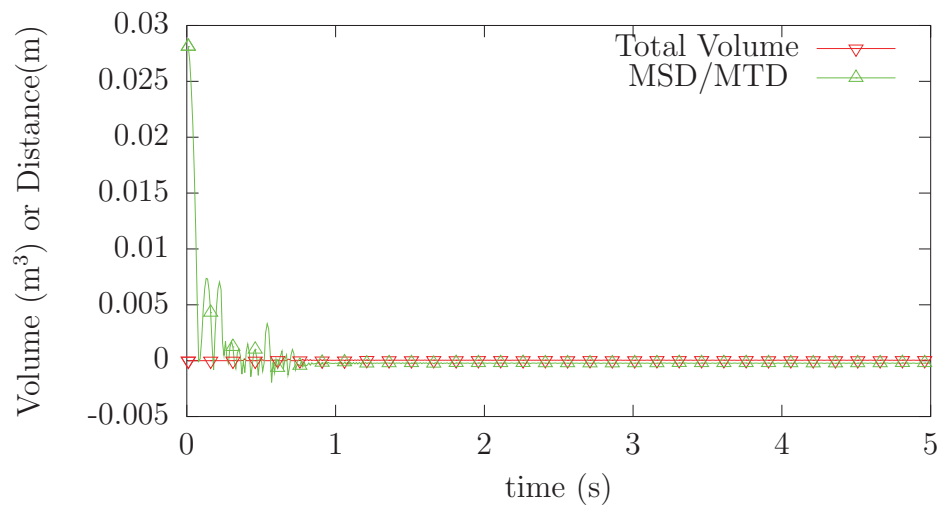


Figure 42: the volume of interference and MSD/MTD of the *SMS::Payload* over time.

3.2.4 Boat and cradle collision test

This simulation example demonstrates the ability of the SMS API to handle collision detection and contact dynamics of complex concave geometries with fine detail by decomposing it into convex sub-pieces. The `SMS::Payload` for this test case is a rescue boat. The hydrodynamics hull for the rescue boat was manually decomposed into 19 convex sub-pieces as shown in Figure 43. These were passed through the Quickhull3D algorithm prior to being used in the collision detection system to ensure their convexity. A cradle was created using four convex boxes with one heavily tapered edge that is used to support the boat as shown in Figure 44.

The material properties for the boat and cradle were both chosen to have a stiffness of 100 MPa and a damping factor of 1.0 MPa·s. The boat is initially approximately half a meter above its final resting position in the cradle. When the simulation begins, it falls under gravity until it collides with the cradle and eventually comes to rest. The sticking and sliding friction coefficients have both been set to 0.3.

3.2.4.1 Results

The boat begins from a height of 0.5 m and accelerates under gravity until it makes contact with the cradle as seen in Figure 45. When it makes first impact, it bounces several times, but the kinetic energy of the boat is quickly dissipated through material damping. The boat makes first impact in the bow, which causes it to first pitch back. However, it quickly settles to its final resting state with a slightly negative pitch angle as seen in Figure 46. There was a maximum of 20 individual collision pairs during the collision which settled to 13 while at steady state as can be seen in Figure 47.

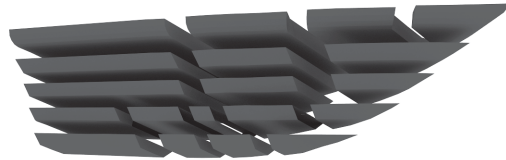


Figure 43: The decomposition of the rescue boat hydrodynamic hull mesh into 19 convex sub-pieces.

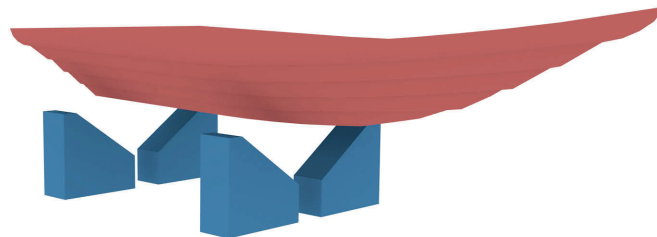


Figure 44: The boat and cradle simulation setup.

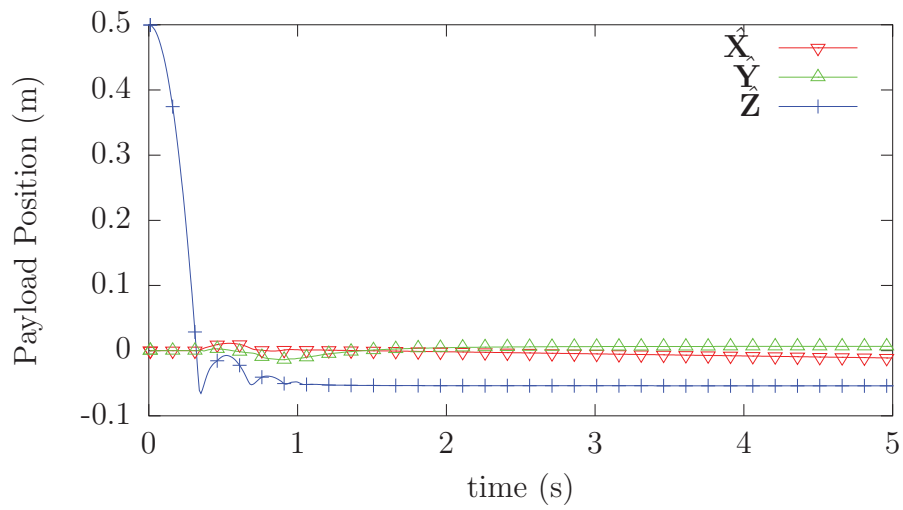


Figure 45: The position of the SMS::Payload for the boat through time.

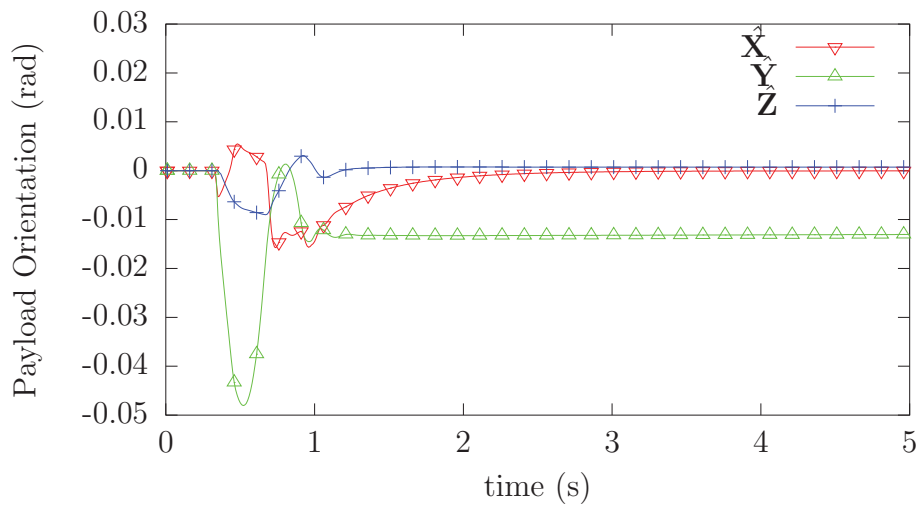


Figure 46: The orientation of the SMS::Payload for boat through time.

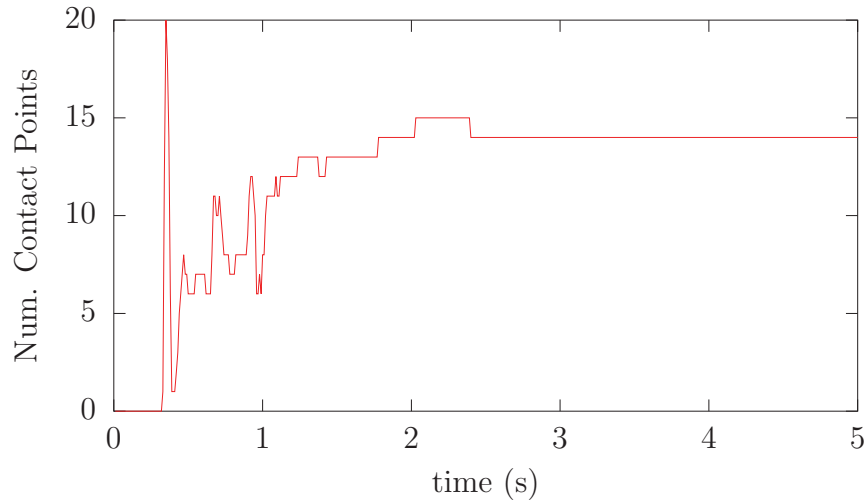


Figure 47: The number of collision pairs between the *SMS::Payload* and the cradle through time.

3.3 Generalised- α validation

Two tests were conducted to compare the performance of the Generalised- α integrator against the adaptive Runge-Kutta (RK45) integrator. The first test, presented in section 3.3.1, consists of the integration of a simple mass-spring-damper system. The second test consists of the integration of a simple gravity pendulum system using a cable, presented in section 3.3.2.

3.3.1 MSD

To verify the proper functioning of the Generalised- α integrator, a simple linear system was simulated and compared using both the Generalised- α and RK45 adaptive integrator. The simple linear system simulated here is a 1DOF mass-spring-damper. The simulation setup and results are described in Section 3.3.1.1 and Section 3.3.1.2, respectively.

3.3.1.1 Setup

The mass-spring-damper system, shown in Figure 48, consists of a mass of 1 kg, a spring with a spring constant of 1 N/m, and a damper damping factor of 0.4 Ns/m. This gives the system a natural frequency of 1 rad/s and a damping ratio of 0.2. The mass has an initial displacement from the static position of 5 m. When the simulation begins, the free vibration of the mass is driven by the reaction forces imparted by the spring and damper. Three Generalised- α simulations were run and compared against a benchmark simulation that consists of the same system integrated with an

RK45 integrator using a constant time step of $1e-3$ s. The first simulation was given a spectral radius of 0 (maximum artificial dissipation) and a time step (dt) of $1e-3$ s. The second simulation was given a spectral radius of 0 and a time step of $5e-1$ s while the last simulation was given a spectral radius of 1 (no artificial dissipation) and a time step of $5e-1$ s.

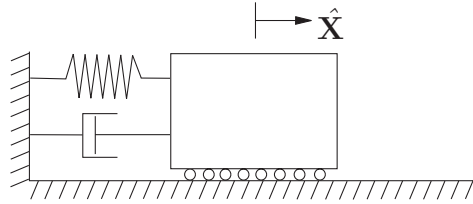


Figure 48: *The setup for the Generalised- α mass-spring-damper test case.*

3.3.1.2 Results

The simulation was executed for 50 seconds during which time the mass oscillated due to its initial offset position. The underdamped vibrations were quickly eliminated after a few oscillations. The Generalised- α simulation with a time step of $1e-3$ s and the RK45 integrator simulation provided effectively identical results. However, the Generalised- α simulation completed in 823 milliseconds while the RK45 simulation completed in 726 milliseconds. However, both integrators are constrained to integrate at the same constant rate and so this does not demonstrate the Generalized- α 's ability to integrate at a larger time step than the RK45 integrator. Increasing the time step size in the generalised- α simulation while maintaining a spectral radius of 0 had the effect of increasing the amount of dissipation while the same simulation with a spectral radius of 1 results adhered much closer to the benchmark case.

3.3.2 Pendulum

Due to the nature of the Generalised- α integrator, the numerical dissipation provided should only affect oscillations at periods smaller than the time step utilized and so more important lower frequency effects should not be significantly influenced. Furthermore, large, potentially unrealistic, structural damping values will not be needed to otherwise control these high frequency effects that could otherwise potentially destabilise the simulation as in the explicit integrator in certain circumstances. The gravity pendulum test demonstrates how the Generalised- α integrator will accurately simulate the desired lower frequency behaviour in a complex stiff multiple DOF non-linear system. In this case, the lateral pendulum oscillation is low frequency, which is unaffected, while the high frequency axial vibration induced from the cable initial conditions is damped.

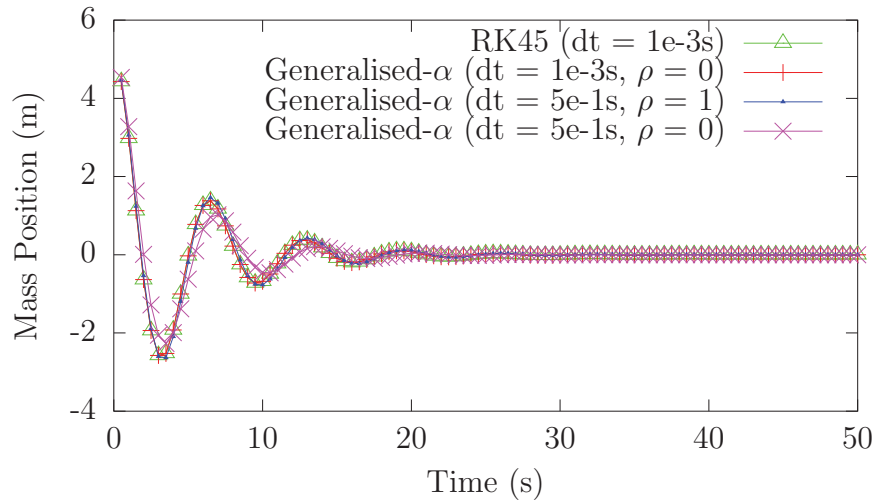


Figure 49: The time history of position of the mass for the mass spring damper using the Generalized- α integrator and the RK45 integrator.

3.3.2.1 Setup

The gravity pendulum system for this test, illustrated in Figure 50, consists of a 20 m long vertical steel cable with a diameter of 5 mm. The cable has an axial stiffness of 1.5×10^6 N. The cable has its top end fixed in space while the bottom end has a spherical mass of 5 kg attached. The cable is initially displaced by a small amount and is also given a small initial velocity to begin oscillation. The expected period of oscillation from this system is 8.97 s. There is no damping in the system, such as air drag, other the axial material damping which had to be provided to the RK45 simulation to maintain stability. The axial damping given to RK45 was 5 Ns while for the Generalised- α simulation the cable was given a material damping of 0.0 Ns.

Both the RK45 and Generalised- α simulations were run at a time step of 1.0×10^{-4} s. The Generalised- α simulation was given a spectral radius of 0 to maximize the numerical damping. The RK45 algorithm was locked at the desired time step by setting the minimum and maximum allowable time step to 1.0×10^{-4} s.

3.3.2.2 Results

Both the RK45 and Generalised- α simulations completed successfully. The positions of the mass at the end of the pendulum is plotted in Figure 51. The simulated period of oscillation matches expectations for both integrators. However, the spectral radius of 0 seems to have a damping effect on the low-frequency amplitude of oscillation of the pendulum, which was unexpected. Setting the spectral radius to 0.5 largely removes this low-frequency damping effect. Figures 52 and 53 show the tensions in the cables for both integrators. Note the rapid damping out of the high frequency

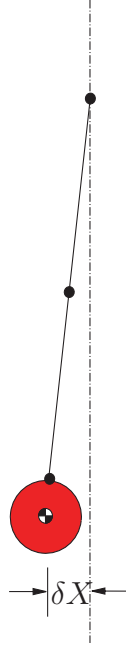


Figure 50: *The setup for the Generalised- α mass-spring-damper test case.*

axial strain vibrations seen in tension time history in the Generalised- α simulation even when there is no material damping used in the simulation.

The RK45 simulation has much larger material damping with a mild effect on the high frequency axial strain vibrations. The execution times for both the Generalised- α and RK45 simulations were 148.478 s and 146.893 s, respectively. It is anticipated that larger time steps will be allowed by an adaptive Generalised- α integrator since the higher frequency effects are damped out quickly and are a constraining factor on the stability of the simulation. An adaptive time step algorithm will likely significantly improve execution speed in the implicit numerical integrator in an analogous manner to RK45's performance benefits over RK4.

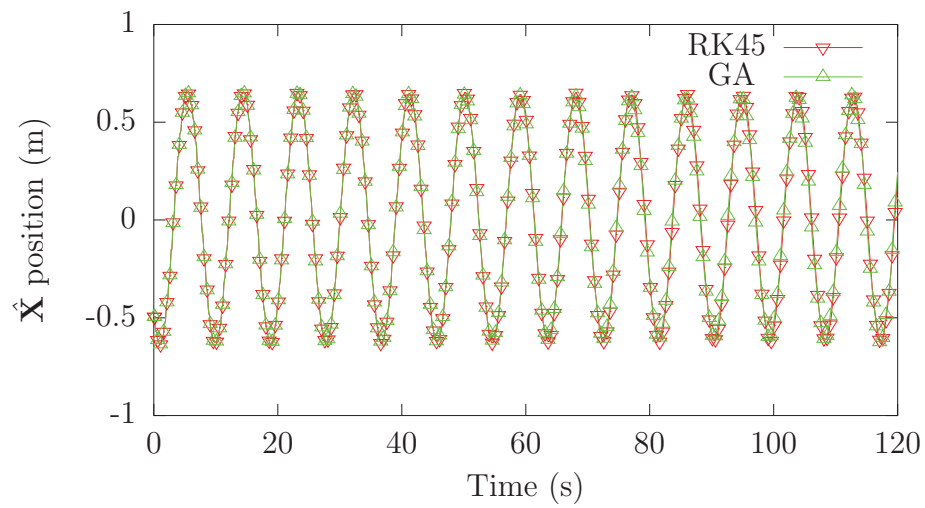


Figure 51: The \hat{X} position of the mass over time for both the RK45 and Generalised- α integrator simulations.

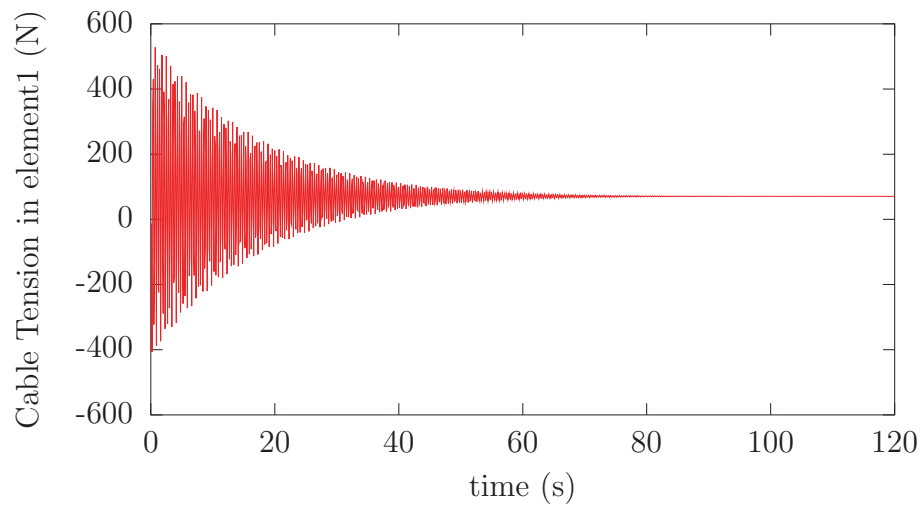


Figure 52: The cable tensions for the Generalised- α integrator simulation.

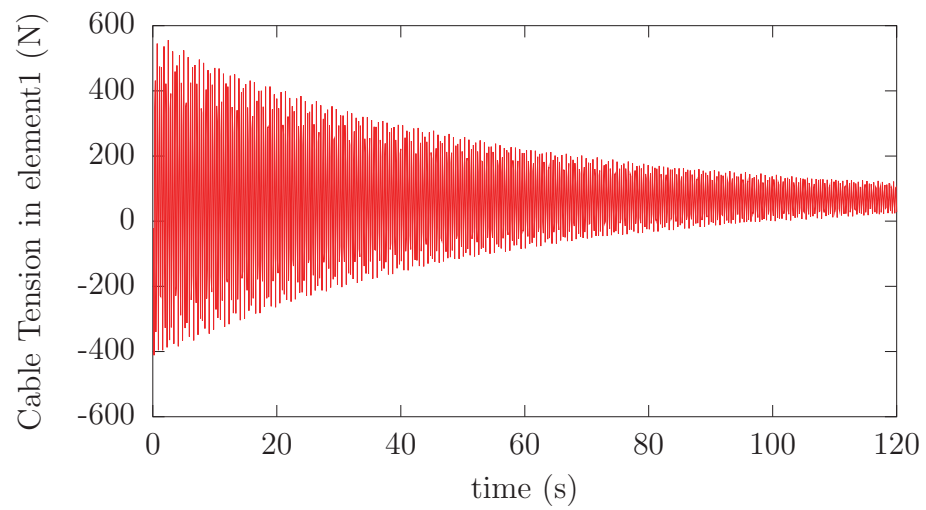


Figure 53: *The cable tensions for the RK45 integrator simulation.*

4 Launch and Recovery simulation

This section describes the process undertaken to produce the Launch and Recovery simulation, in which a small craft is launched from a frigate, then recovered. Section 4.1 describes the creation of a simplified version of the launch and recovery without contact dynamics or naval frigate motions. The results of the simulation of the simplified version showed unacceptable simulation execution speeds. In order to execute the Launch and Recovery simulation at a reasonable rate, some time was spent to optimise the code and to identify simulation bottlenecks. Section 4.2 describes these efforts and their results. Finally, contact dynamics and naval frigate motions in waves were added to the simulation to generate the final simulation. The simulation setup and results of the final simulation can be found in Section 4.3.

4.1 Simplified Launch and Recovery simulation

A simplified simulation was first constructed to demonstrate the SMS API's capability to simulate the launch and recovery of a small vessel from a naval frigate using a Palfinger-like boomcrane (Palfinger PK45000). The Simplified Launch and Recovery simulation scenario is depicted in Figure 54. The simulation begins with the Palfinger-like crane attached to a naval frigate with an `SMS::Cable` attached to its last link while the rescue boat is sitting still on the ship deck. Next, the boat is lifted off of the deck using the crane and winch. The boat is then placed in the water and the cable is detached from the boat. Next, the boat is reconnected with the cable and then placed back on board the ship using the crane and winch.

This simulation demonstrates several key capabilities of the SMS API:

- Use of the Director class within the SMS API
- Boomcrane control
- Attachment and detachment of a cable to a payload midway through a simulation
- Winch operation

4.1.1 Simulation setup

4.1.1.1 Boomcrane setup

The `SMS::BoomCrane` base is fixed in space to simulate being attached to the deck of a still ship. Its base frame relative to the Earth-fixed frame, $\check{\mathbf{P}}_{E \rightarrow p}^{(1)}$ is made of 3 Cartesian position components followed by 3 Euler angles which defines its orientation. The boomcrane's first joint frame is defined in this simulation as $\check{\mathbf{P}}_{E \rightarrow p}^{(1)} =$

$\begin{bmatrix} -1.0 & 0 & 18.5 & 0 & 0 & 0 \end{bmatrix}$. The Palfinger-like boomcrane begins the simulation initially in a folded state (see Figure 54), the initial D-H parameters for the **SMS::-BoomCrane** are defined in Table 8. The details of the Palfinger-like boomcrane's representative rigid body mass properties and CG locations can be found in [1].

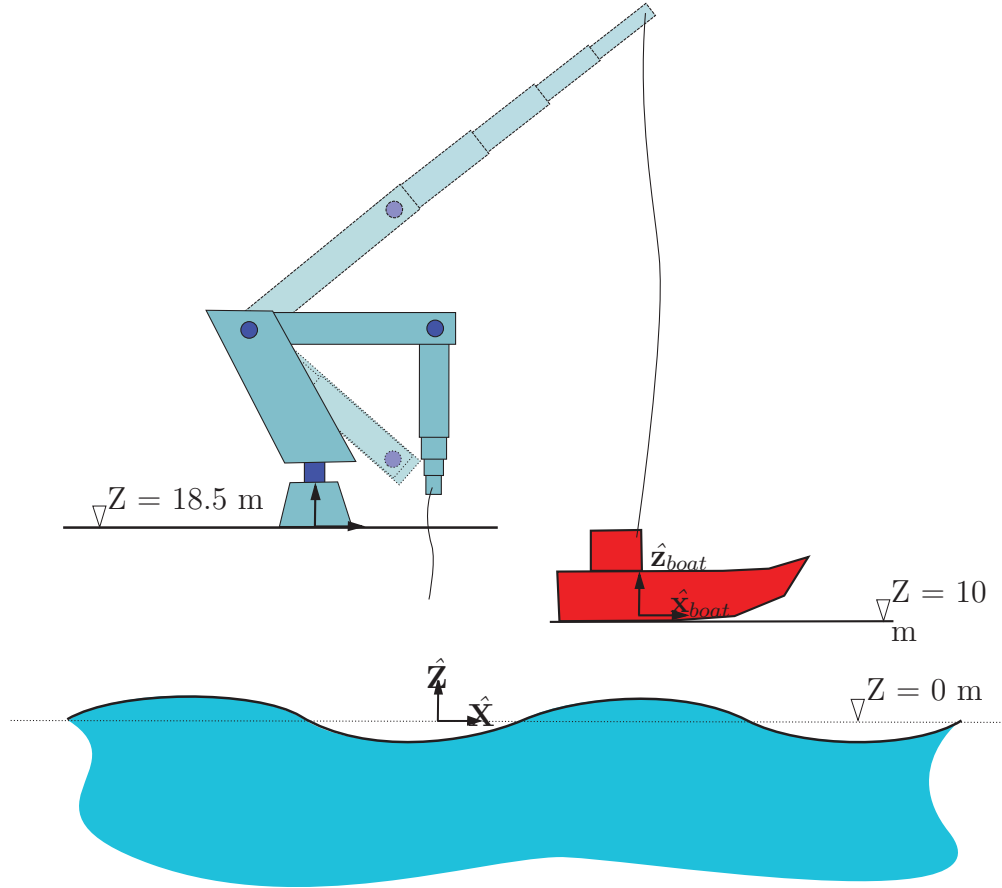


Figure 54: The simulation scenario for the Simplified Launch and Recovery simulation. The Palfinger-like boomcrane is shown folded (dotted), with a zero joint configuration (solid) and in an unfolded pose (dashed).

4.1.1.2 Rescue boat setup (SMS::Payload)

The rescue boat is a 6 m long vessel provided to DSA by DRDC in the form of a VRML geometry file, as shown in Figure 55 a). From this file, DSA generated a closed mesh polyhedron representative hull which is used by the SMS API to calculate hydrodynamic loads. As in the towing simulation, the center of gravity of the rescue boat is located 0.04 m above the baseline and 1.7 m from the stern of the vessel. The rescue boat has a mass of 500 kg and mass moments of inertia about the X, Y and Z axis of 400, 1541 and 1760 kg·m² respectively. The cable connection point relative to

Table 8: The *SMS::BoomCrane*'s initial D-H Parameters in a folded configuration for the Simplified Launch and Recovery capabilities test. Entries with a (*) represent the joint variable.

Link id	α_i	a_i	d_i	θ_i
1	90	-0.65	1.4	0*
2	0	1	0	-15*
3	90	0	0	-90*
4	0	0	0*	0
5	0	0	0*	0
6	0	0	0*	0
7	0	0	0*	0

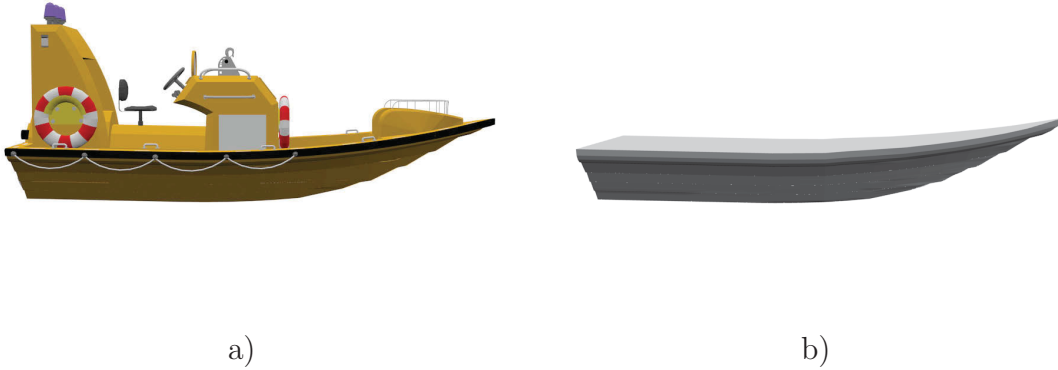


Figure 55: The rescue boat a) visualization mesh and b) hydrodynamic polyhedral mesh.

the rescue boat's body-fixed frame was set to $\mathbf{P}_{B \rightarrow C} = [0, 0, 1.5]^T$. The rescue boat begins the simulation sitting on the deck of a ship at $\mathbf{P}_{E \rightarrow B} = [2.2, 0.0, 10.0]^T$. For this simplified simulation, contact dynamics has not been enabled.

4.1.1.3 Cable Setup

The *SMS::Cable* begins the simulation as 6 m long and hanging vertically with its node N attached to the CG of the *SMS::BoomCrane*'s last link. An *SMS::Winch* is attached to the cable's node N . The cable element lengths are limited to no shorter than 1m and no longer than 7 m. The *SMS::Cable* properties are set to $EA = 1.0e7$ N, $EI = 1.0e3$ Nm² and $GJ = 1.0e3$ Nm². The cable has a density of steel and a diameter of 0.02 m.

4.1.1.4 Seaway Setup

The ocean was assumed to be calm for this simplified simulation and no waves were present in the seaway.

4.1.1.5 SMS::Director script

```
// Place a hold on the payload's position and orientation
// It's lying on the Ship's Deck
myDirector.AddCommand("testPayload hold X 2.2 holdX");
myDirector.AddCommand("testPayload hold Y 0.0 holdY");
myDirector.AddCommand("testPayload hold Z 10 holdZ");
myDirector.AddCommand("testPayload hold EX 0.0 holdEX");
myDirector.AddCommand("testPayload hold EY 0.0 holdEY");
myDirector.AddCommand("testPayload hold EZ 0.0 holdEZ");

// Fully Extend the BoomCrane
myDirector.AddCommand("testBoomCrane 1 0.7854 0.5");
myDirector.AddCommand("testBoomCrane 2 1.57 0.5");
myDirector.AddCommand("testBoomCrane 3 1.0 0.5");
myDirector.AddCommand("testBoomCrane 4 1.0 0.5");
myDirector.AddCommand("testBoomCrane 5 1.0 0.5");
myDirector.AddCommand("testBoomCrane 6 1.0 0.5");

//Pause for a second to let things settle a bit
myDirector.AddCommand("pause 1");

// Payout cable to pick up the payload
myDirector.AddCommand("testWinch testCable 12.4 1.5");

//Release the hold on the Payload and Soft Connect
//with the Cable
myDirector.AddCommand("testPayload release holdX");
myDirector.AddCommand("testPayload release holdY");
myDirector.AddCommand("testPayload release holdZ");
myDirector.AddCommand("testPayload release holdEX");
myDirector.AddCommand("testPayload release holdEY");
myDirector.AddCommand("testPayload release holdEZ");
myDirector.AddCommand("testPayload testCable connect
0.0 0.0 1.5 testWinch");

//Pause for a second to let things settle
myDirector.AddCommand("pause 1.0");
```

```

// Payin cable and move the crane's first joint by
// 90 degrees so the Payload is over the deck and
// over the water.
myDirector.AddCommand("testWinch testCable 8.0 1.0");
myDirector.AddCommand("testBoomCrane 0 1.57 0.5");

myDirector.AddCommand("pause 0.3");

// Payout cable lowering the boat into the water
myDirector.AddCommand("testWinch testCable 15 1.5");
myDirector.AddCommand("testWinch testCable 22.4 0.75");

//Disconnect the cable from the Payload
myDirector.AddCommand("testPayload testCable testWinch
disconnect 0.1");

//Play a hold on X, Y and EZ so the payload does not
//drift away too far.
myDirector.AddCommand("testPayload hold X -1.0 holdX");
myDirector.AddCommand("testPayload hold Y 3.3 holdY");
myDirector.AddCommand("testPayload hold EZ 0 holdEZ");

// Payin Cable in order to simulate a recovery.
myDirector.AddCommand("testWinch testCable 10.0 1.0");

// Wait a bit
myDirector.AddCommand("pause 2");

//Payout Cable in order to Recover the Payload
myDirector.AddCommand("testWinch testCable 22.4 1.5");
myDirector.AddCommand("pause 1");

//Release holds on the payload
myDirector.AddCommand("testPayload release holdX");
myDirector.AddCommand("testPayload release holdY");
myDirector.AddCommand("testPayload release holdEZ");

//Soft connect the cable to the payload
myDirector.AddCommand("testPayload testCable testWinch
connect 0 0.3 0.0 0.0 1.5");

```

```

//Payin Cable and rotate back over the deck.
myDirector.AddCommand("testWinch testCable 10.0 1.0");
myDirector.AddCommand("testBoomCrane 0 0 0.5");

//Payout Cable and drop the Rescue boat on the deck
//of the ship.
myDirector.AddCommand("testWinch testCable 17.17 1.5");
myDirector.AddCommand("testPayload testCable testWinch
disconnect 0.1");
myDirector.AddCommand("testPayload hold X 2.2 holdX");
myDirector.AddCommand("testPayload hold Y 0.0 holdY");
myDirector.AddCommand("testPayload hold Z 5.0 holdZ");
myDirector.AddCommand("testPayload hold EX 0.0 holdEX");
myDirector.AddCommand("testPayload hold EY 0.0 holdEY");
myDirector.AddCommand("testPayload hold EZ 0.0 holdEZ");

//Payin Cable and fold the Palfinger completing
//the simulation
myDirector.AddCommand("testWinch testCable 6.0 1.0");
myDirector.AddCommand("testBoomCrane 6 1.0 0.5");
myDirector.AddCommand("testBoomCrane 5 1.0 0.5");
myDirector.AddCommand("testBoomCrane 4 1.0 0.5");
myDirector.AddCommand("testBoomCrane 3 1.0 0.5");
myDirector.AddCommand("testBoomCrane 2 1.57 0.5");
myDirector.AddCommand("testBoomCrane 1 0.7854 0.5");

```

4.1.2 Simulation results

The simulation performs as expected, The cable is connected to the payload at 23 seconds. When the connection is made, the rescue boat's 6 DOF hold is released. Next, the cable lifts up the boat, and drops it in the ocean and disconnects at 55 seconds. The cable is reconnected at 95 seconds, lifted up and dropped back on the ship deck, where the cable is then disconnected.

The boomcrane begins in a folded state and fully extends each joint sequentially. The unfolding process is completed at around 17 seconds. After the rescue boat is attached and lifted, the SMS::Boomcrane's first joint is then actuated by $\frac{\pi}{4}$ until the boat is over the water. After the boat is recovered, the SMS::BoomCrane's first joint is actuated back to its initial folded position. The first 18 seconds has the cable moving around as the SMS::BoomCrane unfolds. The remainder of the simulation has the

cable being payed in and out through the course of the simulation.

Plots of the simulation results for the various simulation components can be found in Figures C.1 through C.4. During the simulation, there were large tension oscillations experienced by the `SMS::Cable` while the payload was attached. This is likely causing a reduced integration time step in the adaptive integrator, and therefore slowing execution speed during those phases of the simulation.

The simulation script shows that the `SMS::BoomCrane` should fold itself before the simulation is completed. The actual simulation ended in a `SMS::Cable` destabilisation during the last `SMS::Payload` disconnect procedure. It was first thought that the soft disconnect time was set to 0.1 seconds which was likely not enough to reduce the tension in the cable. Thus when the cable was disconnected under tension, the cable released with too much stored elastic energy causing a snap load on release, potentially causing the destabilisation. However, later on, through further investigation it was found that a bug in the code was the culprit.

4.2 Improving execution speeds

4.2.1 The integration timestep size

The adaptive Runge-Kutta (RK45) algorithm is the only integrator currently fully supported by the SMS API. The algorithm adjusts the integration step size to minimize the error between 4th and 5th order approximation of the solution. In situations of abrupt changes in state, the error can be high and thus the timestep size becomes reduced to minimize the error to below some tolerance. This self-adjusting mechanism of the RK45 integrator provides stability and efficient execution compared to fixed time step integrators.

Situations with large and abrupt accelerations tend to have smaller time steps. Most time-domain engineering simulations consist of objects with non-linear external disturbances acting on them and thus the integration timestep size is expected to vary over the course of a simulation as the system is perturbed.

When identifying areas of improvement in simulation scenario execution speeds, the integration step size is an important indicator. For each order of magnitude the time step size is reduced, an order of magnitude more dynamics calculations must be effectuated, which significantly decreases execution speed. On the other hand, an increase in the time step size leads to immediate reductions in the number of dynamics calculations required with much faster execution speed. Dynamics calculations for a single time step can be expensive if for example a `Payload` with contact dynamics or mesh-based hydrodynamics functionality is enabled.

In systems where `SMS::SimObjects` are in a master-slave relationship, such as for an

`SMS::Cable` attached to an `SMS::Payload`, the master `SMS::SimObject`'s integrator concatenates the slave's state vector to its own forcing the dynamics of both objects to integrate at the same rate using the master's integrator. If the `SMS::Cable` is more prone to integration error than the `SMS::Payload`, it will negatively affect the execution speed of the `SMS::Payload` since it will have to effectuate that many more dynamics calculations. This problem is aggravated when the `SMS::Payload`'s hydrodynamics feature is enabled, as those calculations tend to be costly. One way to avoid this phenomenon would be to simulate the `SMS::SimObjects` using their own integrators and their own step sizes. This functionality is not yet implemented in the SMS API. Prior to implementing independent integrators within the SMS API, each `SMS::SimObject`'s code is being investigated independently.

To identify why the integration time step was falling so low in an earlier launch and recovery simulation, the scenario was investigated in more detail. The results of the analysis are presented in Section 4.2.1.1. Based on the analysis in Section 4.2.1.1, and to further identify or isolate the source of the low time steps, 4 variations of a case representative of the operations that caused small timesteps of the launch and recovery simulation have been created. Section 4.2.1.2 discusses execution performance for these tests in detail, optimizes the `SMS::SimObject` parameters to improve performance, and compares the before and after performance results of the 4 test variations. Finally, Section 4.2.2, the full launch and recovery simulation is re-run to check the performance of the simulation with the optimized parameters.

4.2.1.1 The launch and recovery simulation

The launch and recovery simulation presented in Section 4.1 is representative of the types of operations and interactions the SMS API will be used for. A full script and description of the simulation can be found in Section 4.1. The operation, illustrated in Figure 56, consists of the following steps:

1. the crane unfolds
2. the cable pays out
3. the cable connects to the payload
4. the cable pays in
5. the crane rotates the payload overboard
6. the cable pays out
7. the cable disconnects from the payload, which floats in the water
8. the cable pays in

9. the cable pays out
10. the cable connects to the floating payload
11. the cable pays in
12. the crane rotates the payload on-board
13. the cable pays out and disconnects leaving the payload on the deck
14. the cable pays in
15. the crane folds

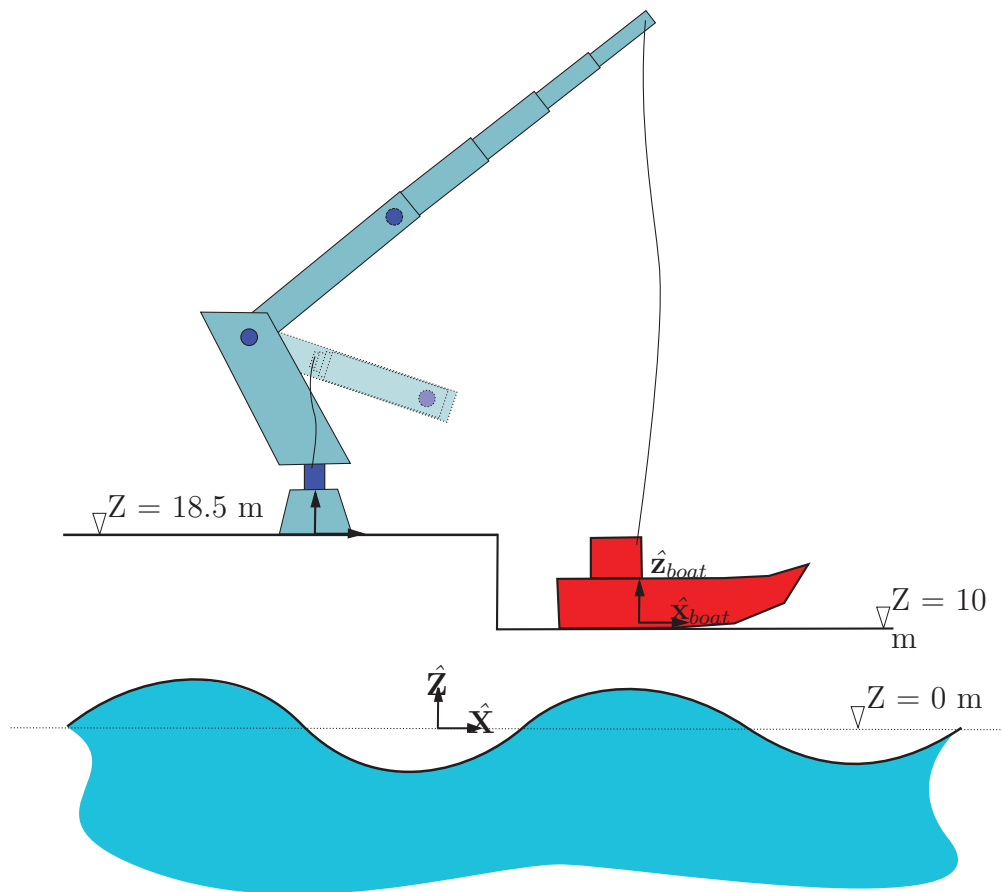


Figure 56: The Launch and Recovery simulation setup with the boomcrane in a folded configuration (dotted outline and light color) and fully unfolded configuration (solid outline and darker color).

The reader is referred to Figures C.5-C.8 to see plots of the state of the various components of the launch and recovery operation over the course of the simulation.

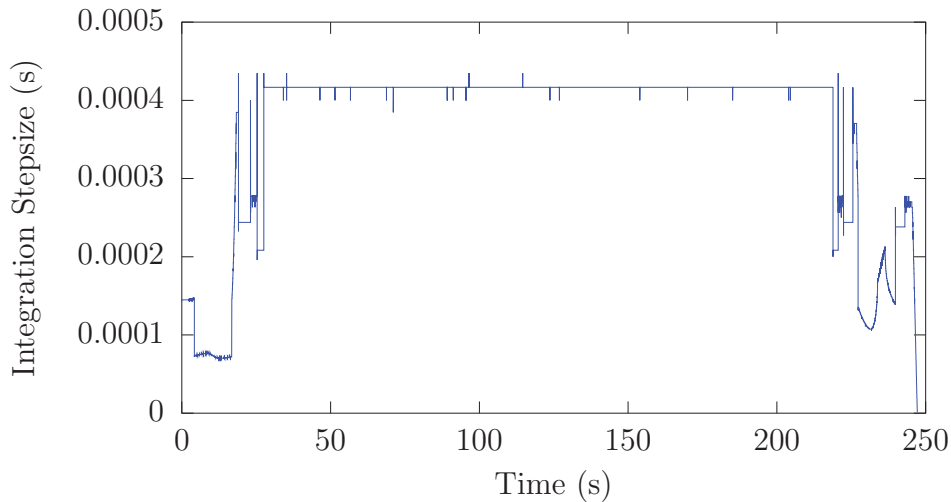


Figure 57: *The integration timestep size over the course of the Launch and Recovery simulation.*

Figure 57 shows the master `SMS::SimObject`'s (i.e. the `SMS::Boomcrane`'s) integration time step variation through the course of the Launch and Recovery simulation. Small timesteps can be seen when the `SMS::BoomCrane` is actuating. Also the best achievable time step with these properties appears limited to $4\text{e-}4$.

This indicates poor time step sizes are caused by the PID joint controllers of the `SMS::BoomCrane`. Currently, the PID controllers achieve desired actuation solely with proportional gain and viscous damping which results in a stiff system response. The `SMS::Cable` is also a possible limiting factor of time step sizes so it was also investigated as an avenue to improve the time step.

4.2.1.2 Identifying and resolving the source of small timesteps for a representative launch and recovery simulation.

Based on the analysis in Section 4.2.1.1 and to further help identify or isolate the source of the small time steps, 4 variations of a case representative of the small timestep operations of the launch and recovery simulation were created. These tests consist of the unfolded Palfinger-like boomcrane with an attached `SMS::Cable` that in turn has an `SMS::Payload` attached at its other end. The first variation consists of the `SMS::Cable` paying out by an `SMS::Winch` attached at the `SMS::BoomCrane` end. The second variation has no `SMS::BoomCrane` with the `SMS::Cable`'s free node fixed in space, the third variation is simply the `SMS::BoomCrane`, the `SMS::Cable`, and the `SMS::Payload` with no `SMS::Winch`, while the last variation is consists only of the `SMS::Cable` and `SMS::Payload` alone.

To see plots of the timestep size and the time ratios for the four setup variations

over time the reader is referred to Figures C.9 and C.10 respectively. Large oscillation amplitudes in the time step of the simulation without the `SMS::BoomCrane` are experienced and likely caused by small oscillations in the tension of the cable as the `SMS::Winch` begins to payout. Another cause is a slightly non steady state initial pre-tension in the cable from the `SMS::Payload`'s initial position. These oscillations are not present when the `SMS::BoomCrane` is attached likely because the `SMS::Boomcrane`'s timestep is lower than the `SMS::Cable`'s.

The permutation with the `SMS::Cable` and `SMS::Payload` alone has a generally large timestep of $3.5\text{e-}3$ s, especially after the oscillations caused by non-equilibrium initial conditions are damped out. When an `SMS::BoomCrane` is involved, the simulation time step is limited considerably to about $6.0\text{e-}4$ s. The presence of an `SMS::Winch` that is paying out cable also seems to be similarly limited. These findings indicate the `SMS::BoomCrane` PID controller coefficients must be better tuned to maximize the time step and therefore simulation execution speed. This process and its results are presented in Section 4.2.1.3. The `SMS::Cable` and `SMS::Winch` properties are then adjusted to help improve simulation performance. This is described in Section 4.2.1.4. The 4 test variation cases presented here were executed again using these new tuned parameters and the results of which can be found in Section 4.2.1.4.

4.2.1.3 Tuning PID controllers and damping values

The `SMS::Boomcrane` induces a small timestep with potential for improvement as discussed in Section 4.2.1.2. It was postulated that the likely source of the small timestep was the PID controller gains. The PID controller was set up using only the P-control and some joint viscous damping was used to dampen out higher frequency oscillation of the joints. To help increase the integration timestep sizes, the PID controller P gains were reduced and the I-control gain was added to help achieve the desired joint positions or velocities desired.

The original PID gains for the Palfinger-like boomcrane for the launch and recovery simulation can be found in Table 9.

The PID gain tuning process consisted of tuning one joint at a time, starting with the last joint and progressing sequentially towards the base. The tuning process used here is similar to the Ziegler-Nichols tuning process [21, 22]. To illustrate the tuning process, the second joint is used as an example. The boomcrane joint positions were configured in such a way to reasonably load the joint by fully extending the crane and adding the payload to the attached cable. With the crane configured to appropriately load the joint, the following steps are followed to tune the PID controller gains to achieve a desired joint position of 45 degrees:

1. Reduce the viscous joint damping to ensure minimal influence during the tuning process.

Table 9: Table of untuned PID controller gains for the Palfinger-like boomcrane, where only P gain and viscous damping are used. Here ξ is a placeholder for rad for joints 1-3 and m for joints 4-7.

Joint #	Position controller			Velocity controller			Joint viscous Damping co. ($N \cdot s/\xi$)
	P (N/ξ)	I ($N/(\xi \cdot s)$)	D ($N \cdot s/\xi$)	P ($N \cdot s/\xi$)	I (N/ξ)	D ($N \cdot s^2/\xi$)	
1	1000000	0	0	1000000	0	0	1000000
2	1000000	0	0	1000000	0	0	1000000
3	1000000	0	0	1000000	0	0	1000000
4	1000000	0	0	1000000	0	0	100000
5	1000000	0	0	1000000	0	0	1000000
6	1000000	0	0	1000000	0	0	100000
7	1000000	0	0	1000000	0	0	100000

2. With the I and D gains initially set to 0, increase the P gain from zero until a sustained joint oscillation can be observed partway to the desired actuation position.
3. With the P gain set, increase the I gain until oscillation about the desired joint position is achieved with a desired or reasonable response time.
4. Joint viscous damping is set to some known real-world amount or adjusted as needed to help the PID controller response and attenuate vibrations³.

The joint position PID controller load, $\tau_{PID}^{(i)}$, for joint i is defined as:

$$\tau_{PID}^{(i)} = C_{p,p}^{(i)}(\varepsilon_{target}^{(i)} - \varepsilon_p^{(i)}) + C_{I,p}^{(i)}\left(\sum (\varepsilon_{target}^{(i)} - \varepsilon_p^{(i)})dt\right) - C_{d,p}^{(i)}(\dot{\varepsilon}_p^{(i)}) \quad (51)$$

where $C_{p,p}^{(i)}$ is the position proportional gain, $C_{I,p}^{(i)}$ is the position integral gain, $C_{d,p}^{(i)}$ is the position derivative gain for joint i , $\varepsilon_{target}^{(i)}$ is the joint deflection set point for joint i , $\varepsilon_p^{(i)}$ is the actual joint deflection, $\dot{\varepsilon}_p^{(i)}$ is the joint deflection rate of change and dt is the simulation integrator time step.

For a velocity controller, the controller load is defined as:

$$\tau_{PID}^{(i)} = C_{p,v}^{(i)}(\dot{\varepsilon}_{target}^{(i)} - \dot{\varepsilon}_p^{(i)}) + C_{I,v}^{(i)}\left(\sum (\dot{\varepsilon}_{target}^{(i)} - \dot{\varepsilon}_p^{(i)})dt\right) - C_{d,v}^{(i)}(\ddot{\varepsilon}_p^{(i)}) \quad (52)$$

³Beware that adding more damping past the optimum amount will reduce the time step.

Table 10: Table of tuned PID controller gains, tuned to maximize integration timestep. No D gain is used in the presence of joint viscous damping. Here ξ is a placeholder for rad for joints 1-3 and m for joints 4-7.

Joint #	Position controller			Velocity controller			Joint viscous Damping co. ($N \cdot s/\xi$)
	P (N/ξ)	I ($N/\xi \cdot s$)	D ($N \cdot s/\xi$)	P ($N \cdot s/\xi$)	I (N/ξ)	D ($N \cdot s^2/\xi$)	
1	10000	100000	0	10000	10000	0	1000000
2	1000000	300000	0	100000	1000000	0	300000
3	1000000	1000000	0	100000	1000000	0	300000
4	100000	100000	0	10000	10000	0	30000
5	100000	300000	0	10000	10000	0	30000
6	100000	300000	0	10000	10000	0	10000
7	100000	400000	0	10000	10000	0	10000

where $\ddot{\varepsilon}_p^{(i)}$ is the acceleration of the joint deflection, and $C_{p,v}^{(i)}$, $C_{I,v}^{(i)}$, and $C_{d,v}^{(i)}$ are the velocity proportional, integral and derivative controller gains respectively.

The joint velocity PID controller gains were set in a similar way to the joint position PID controller gains. This procedure was repeated for each consecutive joint starting with the last. A manual sensitivity study was also conducted to some degree on each gain or damping factor to help optimise the time step size. The tuned PID controller gains and viscous damping ratios can be found in Table 10.

4.2.1.4 Tuning SMS::Cable and SMS::Winch properties

The SMS::Cable damping parameter and SMS::Winch acceleration settings were also modified such that the system response was not adversely affected while maximizing the integration time step. The cable damping value was changed from 5e3 Ns/m to 1e3 Ns/m while the winch acceleration and decelerations were changed from +/- 3 m/s² to +/- 0.5 m/s². This reduced excessive damping in the cable and also prevented the winch from responding too quickly and inducing shock loading in the cable.

Improved results The four variation tests from 4.2.1.2 were repeated with the tuned SMS::BoomCrane PID coefficients. For plots of the simulation time steps and execution speed time ratios over time, the reader is referred to Figures C.11 and C.12, respectively. The simulation time steps have considerably improved. The Crane-Cable-Winch-Payload case is running at a time step of about 1.0e-3s which is similar to how the Cable-Payload case ran previously. This highlights the need for well-tuned PID controller gains. The change in the SMS::Cable's axial damping property has a beneficial effect when in steady state as seen in the variations without any boomcrane.

4.2.2 Improved launch and recovery simulation execution performance

The improved `SMS::SimObject` properties obtained from Sections 4.2.1.3 and 4.2.1.4 were used to rerun the Launch and Recovery simulation in order to compare the execution performance. For plots of the simulation state of various simulation objects over the course of the Launch and Recovery simulation with improved parameters, the reader is referred to Figures C.13 to C.16 respectively. The new PID controller, cable, and winch properties have affected the system's response slightly, but more importantly it has increased integration time step considerably. The time steps are an order of magnitude larger in general as can be seen in Figure 58. This has favorably affected the simulation time ratio which is now 10:1 with the large majority of the simulation running at 3:1.

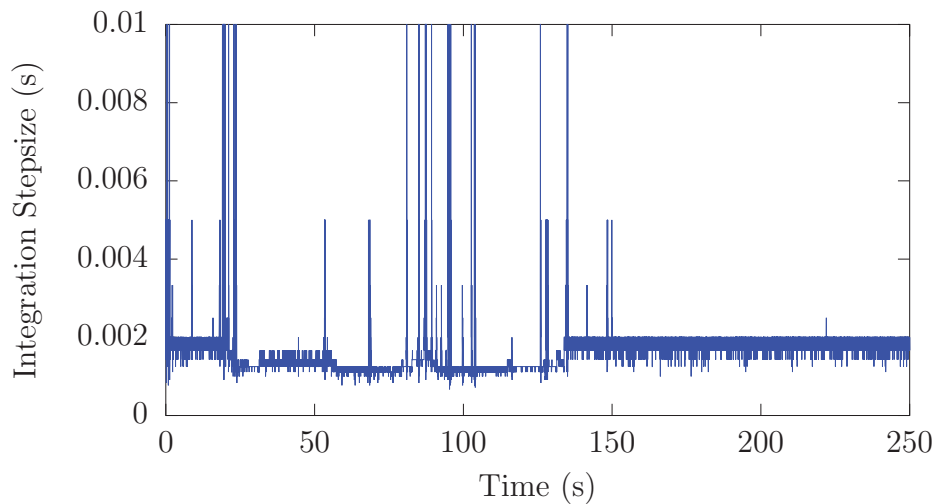


Figure 58: The integration timestep size over the course of the improved launch and recovery example simulation.

4.2.3 Profiling

To help determine how much time is spent in different parts of the code during execution, the code was instrumented and analysed using the AQTime code profiling software. Doing this has allowed DSA to identify bottlenecks in execution speed. To profile the code, a representative simulation was required. The first second of the Crane-Cable-Winch-Payload test variation from Section 4.2.1.4 was selected. The setup was modified to help profile the hydrodynamics code as the `SMS::Payload` was set to be initially floating on the water.

The code profile was obtained using AQTime and a copy of SMS API v0.1.261 and associated code profiling test suite. The instrumented code profile can be found in

Section 4.2.3.1. Based on the profile, the SMS API was updated to address some of the inefficiencies identified in the profile of SMS API v0.1.261. This updated code is versioned as SMS API v0.1.263 and its code profile can be found in Section 4.2.3.2.

Finally, the Launch and Recovery simulation was rerun using SMS API v0.1.263 to compare against the results from Section 4.2.1.1. Those results can be found in Section 4.2.3.3.

4.2.3.1 Profile before improvements

The `CalcDynamics()` functions encapsulate most of the floating point operations completed by an `SMS::SimObject` during a simulation. The rest of the SMS API code is simulation management overhead. Though the the simulation management code merits profiling, it is not as critical to fast simulations as profiling of the `CalcDynamics()` function. Table 11 shows a list of the times with children for the `CalcDynamics()` functions of the `SMS::SimObjects` with hydrodynamics. Table 12 updates the execution times for the `SMS::Payload` without hydrodynamics. Note that the `SMS::Winch::CalcDynamics()` function is not included as the time spent executing that function was insignificant.

Table 11: The execution time of SMS API `CalcDynamics()` functions for 1.0 second of simulation time with hydrodynamics calculations (SMS API v0.1.261).

Function	Time with children (ms)
<code>SMS::Payload::CalcDynamics()</code>	9016.85
<code>SMS::Boomcrane::CalcDynamics()</code>	428.32
<code>SMS::Cable::CalcDynamics()</code>	165.00

Table 12: The execution time of `SMS::Payload::CalcDynamics()` and `SMS::RigidBody::CalcDynamics()` functions for 1.0 second of simulation time without hydrodynamics calculations (SMS API v0.1.261).

Function	Time with children (ms)
<code>SMS::Payload::CalcDynamics()</code>	31.13

Each `CalcDynamics()` function in Table 11 and their related sub-functions were profiled line by line to identify if any line, function, or routine's execution speeds could be improved. The `SMS::BoomCrane::CalcDynamics()` function profile and potential sources for improvement are described in Section 4.2.3.1 with the `SMS::Cable` and the `SMS::Payload` following in Sections 4.2.3.1 and 4.2.3.1, respectively.

SMS::BoomCrane::CalcDynamics() The function `CalcDynamics()` was profiled on a line by line basis to identify the functions and operations that take the longest to execute. The majority of the issues identified as leading to poor performance for this code are due to:

- the use of functions with vectorial (e.g. `std::vector` or UBLAS vector) return types,
- the use of function-scope temporary memory allocation and destruction, and
- the use of `mutil::InvertMatrix()` on orthogonal matrices when a much more efficient transpose function would be ideally suited.

Table 13 lists the functions that took a significant amount of time to execute.

Table 13: The execution times of `SMS::BoomCrane::CalcDynamics()` sub-functions for 1.0 second of simulation time (SMS API v0.1.261).

Function	% Time of BoomCrane::CalcDynamics()	Time (ms)
ABA.RigidBody:: CalcForces()	19.10%	81.89
ABA.RigidBody:: CalcInertiaRBFrame2JointFrame()	18.40%	78.81
ABA.RigidBody:: CalcTransformationMatrices()	17.29%	74.06
ABA.RigidBody:: CalcVelocities()	11.38%	48.74
ABA.RigidBody:: GetNandBetaComponent()	4.05%	17.35

To improve the `SMS::BoomCrane`'s performance, the problems identified above were addressed. The metric for improvement is the `SMS::BoomCrane::CalcDynamics()` execution time and the execution times of the functions listed in Table 13.

SMS::Cable::CalcDynamics() The function `Cable::CalcDynamics()` was profiled on a line by line basis to identify the functions and operations that take the longest to execute. There were no major issues identified for potential improvement from the `SMS::Cable` class itself. That being said, the `SMS::Cable` class does call `SMS::RigidBody` and `SMS::Winch` functions if one is attached. These external class calls have inefficient sections of code caused by:

- the use of functions with vectorial return types,

- the use of function-scope temporary memory allocation and destruction, and
- the use of `mutil::InvertMatrix()` on orthogonal matrices when a much more efficient transpose function would be ideally suited.

Table 14 lists the functions or lines of code that took the most time execute in the `SMS::Cable::CalcDynamics()` function.

Table 14: The execution times of `SMS::Cable::CalcDynamics()` sub-functions for 1.0 second of simulation time (SMS API v0.1.261).

Function	% Time of <code>SMS::Cable::CalcDynamics()</code>	Time (ms)
<code>SMS::Cable::CalcCableProfile()</code>	35.13%	57.96
<code>SMS::Cable::CalcExternalForces()</code>	28.91%	47.70
<code>SMS::Cable::CalcInternalForces()</code>	19.97%	32.95
<code>SMS::Cable::CurvatureAccel()</code>	4.31%	7.11
<code>SMS::Cable::CalcAccel()</code>	3.87%	6.28

To improve the `SMS::Cable`'s performance, the problems identified above were resolved. The metric for improvement was the `SMS::Cable::CalcDynamics()` execution time and the execution times of the functions listed in Table 14.

SMS::Payload::CalcDynamics()

With hydrodynamics The functions `Payload::CalcDynamics()` and `RigidBody::CalcDynamics()` were profiled on a line by line basis to identify the functions and operations that take the longest to execute. The majority of the issues identified as leading to poor performance for this code are due to:

- the use of functions with vectorial return types,
- the use of function level temporary memory allocation and destruction, and
- the use of `mutil::InvertMatrix()` on orthogonal matrices when a much more efficient transpose function would be ideally suited.

Tables 15 through 17 list the functions that took the most time execute.

The hydrodynamic calculations require a considerable amount of effort because the forces must be computed individually for each one of the thousands of hydrodynamic mesh polygons. The hydrodynamic calculations were turned off to help highlight other inefficiencies in the `SMS::Payload` and `SMS::RigidBody` classes. To improve the

Table 15: The execution times of *SMS::Payload::CalcDynamics()* sub-functions for 1.0 second of simulation time with hydrodynamics calculations enabled (SMS API v0.1.261).

Function	% Time of SMS::Payload::CalcDynamics()	Time (ms)
myRigidBody->CalcDynamics()	99.84%	9002.7

Table 16: The execution times of *SMS::RigidBody::CalcDynamics()* sub-functions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.261).

Function	% Time of SMS::RigidBody::CalcDynamics()	Time (ms)
SMS::RigidBody::CalcForces()	99.83%	8987.4

Table 17: The execution times of *SMS::RigidBody::CalcForces()* sub-functions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.261).

Function	% Time of RigidBody::CalcForces()	Time (ms)
RigidBody::CalcHydroDynamicLoading()	53.33%	4793.0
RigidBody::CalcBuoyancy()	45.80%	4116.2

`SMS::RigidBody`'s performance, the problems identified above were addressed. The metric for improvement was the `SMS::Payload::CalcDynamics()` execution time and the execution times of the functions listed in Tables 15 through 17.

Without hydrodynamics The `SMS::Payload::CalcDynamics()` and `SMS::RigidBody::CalcDynamics()` functions were profiled on a line by line basis to identify the functions and operations that take the longest to execute, this time without any hydrodynamic force calculations. The majority of the issues identified as leading to poor performance for this function code are due to:

- the use of functions with vectorial return types,
- the use of function level temporary memory allocation and destruction, and
- the use of `mutil::InvertMatrix()` when a much more efficient transpose function would be ideally suited.

Comparing Table 11 with Table 12 highlights how much time is spent calculating the hydrodynamic forces versus the rest of the `SMS::Payload` and `SMS::RigidBody` dynamics calculations. This is due mostly because the hydrodynamics calculations must be repeated for each of the thousands of polygons that represent the rescue boat. Tables 18 through 20 lists the functions that took a significant amount of time to execute without hydrodynamics involved.

To improve the `SMS::Payload`'s performance, the problems identified above will be resolved. The metric for improvement shall be the `SMS::Payload::CalcDynamics()` function execution time and the execution times of the functions listed in Tables 18 through 20.

Table 18: The execution times of *SMS::Payload::CalcDynamics()* sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).

Function	% Time of SMS::Payload::CalcDynamics()	Time (ms)
myRigidBody->CalcDynamics()	69.87%	21.751
SMS::RigidBody::UpdateRBPosOri()	25.76%	7.7825

Table 19: The execution times of *SMS::RigidBody::CalcDynamics()* sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).

Function	% Time of RigidBody::CalcDynamics()	Time (ms)
RigidBody::CalcForces()	51.23%	11.143
RigidBody::CalcAccelerations()	28.75%	6.25
RigidBody::CalcTd()	19.55%	4.25
RigidBody::CalcT()	0.22%	0.05

Table 20: The execution times of *SMS::RigidBody::CalcForces()* sub-functions for 1.0 second of simulation time without hydrodynamics (SMS API v0.1.261).

Function	% Time of RigidBody::CalcForces()	Time (ms)
RigidBody::CalcContactForces()	41.68%	4.64
RigidBody::CalcHydroDynamicLoading()	17.98%	2.00
RigidBody::CalcGrav()	19.61%	2.18
RigidBody::ConvertForces()	9.25%	1.03
RigidBody::CalcBuoyancy()	6.51%	0.72
RigidBody::CalcCoriolisForce()	4.95%	0.55

4.2.3.2 Profile after improvements

Areas of poor performance were identified using AQTime. The majority of the poor performance was due to the use of functions with vector or matrix return types, the use of in-function or worse in-loop vector or matrix memory allocations, and the use of matrix inversions of orthogonal matrices when the transpose would be faster to execute. These issues have been addressed one by one and a new instrumented code profile of the same case discussed in Section 4.2.3.1 was regenerated. Table 21 shows a list of the times with children for the `CalcDynamics()` functions of the `SMS::SimObjects` with hydrodynamics. Table 22 updates the execution time for the `SMS::Payload` without hydrodynamics. Note that the `SMS::Winch::CalcDynamics()` function is not included as the time spent executing that function was insignificant.

Table 21: The execution time of SMS API `CalcDynamics()` functions for 1.0 second of simulation time, with hydrodynamics calculations (SMS API v0.1.263).

Function	Time with Children (ms)	Speed Up
<code>SMS::Payload::CalcDynamics()</code>	4762.43	1.89
<code>SMS::Boomcrane::CalcDynamics()</code>	177.63	2.41
<code>SMS::Cable::CalcDynamics()</code>	150.24	1.10

Table 22: The execution time of `SMS::Payload::CalcDynamics()` and `SMS::RigidBody::CalcDynamics()` functions for 1.0 second of simulation time, without a hydrodynamics (SMS API v0.1.263).

Function	Time with Children (ms)	Speed Up
<code>SMS::Payload::CalcDynamics()</code>	20.49	1.52

Each `CalcDynamics()` function in Table 21 and their related subfunctions were profiled line by line to compare against the profile of the SMS API v0.1.261 presented earlier. The `SMS::BoomCrane::CalcDynamics()` function profile is presented in Section 4.2.3.2, with the `SMS::Cable`, and the `SMS::Payload` following suit in Sections 4.2.3.2 and 4.2.3.2 respectively.

SMS::BoomCrane::CalcDynamics() Profiling was performed for function `BoomCrane::CalcDynamics()`. Table 23 lists the functions that took a significant amount of time to execute.

SMS::Cable::CalcDynamics() Table 24 lists the functions or lines of code that took the most time to execute in the `SMS::Cable::CalcDynamics()` function.

Table 23: The execution times of *SMS::BoomCrane::CalcDynamics()* subfunctions for 1.0 second of simulation time (SMS API v0.1.263).

Function	% Time of BoomCrane::CalcDynamics()	Time (ms)	Speed Up
ABA_RigidBody:: CalcTransformationMatrices()	15.34%	27.25	2.71
ABA_RigidBody:: CalcVelocities()	14.14%	25.12	1.94
ABA_RigidBody:: CalcForces()	12.94%	22.98	3.56
ABA_RigidBody:: CalcInertiaRBFrame2JointFrame()	7.10%	12.612	6.25
ABA_RigidBody:: GetNandBetaComponent()	2.52%	4.47	3.88

Table 24: The execution times of *SMS::Cable::CalcDynamics()* subfunctions for 1.0 second of simulation time (SMS API v0.1.263).

Function	% Time of Cable::CalcDynamics()	Time (ms)	Speed Up
Cable::CalcCableProfile()	35.31%	53.03	1.09
Cable::CalcExternalForces()	28.22%	42.39	1.12
Cable::CalcInternalForces()	19.37%	29.10	1.13
Cable::CurvatureAccel()	3.79%	5.69	1.25
Cable::CalcAccel()	5.14%	7.72	0.81

SMS::Payload::CalcDynamics()

With hydrodynamics Tables 25 through 27 list the functions that took the most time execute.

Table 25: The execution times of the *SMS::Payload::CalcDynamics()* subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).

Function	% Time of Payload::CalcDynamics()	Time (ms)	Speed Up
myRigidBody->CalcDynamics()	99.88%	4755.68	1.89

Table 26: The execution times of *SMS::RigidBody::CalcDynamics()* subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).

Function	% Time of RigidBody::CalcDynamics()	Time (ms)	Speed Up
RigidBody::CalcForces()	99.82%	4747.10	1.89

Table 27: The execution times of *SMS::RigidBody::CalcForces()* subfunctions for 1.0 second of simulation time with hydrodynamics enabled (SMS API v0.1.263).

Function	% Time of RigidBody:: CalcForces()	Time (ms)	Speed Up
RigidBody::CalcHydroDynamicLoading()	57.76%	2741.9	1.75
RigidBody::CalcBuoyancy()	41.54%	1971.9	2.09

Without hydrodynamics Comparing Table 21 with Table 22 highlights how much time is spent calculating the hydrodynamic forces versus the rest of the *SMS::Payload* and *SMS::RigidBody* dynamics calculations. The hydrodynamics calculations still dominate the execution times for the *SMS::Payload* due to the repetition of the hydrodynamics calculations for each polygon. Tables 28 through 30 lists the functions that took a significant amount of time to execute without hydrodynamics involved.

Table 28: The execution times of *SMS::Payload::CalcDynamics()* subfunctions for 1.0 second of simulation time (SMS API v0.1.263).

Function	% Time of Payload:: CalcDynamics()	Time (ms)	Speed Up
myRigidBody->CalcDynamics()	81.86%	16.77	1.30
RigidBody::UpdateRBPosOri()	17.67%	3.62	2.15

Table 29: The execution times of *SMS::RigidBody::CalcDynamics()* subfunctions for 1.0 second of simulation time (SMS API v0.1.263).

Function	% Time of RigidBody:: CalcDynamics()	Time (ms)	Speed Up
RigidBody::CalcForces()	44.27%	7.42	1.5
RigidBody::CalcAccelerations()	37.71%	6.32	0.99
RigidBody::CalcTd()	17.39%	2.92	1.46
RigidBody::CalcT()	0.36%	0.06	0.83

Table 30: The execution times of *SMS::RigidBody::CalcForces()* subfunctions for 1.0 second of simulation time (SMS API v0.1.263).

Function	% Time of RigidBody:: CalcForces()	Time (ms)	
RigidBody::CalcContactForces()	19.89%	1.47	3.16
RigidBody::CalcHydroDynamicLoading()	24.21%	1.97	1.01
RigidBody::CalcGrav()	15.84%	1.17	1.86
RigidBody::ConvertForces()	6.93%	0.51	2.02
RigidBody::CalcBuoyancy()	11.53%	0.85	0.85
RigidBody::CalcCoriolisForce()	8.62%	0.64	0.86

4.2.3.3 Improvements to the launch and recovery simulation

After improvements to the SMS API code were made using a code profiling software, the Launch & Recovery simulation from Section 4.2.2 has been rerun to highlight the improvements made in the SMS API v0.1.263 code. The 250 second simulation executed with a time ratio of 5.6:1 which is 1.7 times faster than the performance in Section 4.2.2 and 7.3 times faster than the original performance of the simulation from Section 4.2.1.1.

4.3 Full Launch and Recovery simulation

After the Simplified Launch and Recovery simulation was profiled to improve execution speed, it was expanded to include both ship motions as well as contact dynamics. To visualise the simulation results, a simple software tool was created. A screenshot taken from it can be seen in Figure 59. A geometric model of a Halifax class naval frigate was obtained from DRDC and used to represent the ship in the seaway. The ship motions for this simulation were generated from ShipMo3D's GenericFrigate sample project, which was modified slightly to suit this particular simulation scenario. The FreeMo module generated a time history of the position, velocity, and acceleration for the frigate which were then used to define the position of the base of the boomcrane as well as the position of the rescue boat cradle which is located on the ship deck. The naval frigate has no forward velocity.

In the SMS simulation, the position, velocity, and acceleration of the frigate between time points was linearly interpolated between time points obtained from ShipMo3D. These were then used to extrapolate the position of the frigate over the course of simulation.

The boat and cradle simulation files from Section 3.2.4 were used in the Launch and Recovery simulation in that the rescue boat was given the same convex decomposed contact geometry and the cradle was positioned and fixed on the ship deck. The material properties of both the rescue boat and the cradle were maintained. The rescue boat was initially sitting in the cradle. The SMS API's contact dynamics capabilities prevent interference of the rescue boat and cradle, deck, and side of the naval frigate.

A `DeepSeaway:FixedRegularSeaway` was used in SMS to model the seaway with 3 m, 10 s waves, which were the same parameters used in ShipMo3D to generate the ship's motions. To help control the orientation of the rescue boat during the recovery process and to guide it back into the cradle, a pair of cables were introduced into the simulation to act as tag lines. Tension control winches were added to represent human behavior in controlling the tag lines.

The full simulation script that governs this simulation is very similar to that found



Figure 59: A screenshot from a visualisation of the Launch and Recovery simulation, showing the rescue boat, cradle, frigate, boomcrane, and cable.

in Section 4.1 with a few minor modifications. The boomcrane starts in a folded configuration and the simulation process is as follows:

- the boomcrane unfolds,
- the lifting cable is attached to the rescue boat,
- the boat is lifted out of the cradle,
- the boat lowered into the water and detached from the cable.

From the water:

- the cable is reattached,
- the tag lines are attached to the boat,
- the boat is lifted back up,
- the boat lowered back into the cradle.

The full script can be found in the following section.

4.3.0.4 Simulation script

```
#####
#Unfold the crane
#####
myCrane 1 0.7854 0.5
myCrane 2 1.57 0.5
myCrane 3 1.25 0.5
myCrane 4 1.25 0.5
myCrane 5 1.25 0.5
myCrane 6 1.25 0.5
myCrane 0 3.14 0.75

#####
#Payout the cable and attach the boat
#####
myWinch myCable 6.5 1.5
rescueboat myCable myWinch connect 0 0.5 0.0 0.0 1.5

pause 2

#####
#Lift the boat out of the cradle
#####
myWinch myCable 4 1.0
pause 2

#####
#Rotate the crane overboard
#####
myCrane 0 1.57 0.5

#####
#Payout Cable
#####
myWinch myCable 15.5 1.5

#####
#Disconnect the boat
#####
rescueboat myCable myWinch disconnect 0.5

#####
#Payout a bit in preparation for reattachment
#####
```

```

myWinch myCable 17.5 3.5
pause 4

#####
#Connect the boat
#####
rescueboat myCable myWinch connect 0 0.5 0.0 0.0 1.5

#####
#Connect the taglines to the rescueboat
#####
tagLineMan tagLine 15 1.0
tagLineManBack tagLineBack 15 1.0

rescueboat tagLine tagLineMan connect 0 3.0 3.0 0.0 0.4
tagLineMan tagLine 75

rescueboat tagLineBack tagLineManBack connect 0 3.0 -1.66 0.0 0.4
tagLineManBack tagLineBack 75

#####
#Payin
#####
pause 3
myWinch myCable 6 0.5

#####
#Increase the tension
#####
tagLineMan tagLine 155
tagLineManBack tagLineBack 155

#####
#complete the payin
#####
myWinch myCable 4 0.25

#####
#Rotate over the cradle
#####
pause 2
myCrane 0 3.14 0.25

#####
#Payout the cable and disconnect the cable and tagline

```

```
#####

myWinch myCable 6.6 0.5
tagLineMan tagLine 30.0
tagLineManBack tagLineBack 30
pause 2

rescueboat tagLine tagLineMan disconnect 0.3
rescueboat tagLineBack tagLineManBack disconnect 0.3
pause 2
rescueboat myCable myWinch disconnect 0.3

#####
# Reel in a bit of cable
#####
myWinch myCable 4 0.25

#####
#Fold the crane, and Complete.
#####
myCrane 0 0 0.75
myCrane 6 0 0.5
myCrane 5 0 0.5
myCrane 4 0 0.5
myCrane 3 0 0.5
myCrane 2 -1.57 0.5
myCrane 1 -0.26 0.5

pause 5
```

4.3.1 Simulation results

The Launch and Recovery simulation began with the naval frigate floating in waves with no forward velocity. The boomcrane can be seen in the process of unfolding in Figure 60. The cable was then payed out, attached to the rescue boat, and lifted out of its cradle as shown in Figure 61.

The boomcrane rotated the rescue boat overboard and lowered it in to the water as shown in Figure 62. While in the water, the cable was detached from the rescue boat and was left to float there for a few seconds before both the boomcrane lifting cable and the two tag lines were attached to begin the recovery process. The tag lines were used to help control the rescue boat orientation. A snapshot of the rescue boat being lifted back up out of the water with the tag lines attached can be seen in Figure 63.

Finally, the boomcrane rotated back over the cradle, with the tag lines maintaining a steady tension to keep the rescue boat oriented with the cradle. The rescue boat was then lowered back into its cradle, and all cables were detached, as shown in Figure 64. The boomcrane then completed the simulation by folding back up to its original state.



Figure 60: A snapshot of the Launch and Recovery simulation as the Palfinger-like boomcrane begins to unfold.



Figure 61: A snapshot of the Launch and Recovery simulation as the rescue boat is lifted out of its cradle.



Figure 62: A snapshot of the Launch and Recovery simulation after the rescue boat was dropped in the water.



Figure 63: A snapshot of the Launch and Recovery simulation as the rescue boat is being lifted out of the water showing the tag lines used to prevent the boat from yawing.



Figure 64: A snapshot of the Launch and Recovery simulations with the rescue boat back in its cradle.

One of the challenges that arose when creating this scenario was that the simulation was prone to destabilisation when the rescue boat is in the cradle. In this case, the normal contact forces and friction forces between the cradle and the boat all sum up to act vertically against the rescue boat weight. Friction simulations, especially at low velocities around the stick-slip transition velocity, can be very stiff systems and explicit integrators tend to have difficulty handling such stiff system [23]. One potential solution to resolving this issue may be the development of a more advanced friction model that can mitigate difficulties at the transition velocity.

5 Tuna Clipper Towing simulation

This section describes the creation of the Tuna Clipper Towing simulation. Section 5.1 describes the creation of a simplified version of the towing simulation using the rescue boat from the Launch and Recovery simulation and kinematically moving the free end of the cable at a constant velocity. Section 5.2 describes the final Tuna Clipper Towing simulation which consists of a much larger vessel being towed in a seaway by a naval frigate whose motions were determined using ShipMo3D.

5.1 Simplified towing simulation

A simplified simulation was designed to demonstrate the towing of a small vessel through a seaway. The simulation scenario is depicted in Figure 65 shows a small vessel being towed by a cable. The cable is 20 m long and it is towing the small craft at a constant velocity through the seaway.

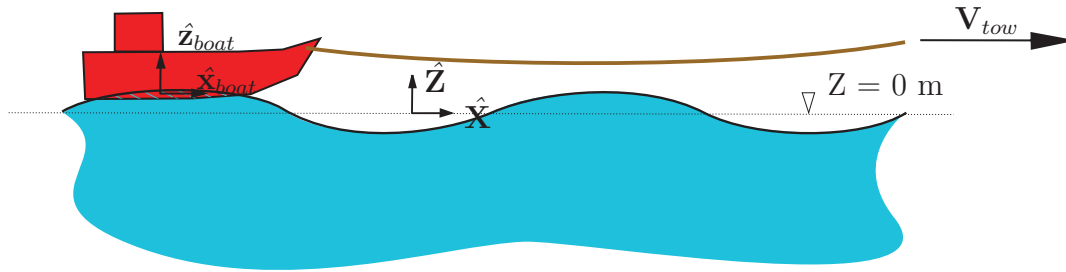


Figure 65: Simplified simulation of a small vessel tow.

5.1.1 Simulation Setup

5.1.1.1 Payload setup

The 6m long rescue boat was provided to DSA by DRDC in the form of a VRML geometry file and can be seen in Figure 66 a). From this file, DSA generated a closed mesh polyhedron representative hull, as shown in Figure 66 b), which is used by the SMS API to calculate the hydrodynamic loading.

The actual mass properties of this vessel are unknown. For the purposes of testing the SMS API, values were estimated for the key simulation parameters. The center of gravity of the rescue boat was chosen to be located 0.04 m above the baseline and 1.7 m from the stern of the vessel. The rescue boat has a mass of 500 kg and mass moments of inertia about the X, Y and Z axis of 400, 1541 and 1760 kg·m², respectively. The cable connection point relative to the rescue boat's body-fixed frame was set to $\mathbf{P}_{B \rightarrow C} = [2.73, 0, 0.5]^T$. The rescue boat begins the simulation floating in the wave at $\mathbf{P}_{E \rightarrow B} = [0.0, 0.0, 0.0]^T$.

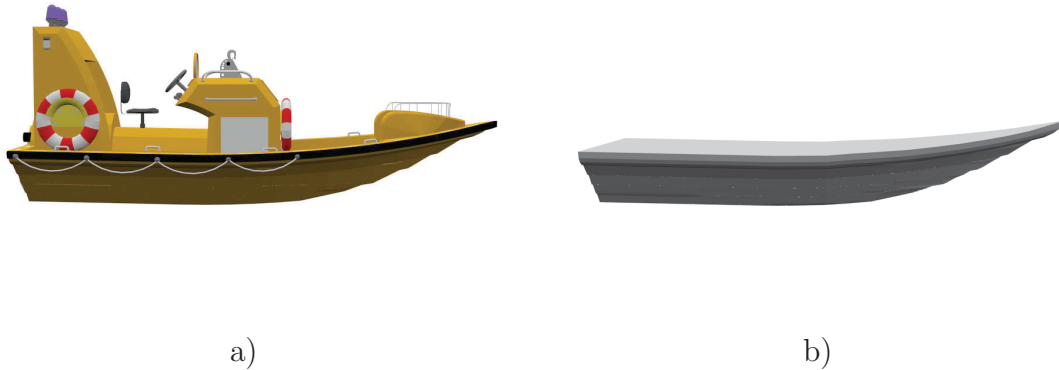


Figure 66: The rescue boat a) visualization mesh and b) hydrodynamic polyhedral mesh.

5.1.1.2 Cable setup

The `SMS::Cable` towing the small vessel is 20 m long. It begins with its node 0 attached to the small vessel and it is stretched out horizontally in the global \hat{X} direction. The `SMS::Cable`'s node N is moved in the X direction at a velocity of 1 m/s. The `SMS::Cable`'s properties are set to $EA = 1.0e7$ N, $EI = 1000.0$ Nm² and $GJ = 0.0$ Nm². The cable has a density of water and a diameter of 0.02. No hydrodynamic forces are acting on the `SMS::Cable`.

5.1.1.3 Seaway setup

The seaway consists of a single regular wave oscillating with a period of 8 seconds, and an amplitude of 0.5 m. The wave has a heading of 90 degrees.

5.1.2 Simulation results

Figures 67 and 68 shows the boat position and the `SMS::Cable`'s node N, respectively. The boat is moving with a velocity of approximately 1 m/s. The accelerations of the boat is due to the elastic cable that stretches as it pulls the boat from an initial non-moving state. The boat also heaves due to the ocean waves present.

5.2 Full Tuna Clipper Towing simulation

The final version of this simulation consists of an unpowered tuna clipper, a naval frigate, and a towing line, as shown in Figure 69. In this simulation, the tuna clipper is modeled using an `SMS::Payload` object. A visualisation geometry of the tuna clipper was provided by DRDC from which a hydrodynamics mesh was generated to provide hydrodynamics forces acting on the tuna clipper similarly to Figure 55.

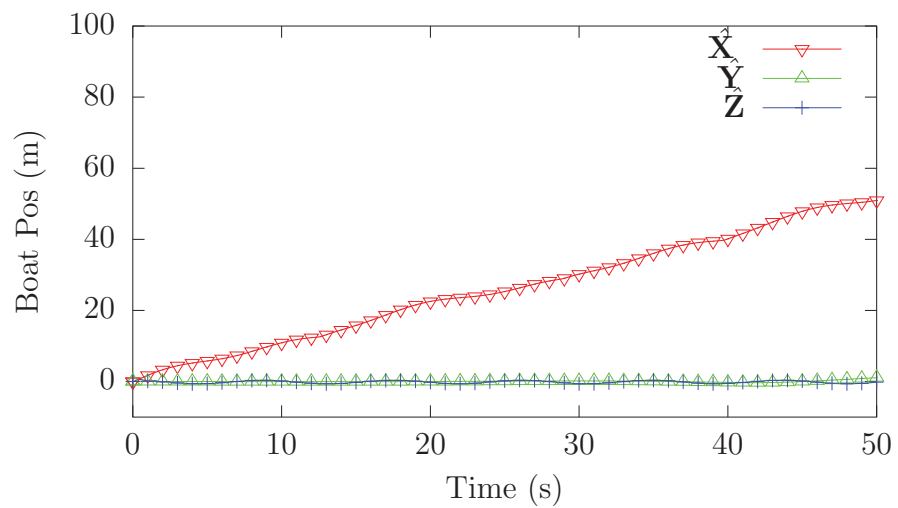


Figure 67: The simulated rescue boat vessel position over time.

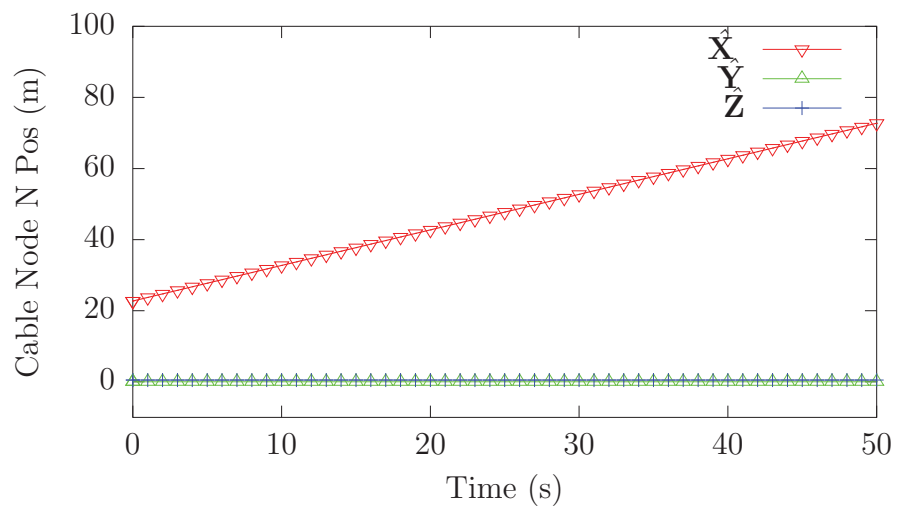


Figure 68: The simulated cable node N position over time.

The tuna clipper was provided drag coefficients of 1.5 and added mass coefficients of 1.0 for all directions. Note that the tuna clipper is about 45 m in length, and that the small body approximation (relative to the wave length) is assumed by the SMS::Payload's hydrodynamics model. The simulation of the tuna clipper violates the small body approximation and thus the results may be inaccurate. However, the simulation is intended as a proof of concept and the small body approximation allows this complex scenario to be simulated in a reasonable amount of time.

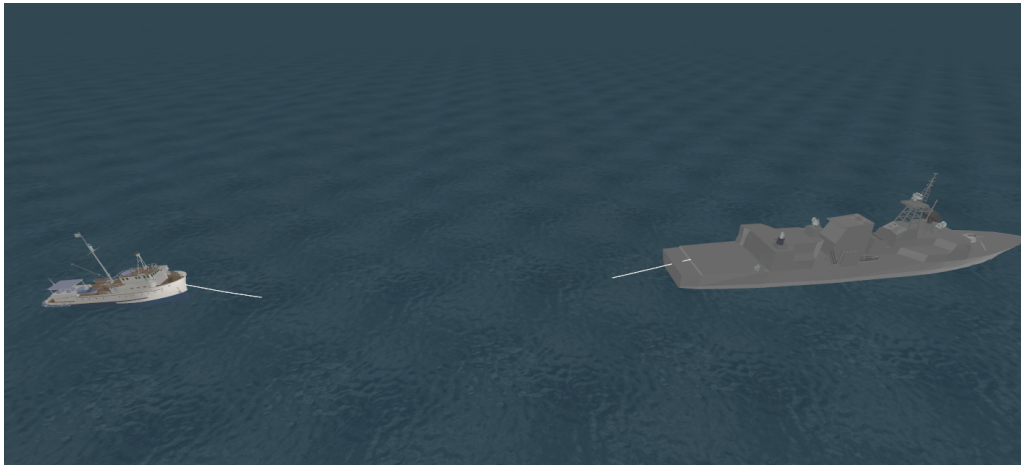


Figure 69: A visualisation of the Tuna Clipper Towing simulation.

The motions of the naval frigate were determined using DRDC's ShipMo3D simulation software a-priori and used to kinematically control the end node of the cable. The naval frigate has a forward velocity of 2.5 m/s north and is floating in a seaway with a 1.5 m wave height and 10 second period. The tuna clipper is 40 m long and was given a mass of 400,000 kg. A 175 m tow cable is attached to the stern of the naval frigate and the bow of the tuna clipper. The cable, assumed to be wire rope, has a diameter of 90 mm and an axial and bending stiffness of $5.0e8$ N and $2.5e5$ Nm² respectively.

5.2.1 Results

A full animated visualization of the results of the simulation was submitted to DRDC with this report. A screenshot of that visualisation can be found in Figure 69.

The tensions in the cable during the simulation are shown in Figure 70. The average tension in the cable is approximately 60 kN.

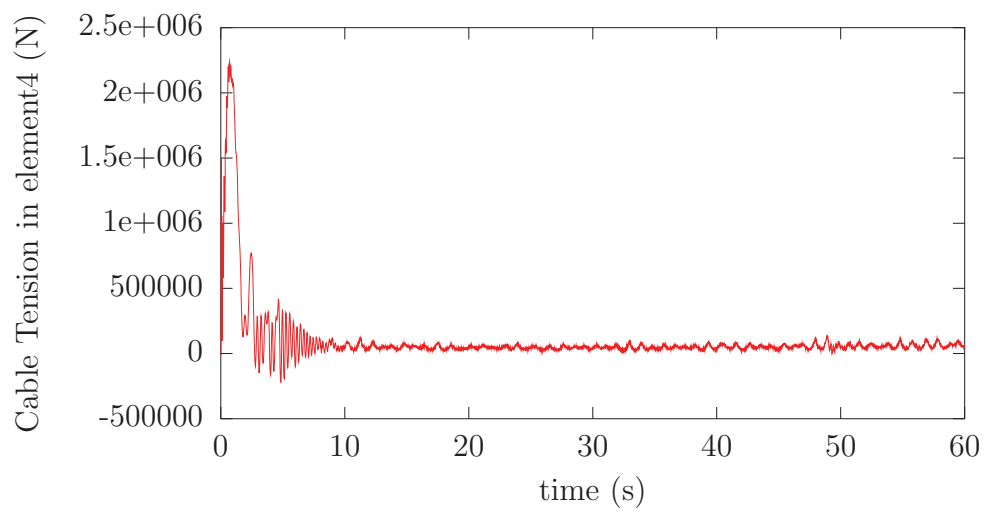


Figure 70: *The tensions in the tow cable for the Tuna Clipper Towing simulation.*

6 Future work

6.1 General

One of the requirements of the SMS API is to provide the ability to conduct a distributed simulation by communicating the boundary conditions of various `SMS::SimObjects` at regular intervals. While this capability has been implemented in a basic form, it should be expanded and properly tested.

Parallelizing of the hydrodynamics calculations could significantly improve the execution speed of simulations. This could be supported either via GPGPU programming or simple CPU multi-threading exploiting modern multi-core CPUs.

The `SMS::Payload`'s orientation is described and integrated over time using Euler angles which have the potential to undergo a singularity event called “gymbal lock”. This is likely to occur and halt the simulation if the body passes through a 90-degree change in pitch. To alleviate the potential for such an event, quaternion based orientation descriptions could be implemented as an option in the SMS API.

6.2 Winch class

A relatively simple winch model that accounts for drum inertia and braking friction could be implemented to increase fidelity of the `SMS::Winch` class.

6.3 Contact dynamics

6.3.1 Resolving N-body collisions

The current collision detection system does not scale well as it must effectuate $O(N_A N_B)$ collision checks where N_A is the number of convex geometrical features that make up the `SMS::Payload` and N_B is the number of convex geometrical features that make up the external object. The system can only account for a single rigid external object, such as the naval frigate, and so for example it cannot handle collisions between an `SMS::Payload` and an `SMS::BoomCrane`. In order to provide a scalable system to handle the ability to resolve the N-body collision problem, the implementation of a broad-phase collision detection system would be required. The foundation for such a system has already been laid in the SMS API.

6.3.2 Continuous collision detection

The current SMS API contact resolution system only checks for instantaneous collisions and does not account for tunneling. Tunneling occurs when two objects pass through each other in between time steps and will therefore not be detected without

the use of continuous collision detection. To handle tunneling, continuous collision detection methods would be implemented, such as through the use of swept volumes. Tunneling is not an important phenomena to detect if the objects are large, and velocities as well as time steps are small.

6.3.3 Rolling resistance implementation

Rolling resistance was discussed in Section 3.2.2.2 of this report. Rolling resistance is currently not accounted for in the current contact model implementation, though a crude effect is produced by hull partitioning.

6.3.4 Friction model improvements

An improved more continuous friction model could increase run times by reducing instabilities that cause time-step reductions in the integrator. A potential candidate for investigation could be the Gonthier *et al.* version of the LuGre friction model [24]. In addition, spinning friction is currently not implemented. A simple Karnopp-style friction model is currently implemented. This model can have detrimental effects on the integrator in high friction situations.

A more sophisticated friction model to help model lubricated conditions like wet ship decks should also be implemented for increased accuracy. The LuGre friction model would account for this. Squeeze-film lubrication could also be taken into account and might be important under some circumstances.

6.3.5 Volume of interference research

Computing the volume of interference is currently the execution speed bottleneck when a collision is occurring. Improvements in execution speed of simulations with contact dynamics may require research into alternative methods of determining the volume of interference. Computing the normal contact force direction using the MTD as well as the use of the volume of interference centroid as the contact point are approximations that have some limitations.

6.3.6 Volume of interference limitations

The volume of interference algorithm implementation is limited to double precision. Errors can build up quickly after a few numerical operations in certain circumstances. A problem has been identified when the integration time step is small during the first occurrence of a collision. Volumes of interference smaller than $1\text{e-}9 \text{ m}^3$ are generally not detectable due to numerical precision and the algorithmic tolerances required to handle such precision errors. This can lead to a very large instantaneous volume

rate of change destabilizing the simulation. That is, the smallest detectable volume will never be much smaller than $1\text{e-}9 \text{ m}^3$ which will get divided by the time step to obtain the volume rate of change. The time step can drop very low, while the volume rate of change stays constant at the lowest discrete detectable level, leading to unrealistically large damping factors. Currently, a workaround to this problem has been implemented where the damping term is artificially limited to prevent this situation from destabilizing the simulation. An alternative to this approach might be to use higher precision data types offered by, for example, the boost library, though the use of such data types would come at the expense of execution speed. Logic could potentially be used to switch between regular doubles to higher precision data types when situations requiring higher precision are encountered.

6.3.7 Winkler elastic bed depth

Computing the Winkler elastic bed depth is currently based on the radius of a sphere with the same volume of the sum of the convex sub pieces of the contact geometry. This is an approximation and more accurate methods of defining the elastic bed depth could be identified.

6.3.8 Improving normal contact model fidelity

Currently the normal contact force model is modelled using a Winkler elastic bed depth where each spring is free to move independently of its neighbours. To improve the contact dynamics model, three dimensional contact stress effects, such as accounting for Poisson's ratio, for the contact model could be investigated.

6.4 Cable improvements

There are several options for further development of the cable model.

6.4.1 Clamped terminations

The current cable model only supports pinned cable connections. Though clamped cable connection `SMS::Payload` and `SMS::Boomcrane` objects would only be needed in rare circumstances, particularly when flexural stiffness of the cable is high, it may be desirable to implement that capability.

6.4.2 Implicit numerical integration

Promising results have been obtained with the implicit numerical integrator. In the simple test cases studied so far, it successfully damps out high frequency effects.

However, the simulation execution speed can likely be significantly increased by implementing an adaptive time step capability. This requires a method to assess error in the simulation results as a function of time step. Generally, it is difficult to assess error as computationally inexpensive as the RK45 algorithm does. However, a literature review indicates that an error control algorithm is available for the Generalised- α method that affords significant increase in simulation execution speed and this should be investigated. In addition, more insight is needed as to why validation experiment pendulum oscillations decay when such a small time step is utilized in conjunction with maximum numerical dissipation, which was unexpected.

The implicit integrator is currently only supported by the mass spring damper and cable models in isolation. Effort should be expended to facilitate using a separate integrator for the cable when used in conjunction with the boomcrane and payload models.

6.5 Launch and Recovery simulation

The Launch and Recovery simulation showed that the simple friction model used by the contact force calculation model may induce numerical instability due to the abrupt transition between static and dynamic friction. One way to address this issue would be to use a friction model that has a more continuous approach in modeling the transition between static and dynamic friction and also includes a viscous component, as mentioned in the contact dynamics recommendations.

6.6 Tuna Clipper Towing simulation

The Tuna Clipper Towing simulation revealed the need for further software development to allow for environment load modeling for the cable due to the ShipMo3D seaway. Additional development is needed to couple seaway effects with the cable hydrodynamics model.

7 Conclusion

The preceding report reviews improvements to the SMS API, namely:

- the addition of a hydrodynamics model for the `SMS::Payload` class,
- the inclusion of the `ShipMo3D::DeepSeaway` class via DLL to model fluid environment,
- the creation of the `SMS::Director` class to facilitate the scripting of simulation events,
- the general improvement of code performance by identifying and resolving bottlenecks,
- the improvement of the collision detection code by implementing a new collision detection software architecture, the EPA algorithm, and exploiting temporal coherence,
- the investigation and implementation of the Generalised- α implicit integrator for cables.

The new additions and improvements made to the SMS API have enabled the simulation of the launch and recovery of a small vessel in waves as well as the towing of a small vessel through a seaway. These two simulations were set up, simulated, and visualized. The visualizations were submitted to DRDC as attachments along with this report.

References

- [1] Roy, A., Steinke, D., and Nicoll, R. (in review). Ship Mechanical Systems API – Final Report: Modeling Methodologies, API User Manual, API Validation. (DRDC Atlantic CR 2010-256). Defence Research and Development Canada – Atlantic.
- [2] McTaggart, K.A. (2003). Modelling and Simulation of Seaways in Deep Water for Simulation of Ship Motions. (DRDC Atlantic TM 2003-190). Defence Research and Development Canada – Atlantic.
- [3] Fish, P. R., Dean, R. B., and Heaf, N. J. (1980). Fluid-structure interaction in Morison’s equation for the design of offshore structures. *Engineering Structures*, **2**, 15–26.
- [4] Det Norske Veritas (2007). Recommended practice DNV-RP-C205 Environmental conditions and environmental loads.
- [5] White, F. M. (1999). Fluid Mechanics, fourth ed. Toronto: McGraw-Hill.
- [6] Dobrovolskis, A. R. (1996). Inertia of any polyhedron. *Icarus*, **124**, 698–704.
- [7] McTaggart, K.A. (2007). ShipMo3D Version 1.0 User Manual for Simulating Motions of a Freely Maneuvering Ship in a Seaway. (DRDC Atlantic TM 2007-172). Defence Research and Development Canada – Atlantic.
- [8] McTaggart, K.A. (2002). Three Dimensional Ship Hydrodynamic Coefficients Using the Zero Forward Speed Green Function. (DRDC Atlantic TM 2002-059). Defence Research and Development Canada – Atlantic.
- [9] McTaggart, K.A. (2003). Hydrodynamic Forces and Motions in the Time Domain for an Unappended Ship Hull. (DRDC Atlantic TM 2003-104). Defence Research and Development Canada – Atlantic.
- [10] Sarpkaya, T. and Isaacson, M. (1981). Mechanics of Wave Forces on Offshore Structures, Van Nostrand Reinhold.
- [11] Garland, M. and Kirk, D. (2010). Understanding Throughput-Oriented Architectures. *Communications of the ACM*, **53**(11), 58–66.
- [12] Muller, D. E. and Preparata, F. P. (1978). Finding the Intersection of Two Convex Polyhedra. *Theoretical Computer Science*, Vol. 7, 2.
- [13] van den Bergen, G. (2001). Proximity queries and penetration depth computation on 3D game objects. In *Proc. Game Developers Conf.*

- [14] Ericson, E. (2005). Real-Time Collision Detection, The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann Publishers.
- [15] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A Fast Procedure for Computing the Distance Between Complex Objects in three Dimentional Space. *IEEE Journal of Robotics and Automation*, **4**(2), 193–203.
- [16] Chung, J. and Hulbert, G. M. (1993). A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-alpha method. *Journal of Applied Mechanics*, **60**, 371–375.
- [17] Gobat, J. I. and Grosenbaugh, M. A. (2001). Applications of the generalized-alpha method to the time integration of the cable dynamics equations. *Computer methods in applied mechanics and engineering*, **190**, 4817–4829.
- [18] Low, Y.M. and Langley, R.S. (2006). Time and frequency domain coupled analysis of deepwater floating production systems. *Applied Ocean Research*, **28**(6), 371–385.
- [19] Chung, Jintai, Cho, Eun-Hyoung, and Choi, Keeyoung (2003). A priori error estimator of the generalized- α method for structural dynamics. *Int. J. Numer. Meth. Engng.*, **57**(4), 537–554.
- [20] Gonthier, Y., Mcphee, J., Lange, C., and Piedbœuf, J-C (2005). A Contact Modeling Method Based on Volumetric Properties. *Proceedings of IDETC'05 2005 ASME Design Engineering Technical Conferences and 5th International Conference on Multibody Systems, Nonlinear Dynamics and Control*.
- [21] Ziegler, J.G. and Nichols, N. B. (1942). Optimum settings for automatic controllers. *Transactions of the ASME*, **66**, 759–768.
- [22] Ogata, Katsuhiko (2002). Modern control engineering, 4 ed. Prentice-Hall, Inc.
- [23] Gonthier, Y., McPhee, J., Lange, C., and Piedboeuf, J.-C. (2004). A Regularized Contact Model with Asymmetric Damping and Dwell-Time Dependent Friction. *Multibody System Dynamics*, **11**, 209–233.
- [24] Gonthier (2007). Contact Dynamics Modelling for Robotic Task Simulation. Ph.D. thesis. University of Waterloo.

Annex A: Sample initialisation files

A.1 Cable class

```
//Verbose output mode on/off
$VERBOSE 1

//////////
//Cable material and geometrical properties

//Axial Stiffness
$EA 5e5
//Bending stiffness
$EI 20
//Torsional stiffness
$GJ 20
//Material density
$rho_cable 7778.0
//Axial damping
$CID 5e3
//Bending damping
$BCID 10
//ratio of compressive stiffness/tensile stiffness
$CE 1.0
//Cable Diameter
$dia_cable 0.02

//////////
// Hydrodynamic properties
//Drag coefficient across cable
$CDc 1.0
//Added mass coefficient across cable
$CAc 1.0

//////////
// Element length control properties for Winch Payout/Payin
$ElementLengthMax 6
$ElementLengthMin 1

//////////
// End-node boundary condition properties
```

```

//end-nodes statically clamped in space, dynamics
//not computed for the end node if turned on.
$NODE0_STATIC 0
$NODEN_STATIC 0

//////////
//Cable node initial conditions.
//Format: vX X vY y vZ Z

//Node 0
$0|IC 0 3.0426 0 8.4 0 6.4
//Node 1
$1|IC 0 3.0426 0 8.4 0 7.9
//Node 2
$2|IC 0 3.0426 0 8.4 0 9.4
//Node 3
$3|IC 0 3.0426 0 8.4 0 10.9
//Node 4
$4|IC 0 3.0426 0 8.4 0 12.4//

//Unstretched element lengths
$ElementLengths 1.525 1.525 1.525 1.527

//////////
// External spherical masses added to cable.  Format:
//[0]: arcdistance to applied mass
//[1]: diameter of extmass
//[2]: density (dry) of extmass
//[3]: drag coefficient (optional, default: 1.0)
//[4]: added mass coefficient (optional, default: 0.5)

//external mass 0
$0|ExtMasses 0.1 0.1 7778 1.0 0.5
//external mass 1
$0|ExtMasses 0.1 0.1 7778 1.0 0.5

//Output folder for cable's results files
//(relative or absolute path).
$OutputFolder LaunchAndRecovery/Results/Cable

```

A.2 Winch class

```
//The winch controller's target velocity for
//velocity control mode
$VelocitySetPoint 0

//The winch controller's target tension for
//tension control mode
$TensionSetPoint 15

//Forces the target velocity when in velocity
//control mode
$KinematicControl 1

//maximum positive acceleration
$accel_max 5
//maximum negative acceleration
$decel_max -25

//Winch controller's PID coefficients
$Kp 0.01
$Kd 0.000001
$Ki 0.00001

//Output folder for Winch's results files
//(relative or absolute path).
$OutputFolder LaunchAndRecovery/Results/Winch
```

A.3 Payload class

```
//////////
// Payload Initial conditions (12 entries)

$0|IC 1 //Vx
$1|IC 0 //Vy
$2|IC 0 //Vz
$3|IC 0 //phiDot (Euler Z-Y'-X'')
$4|IC 0 //thetaDot(Euler Z-Y'-X'')
$5|IC 0 //psiDot (Euler Z-Y'-X'')
$6|IC -1.35 //X
$7|IC 8.4 //Y
```

```

$8|IC 9.6 //Z
$9|IC 0 //phi (Euler Z-Y'-X'')
$10|IC 0 //theta (Euler Z-Y'-X'')
$11|IC 3.14 //psi (Euler Z-Y'-X'')

////////////////////////////////////
// Mass Inertia Matrix about Body Frame
$0|Mass 500 0 0 0 0 0
$1|Mass 0 500 0 0 0 0
$2|Mass 0 0 500 0 0 0
$3|Mass 0 0 0 400 0 0
$4|Mass 0 0 0 0 1541 0
$5|Mass 0 0 0 0 0 1760

////////////////////////////////////
// Friction properties if using contact dynamics

//coulomb friction
$muc 0.05
//sticking friction
$mus 0.05

//Output folder for Payload's results files
//(relative or absolute path).
$OutputFolder LaunchAndRecovery/Results/RescueBoat

```

A.4 Boomcrane class

```

//Verbose output mode on/off
$VERBOSE 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%      Link 1 Red
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

//Description of Link 1's proximal joint frame relative
//to the base frame using Paul's Denevit-Hartenberg
//parameter convention
//Format: alpha a d theta
$0|LinkDHPParameters 90 -0.65 1.4 0

```

```

//Defines whether the link's proximal joint is revolute
//('theta') or prismatic ('d')
$0|LinkJointVariable Theta

//target joint velocity when in velocity control mode.
$0|LinkJointVelocity 0

//The link's rigidbody cg location relative to the proximal
//joint frame
$0|LinkCGLocation -0.35 0.0 0.7

//The link's rigidbody body-fixed frame orientation relative to
//the proximal joint frame
$0|LinkCGOrientation 0 0 0

//The mass of the link's rigidbody in the X Y and Z directions
$0|LinkMass 682.97 682.97 682.97

//The mass moment of inertia of the rigidbody described about the
//body-fixed frame(Ixx, Iyy, Izz)
$0|LinkMomentOfIntertia 139.44 139.44 55.776

//Joint controller's control type (position or velocity)
$0|JointControlType position

//Joint controller's position control PID gains
$0|JointProportionalCoefficientPos 10000
$0|JointIntegralCoefficientPos 100000
$0|JointDerivativeCoefficientPos 0

//Joint controller's velocity control PID gains
$0|JointProportionalCoefficientVel 10000
$0|JointIntegralCoefficientVel 10000
$0|JointDerivativeCoefficientVel 0

//viscous damping coefficient for the joint
$0|JointViscousDamping 1000000

//The maximum allowable power applicable by the joint controller
$0|MaximumActuatablePower 10000000

//Joint limits to prevent the joint from actuating past some

```

```

//threshold
$0|UpstreamJointLimits -1e30 1e30 //no limits

//Joint limit penalty force's stiffness and damping coefficients
$0|JointLimitStiffness 10000
$0|JointLimitDamping 10000

%%%%%%%%%%%%%%
%%      Link 2 Green
%%%%%%%%%%%%%%

$1|LinkDHPParameters 0 1 0 -15
$1|LinkJointVariable Theta

$1|LinkJointVelocity 0

$1|LinkCGLocation 0.5 0 0
$1|LinkCGOrientation 0.0 0 0

$1|LinkMass 754 754 754
$1|LinkMomentOfInertia 31.42 78.5 78.5

$1|JointControlType position

$1|JointProportionalCoefficientPos 1000000
$1|JointIntegralCoefficientPos 300000
$1|JointDerivativeCoefficientPos 0

$1|JointProportionalCoefficientVel 100000
$1|JointIntegralCoefficientVel 1000000
$1|JointDerivativeCoefficientVel 0

$1|JointViscousDamping 300000

$1|MaximumActuatablePower 10000000

$1|UpstreamJointLimits -45 90
$1|JointLimitStiffness 10000000
$1|JointLimitDamping 1000000

//Output folder for Winch's results files
//(relative or absolute path).

```


\$OutputFolder LaunchAndRecovery/Results/Boomcrane

This page intentionally left blank.

Annex B: Descriptions of important class methods

This section is a short, non-comprehensive manual to document some of the important methods that a user might require.

B.1 Payload class methods

B.1.1 Creating a payload object

An `SMS::Payload` object can be created using the class constructor:

```
SMS::Payload(std::string init_file,  
double t0,  
bool file_output)
```

where `init_file` is a string that contains the location of the payload's initialization file, `t0` is the start time of the cable object and `file_output` is set to true to allow simulation data to be output to disk.

B.1.2 Overriding the payload position and velocity

An `SMS::Payload`'s position and velocity can be overridden using:

```
SMS::Payload::SetPosOri(ub::vector<double> &posOri,  
ub::vector<double> &posOriDot)
```

where `posOri` is a size 6 vector containing the Cartesian position followed by the 3 Euler angles that describe its orientation, and `posOriDot` is the time rate of change of `posOri`.

B.1.3 Overriding gravitational constant

An `SMS::Payload`'s gravitational constant can be overridden using:

```
SMS::Payload::SetGrav(double grav)
```

where `grav` is the new gravitational constant.

B.2 Adding a hydrodynamics mesh for non-linear hydrodynamics

A hydrodynamics mesh can be provided to an `SMS::Payload` to allow for non-linear hydrodynamics forces to be calculated using:

```
SMS::Payload::InitialiseHydrodynamicPolyhedronWith3DS(  
std::string file_name,  
double scaleX,  
double scaleY,  
double scaleZ,  
ub::vector<double> &posOri,  
ub::vector<double> &CDc,  
ub::vector<double> &CAc)
```

where `file_name` is the name of the .3ds geometry file, `scaleX`, `scaleY` and `scaleZ` are the scaling factors for the geometry in the \hat{X} , \hat{Y} and \hat{Z} directions respectively, `posOri` is the length 6 position and orientation vector, and `CDc` and `CAc` are length 3 vectors that describe the drag and added mass coefficients in the \hat{X} , \hat{Y} and \hat{Z} directions respectively.

B.2.1 Defining wave conditions of the environment

Waves can be added to the `SMS::Payload`'s description of the environment using ShipMo3D's DeepSeaway library. Currently, only fixed regular seaways are supported, adding it to the simulation can be accomplished using:

```
SMS::Payload::InitialiseEnvironmentAsFixedRegularSeaway(  
std::string label,  
double waterDensity,  
double waveHeadingFromDeg,  
double waveFreq,  
double waveAmp,  
double phaseDeg,  
bool StokesSecondOrder,  
bool WheelerStretching)
```

where `label` is the ocean label for identification purposes, `waterDensity` is the density of the water, `waveHeadingFromDeg` is the heading of the waves in degrees, `waveFreq` is the frequency of oscillation of the wave in radians, `waveAmp` is the amplitude of the wave, `phaseDeg` is the phase of the wave in degrees, and `StokesSecondOrder` and `WheelerStretching` are flags that turn stokes second order waves and Wheeler stretching on and off.

B.2.2 Adding contact dynamics geometries

To enable contact dynamics between an `SMS::Payload` and the environment, geometries must be added to the `SMS::Payload` that describes the payload's contact geometry as well the external object's contact geometry. These can be done respectively using:

```
SMS::Payload::Add3DSContactPolyhedron(  
    std::string file_name,  
    double scaleX,  
    double scaleY,  
    double scaleZ,  
    ub::vector<double> &posOri)  
  
SMS::Payload::AddExternal3DSContactPolyhedron(  
    std::string file_name,  
    double scaleX,  
    double scaleY,  
    double scaleZ,  
    ub::vector<double> &posOri)
```

where `file_name` is the name of the .3ds geometry file, `scaleX`, `scaleY` and `scaleZ` are the scaling factors for the geometry in the \hat{X} , \hat{Y} and \hat{Z} directions respectively, and `posOri` is the length 6 position and orientation vector. The position and orientation of the `SMS::Payload`'s contact objects are described relative to the `SMS::Payload`'s body-fixed frame, while the position and orientation of the external object's contact geometries are described relative to the external object's origin which defaults to the inertial frame's origin.

The geometries added must be convex. Multiple convex objects can be added to the `Payload` to build more complex concave shapes.

B.2.3 Setting the material properties for contact dynamics

The material properties for both the `SMS::Payload`'s and external object's contact objects can be set using:

```
SMS::Payload::SetMaterialProperties(double Young_modulus_set,  
    double B_set)  
  
SMS::Payload::SetExternalObjectMaterialProperties(  
    double Young_modulus_set,  
    double B_set)
```

where `Young_modulus_set` is the Young's modulus and `B_set` is the damping coefficient.

B.2.4 Changing the position and orientation of the external contact objects

The position and orientation of the external contact objects are defined relative to the external object's origin. The origin of the external object can be redefined in the `SMS::Payload` using:

```
SMS::Payload::SetExternalContactPolyhedraPositionOrientation(  
ub::vector<double> &posOri,  
ub::vector<double> &posOriDot,  
ub::vector<double> &posOriDotDot,  
double &currTime)
```

where `posOri` is the length 6 position and orientation vector while `posOriDot` and `posOriDotDot` are the first and second time derivative of `posOri`.

B.2.5 Advancing simulation through time

```
SMS::Payload::AdvanceTime(double DeltaT)
```

B.2.6 Outputting simulation results to disk

```
SMS::Payload::OutputSimObjectData()
```

B.3 Cable class methods

B.3.1 Creating a Cable object

An `SMS::Cable` object can be created using the class constructor:

```
SMS::Cable(std::string init_file,  
double t0,  
bool file_output)
```

where `init_file` is a string that contains the location of the cable's initialization file, `t0` is the start time of the cable object and `file_output` is set to true to allow simulation data to be output to disk.

B.3.2 Defining wave conditions of the environment

Waves can be added to the `Cable`'s description of the environment using ShipMo3D's DeepSeaway library. Currently, only fixed regular seaways are supported, adding it to the simulation can be accomplished using:

```
SMS::Cable::InitialiseEnvironmentAsFixedRegularSeaway(  
std::string label,  
double waterDensity,  
double waveHeadingFromDeg,  
double waveFreq,  
double waveAmp,  
double phaseDeg,  
bool StokesSecondOrder,  
bool WheelerStretching)
```

where `label` is the ocean label for identification purposes, `waterDensity` is the density of the water, `waveHeadingFromDeg` is the heading of the waves in degrees, `waveFreq` is the frequency of oscillation of the wave in radians, `waveAmp` is the amplitude of the wave, `phaseDeg` is the phase of the wave in degrees, and `StokesSecondOrder` and `WheelerStretching` are flags that turn stokes second order waves and Wheeler stretching on and off.

B.3.3 Attaching a Winch to the cable

A `SMS::Winch` object can be added to an `SMS::Cable` to handle payin and payout effects. The winch can be added to either the `SMS::Cable`'s node 0 or node N using:

```
SMS::Cable::AddWinch0(Winch &winch_add)
```

```
SMS::Cable::AddWinchN(Winch &winch_add)
```

respectively, where `winch_add` is a reference to the winch object being attach.

B.3.4 Setting the Cable's end node condition

The condition of the cable at its end nodes can be set using:

```
SMS::Cable::SetNode0BoundaryConditions(  
ub::vector<double> &NodePos,  
ub::vector<double> &NodeVel,  
ub::vector<double> &NodeAccel)
```

```
SMS::Cable::SetNode0BoundaryConditions(
ub::vector<double> &NodePos,
ub::vector<double> &NodeVel,
ub::vector<double> &NodeAccel)
```

for node 0 and N respectively where `NodePos` is the Cartesian position of the node while `NodeVel` and `NodeAccel` are the first and second time derivatives of `NodePos`.

B.3.5 Advancing simulation through time

```
SMS::Cable::AdvanceTime(double DeltaT)
```

B.3.6 Outputting simulation results to disk

```
SMS::Cable::OutputSimObjectData()
```

B.4 Winch class methods

B.4.1 Creating a Winch object

An `SMS::Winch` object can be created using the class constructor:

```
SMS::Winch(std::string init_file,
double t0,
bool file_output)
```

where `init_file` is a string that contains the location of the winch's initialization file, `t0` is the start time of the winch object and `file_output` is set to true to allow simulation data to be output to disk.

B.4.2 Setting the winch's controller mode

An `SMS::Winch`'s controller mode can be set to either velocity or tension control mode using:

```
SMS::Winch::SetMode(std::string velocity_or_tension)
```

where `velocity_or_tension` is a string that can be set to either "velocity" for velocity control or "tension" for tension control.

B.4.3 Setting the velocity control set point

The target velocity for velocity control mode can be set using:


```
SMS::Winch::SetVelocitySetPoint(double setpoint)
```

where `setpoint` is the target velocity.

B.4.4 Setting the tension control set point

The target tension for tension control mode can be set using:

```
SMS::Winch::SetTensionSetPoint(double setpoint)
```

where `setpoint` is the target tension.

B.5 Boomcrane class methods

B.5.1 Creating a Boomcrane object

An `SMS::Boomcrane` object can be created using the class constructor:

```
SMS::Boomcrane(std::string init_file,  
double t0,  
bool file_output)
```

where `init_file` is a string that contains the location of the boomcrane's initialization file, `t0` is the start time of the boomcrane object and `file_output` is set to true to allow simulation data to be output to disk.

B.5.2 Attaching a Cable to one of the Boomcrane's links

An `SMS::Cable` can be attached to a `SMS::Boomcrane`'s link using:

```
SMS::BoomCrane::AddCablePinned(int joint_num,  
SMS::Cable *cable_to_attach,  
ub::vector<double> &location,  
int end)
```

where `joint_num` is the id of proximal joint belonging to the link the cable is being attached to, `cable_to_attach` is a pointer to the cable being attached, `location` is a length 3 vector of the attachment location of the cable with respect to the link's body-fixed frame, and `end` is which end node of the cable is being attached: either 0 for node 0 or 1 for node N.

B.5.3 Adjusting the boomcrane base position and orientation

The position of the `SMS::Boomcrane`'s base frame can be modified using:

```
SMS::BoomCrane::SetBaseGlobalPositionAndOrientation(
ub::vector<double> &posOri,
ub::vector<double> &velocity,
ub::vector<double> &acceleration)
```

where `posOri` is a length 6 array containing the length 3 position of the base, and the length 3 Euler angles, and `posOriDot` and `posOriDotDot` are its first and second time derivatives.

B.5.4 Advancing simulation through time

```
SMS::Cable::AdvanceTime(double DeltaT)
```

B.5.5 Outputting simulation results to disk

```
SMS::Cable::OutputSimObjectData()
```

B.6 Director class methods

B.6.1 Creation a Director object

A `SMS::Director` class object can be created to help control a simulation using:

```
SMS::Director()
```

B.6.2 Adding actors

`SMS::SimObjects` can be assigned to an `SMS::Director` to be controlled as an actor using:

```
SMS::Director::AddActor(SimObject *object,
std::string actorName)
```

where `object` is a pointer to any of the child classes of `SMS::SimObject`, and `actorName` is a label assigned to the actor for future identification.

B.6.3 Adding acting command to a script

A single command can be appended to a script using:

```
SMS::Director::AddCommand(std::string command)
```

where `command` is a string containing the command. A list of available commands can be found in Section 2.2.

B.6.4 Loading a script from file

A script can be loaded from file using:

```
SMS::Director::ReadScriptFromFile(std::string filePath)
```

where `filePath` is the absolute or relative path of the file to be loaded. A list of available commands can be found in Section 2.2.

B.6.5 Managing a scene

Before every `AdvanceTime` call round, the Director must be told to manage the scene. The `SMS::Director` will check that commands are still in progress, or if completed, begin to the next command. This can be done using:

```
SMS::Director::ManageScene(double time,  
double tStep)
```

where `time` is the current time of the scene and `tStep` is the time step for the next `AdvanceTime` call.

This page intentionally left blank.

Annex C: Supplementary plots

C.1 Plots of the simplified Launch and Recovery simulation from Section 4.1

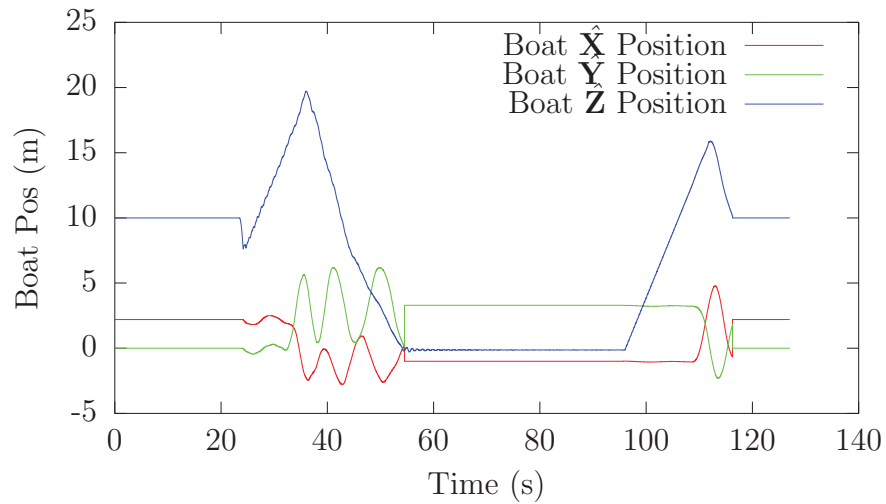


Figure C.1: The simulated position of the rescue boat for the Simplified Launch and Recovery simulation (the \hat{X} position indicated in red, the \hat{Y} position indicated in green and the \hat{Z} position indicated in blue).

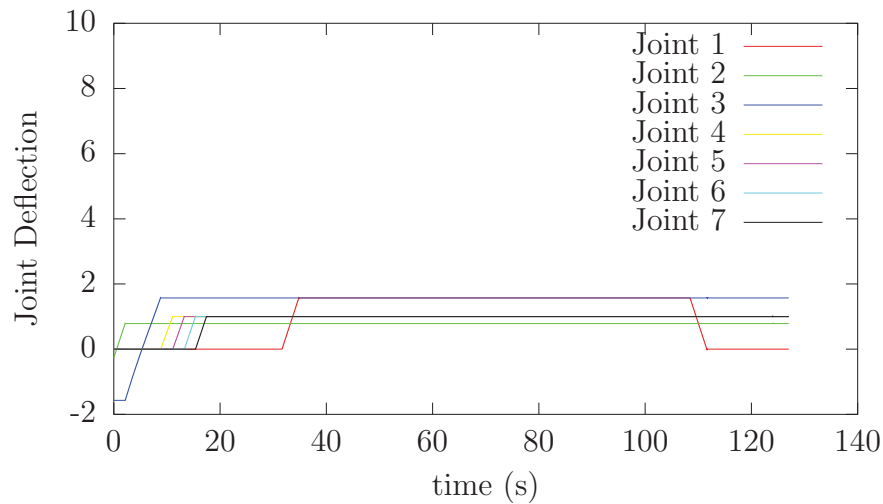


Figure C.2: The simulated joint positions of the boomcrane for the Simplified Launch and Recovery simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).

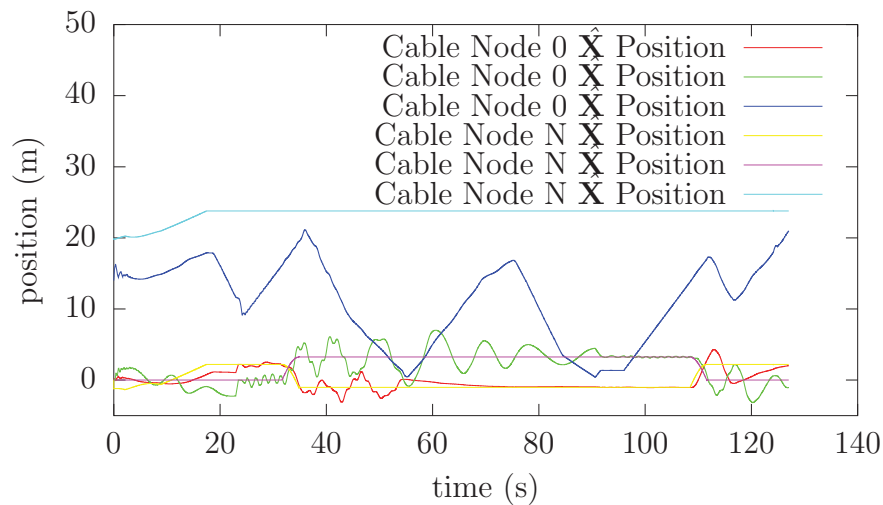


Figure C.3: The simulated position of the cable end nodes for the Simplified Launch and Recovery simulation (the cable's node 0 \hat{X} position is indicated in red, the cable's node 0 \hat{Y} position is indicated in green, the cable's node 0 \hat{Z} position is indicated in blue, the cable's node N \hat{X} position is indicated in yellow, the cable's node N \hat{Y} position is indicated in magenta, the cable's node N \hat{Z} position is indicated in cyan).

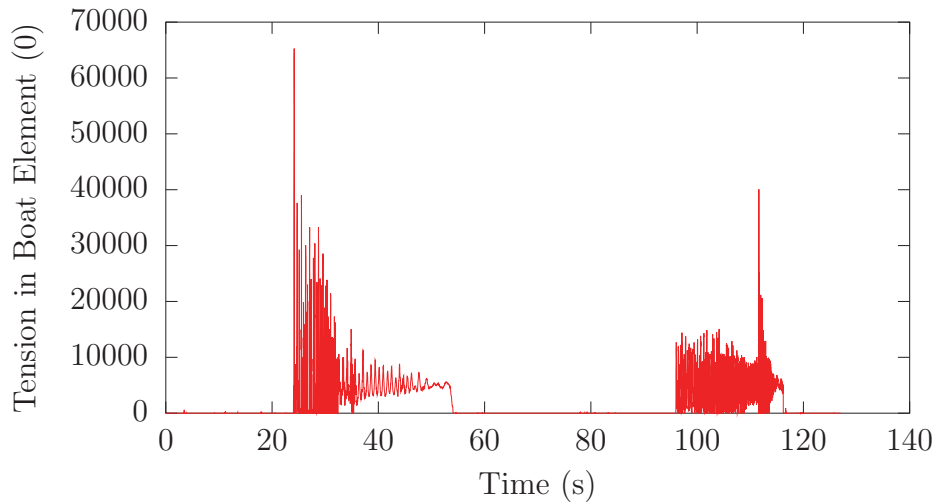


Figure C.4: The simulated tension of the cable's first element for the Simplified Launch and Recovery simulation.

C.2 Plots from the process of improving execution speeds of the Launch and Recovery simulation from Section 4.2.1

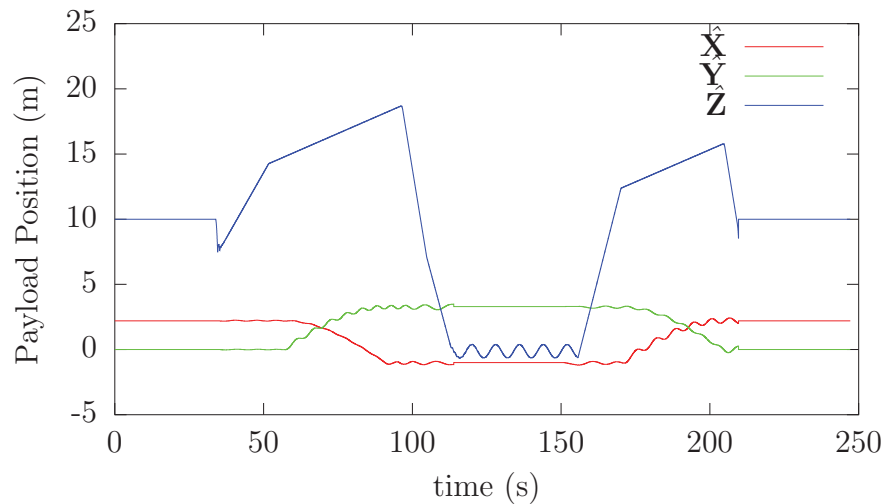


Figure C.5: The position of the rescue boat over the course of the Launch and Recovery simulation (the \hat{X} position indicated in red, the \hat{Y} position indicated in green and the \hat{Z} position indicated in blue).

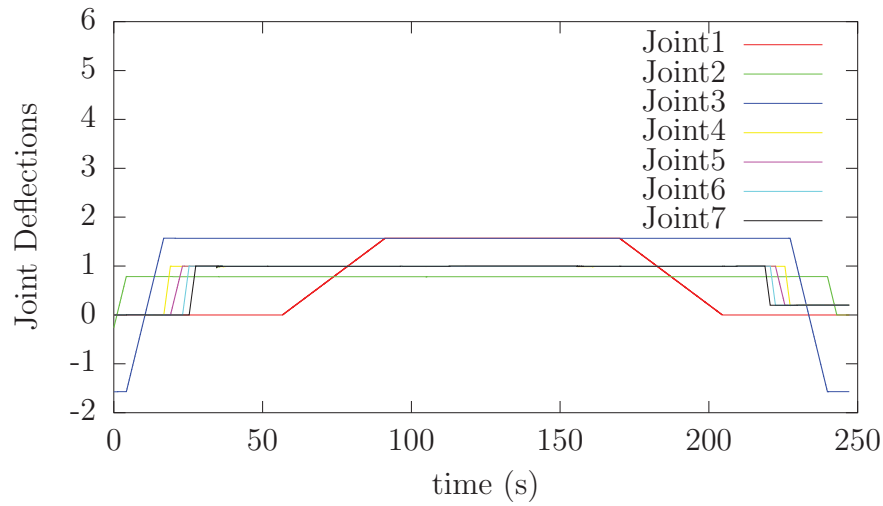


Figure C.6: The position of the Palfinger-like boomcrane joints over the course of the Launch and Recovery simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).

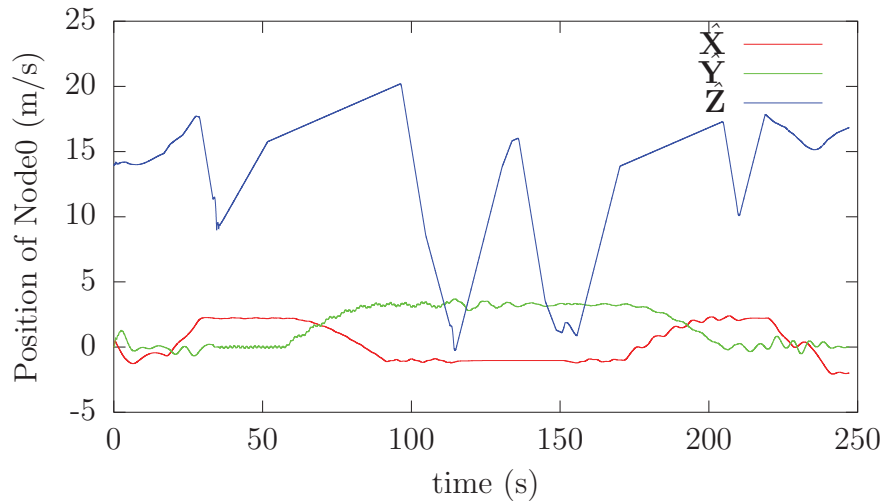


Figure C.7: The position of the cable's end node 0 over the course of the Launch and Recovery simulation (the cable's node 0 \hat{X} position is indicated in red, the cable's node 0 \hat{Y} position is indicated in green, the cable's node 0 \hat{Z} position is indicated in blue, the cable's node N \hat{X} position is indicated in yellow, the cable's node N \hat{Y} position is indicated in magenta, the cable's node N \hat{Z} position is indicated in cyan).

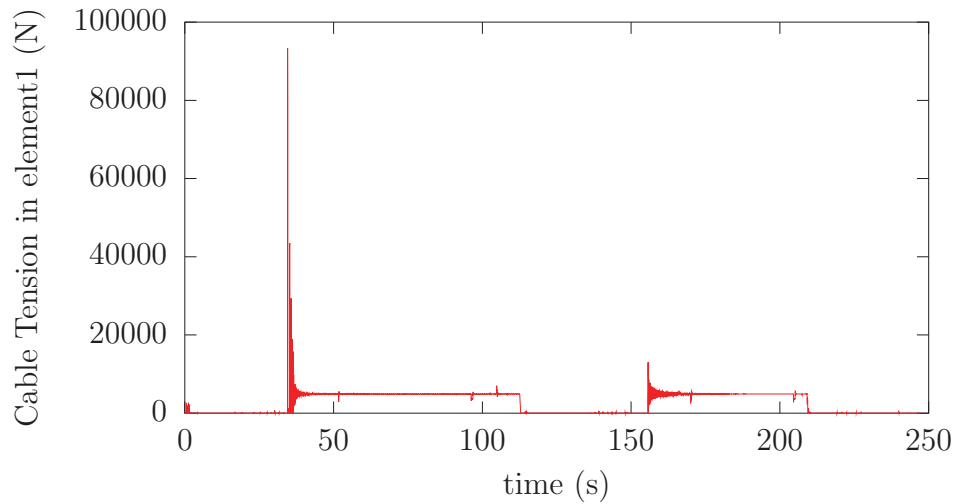


Figure C.8: The tension in the cable’s first element over the course of the Launch and Recovery simulation.

C.3 Plots from the process of Identifying source of small timesteps from Section 4.2.1.2

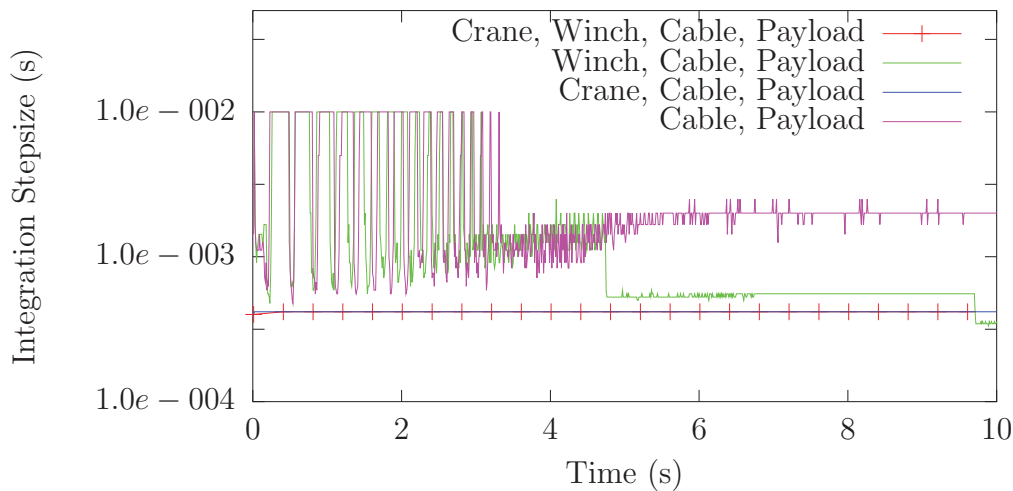


Figure C.9: The integration timestep size over the course of the simulation for the 4 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).

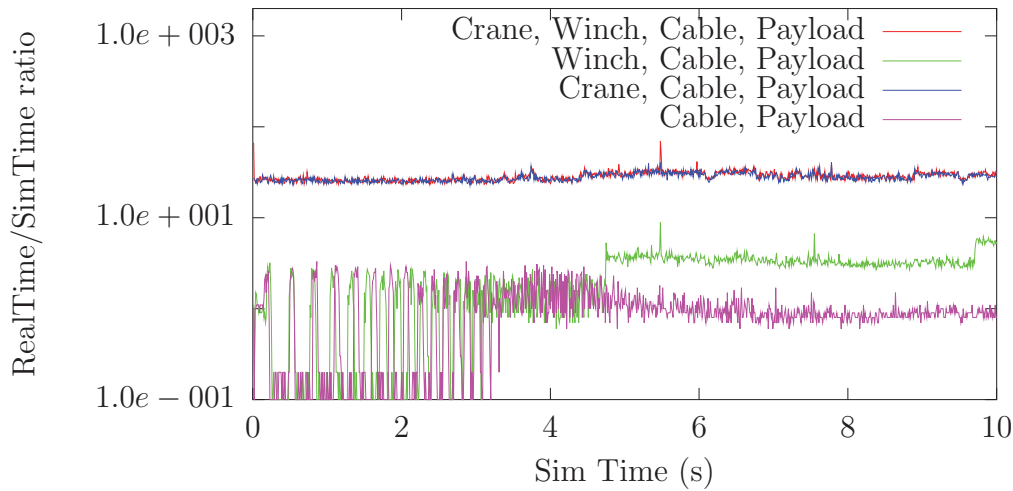


Figure C.10: The integration execution speed time ratio over the course of the simulation for the 4 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).

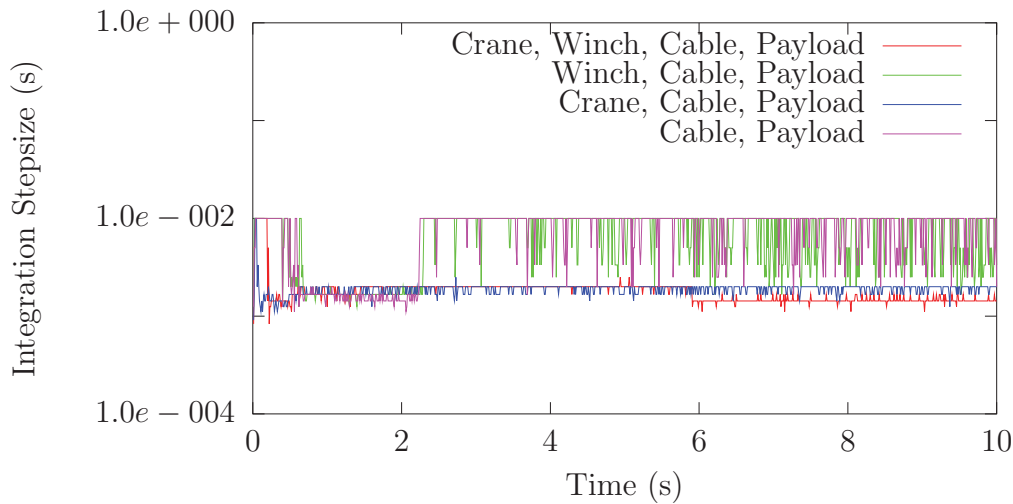


Figure C.11: The integration timestep size over the course of the simulation for the 3 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).

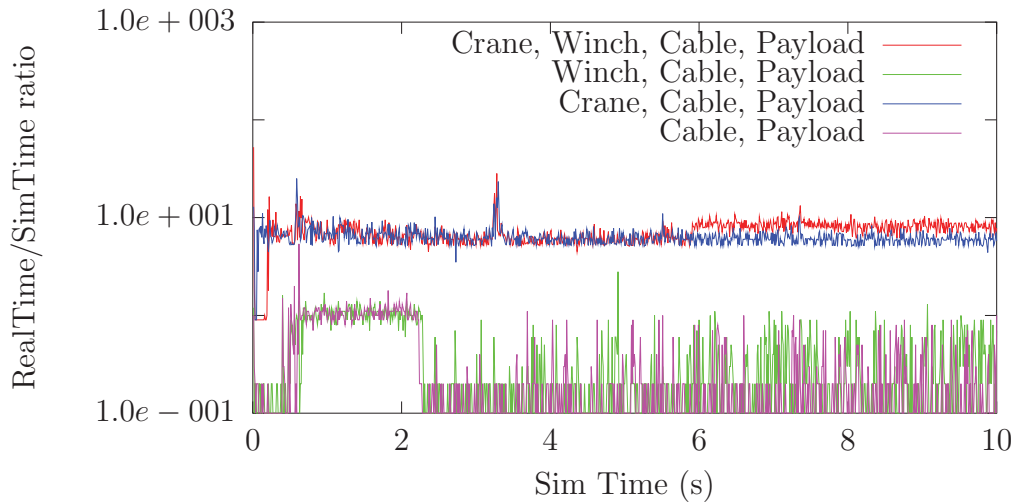


Figure C.12: The time ratio over the course of the simulation for the 3 setup variations (Crane-Winch-Cable-Payload case indicated in red, Winch-Cable-Payload case indicated in green, Crane-Cable-Payload case indicated in blue, Cable-Payload case indicated in magenta).

C.3.1 Plots of the optimised Launch and Recovery simulation of Section 4.2.2

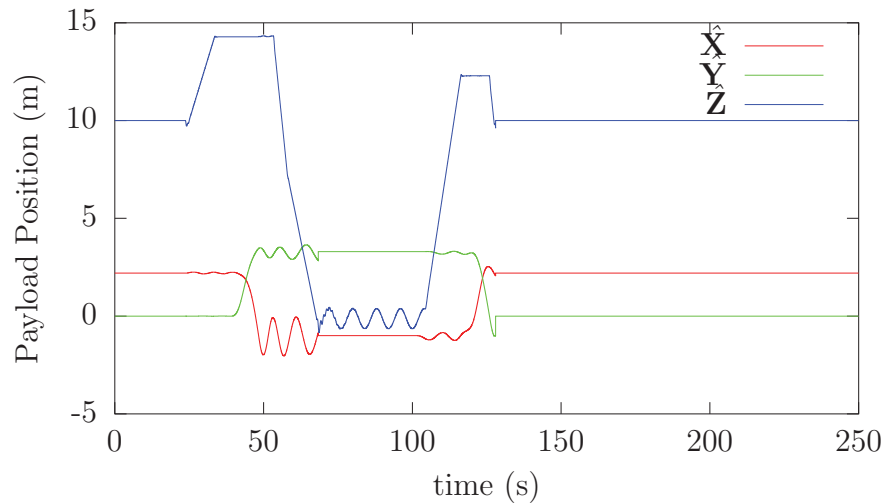


Figure C.13: The position of the rescue boat over the course of the improved launch and recovery example simulation (the \hat{X} position indicated in red, the \hat{Y} position indicated in green and the \hat{Z} position indicated in blue).

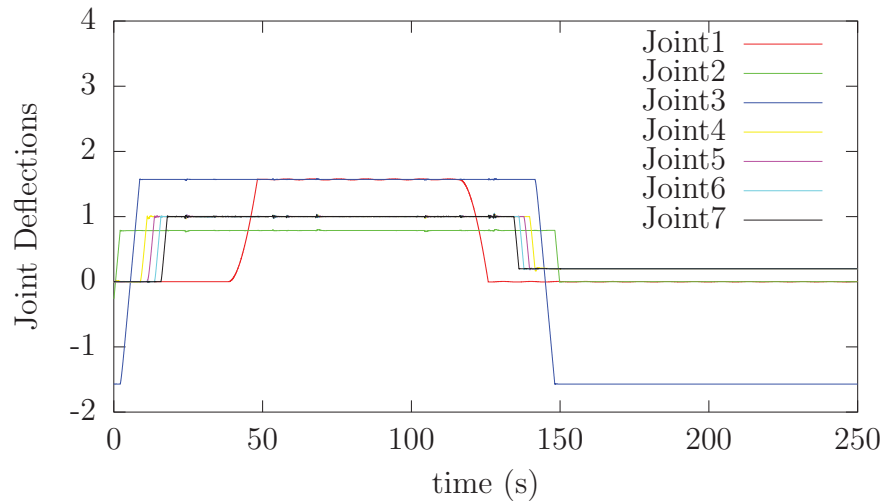


Figure C.14: The position of the boomcrane joints over the course of the improved launch and recovery example simulation (the position of joint 1 is indicated in red, the position of joint 2 is indicated in green, the position of joint 3 is indicated in blue, the position of joint 4 is indicated in yellow, the position of joint 5 is indicated in magenta, the position of joint 6 is indicated in cyan, the position of joint 7 is indicated in black).

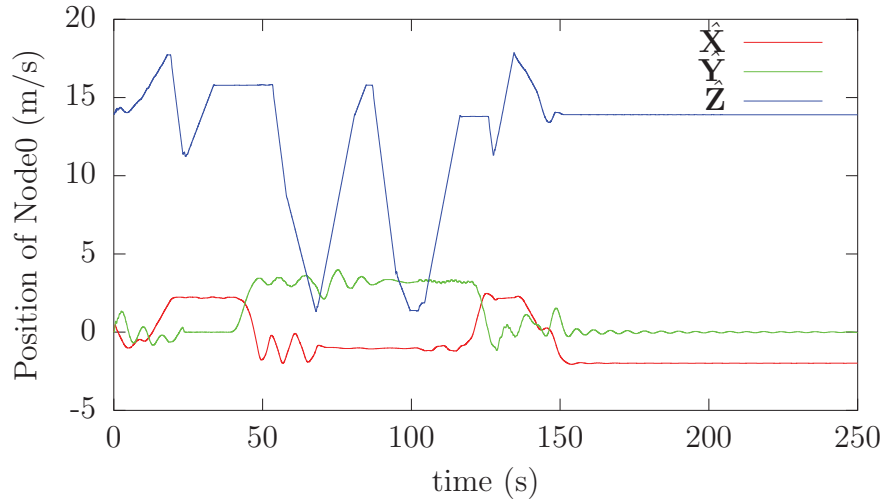


Figure C.15: The position of the cable's end node 0 over the course of the improved launch and recovery example simulation (the cable's node 0 \hat{X} position is indicated in red, the cable's node 0 \hat{Y} position is indicated in green, the cable's node 0 \hat{Z} position is indicated in blue, the cable's node N \hat{X} position is indicated in yellow, the cable's node N \hat{Y} position is indicated in magenta, the cable's node N \hat{Z} position is indicated in cyan).

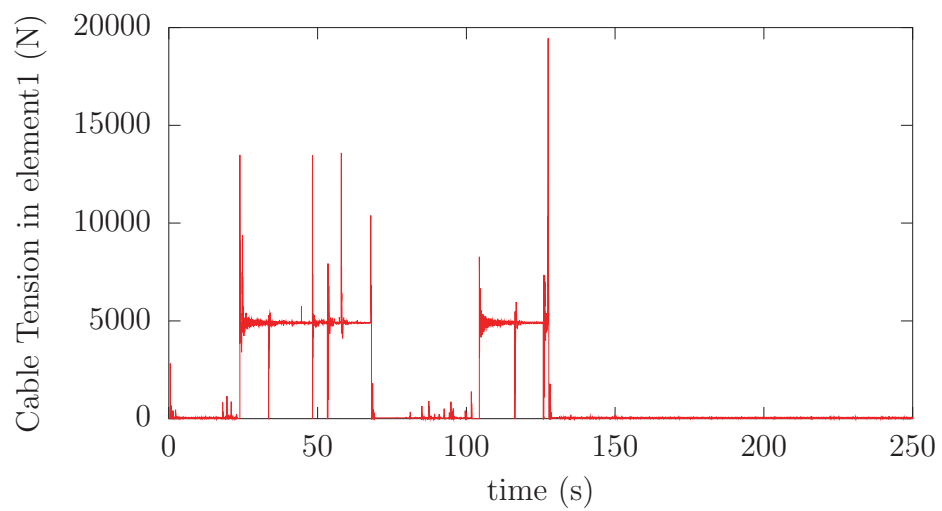


Figure C.16: *The tension in the cable's first element over the course of the improved launch and recovery example simulation.*

This page intentionally left blank.

Distribution List

Improved Simulation of Ship Mechanical Systems

Andre Roy, Dean Steinke and Ryan Nicoll; DRDC Atlantic CR 2012-093; Defence R&D Canada – Atlantic; May 2012.

LIST PART 1: Internal Distribution by Centre

- 1 Author (1 hard copy)
 - 3 DRDC Atlantic Library (1 hard copy)
 - 3 Project Officer ABCA-09-0001 (H/WP) for distribution
 - 4 Project Officer ABCANZ-97-12 (H/WP) for distribution
-
- 11 TOTAL LIST PART 1 (9 CDs, 2 Hard Copies)

LIST PART 2: External Distribution by DRDKIM

- 1 Library and Archives Canada
Attn: Military Archivist, Government Records Branch
- 1 DRDKIM 3
- 1 DMSS 2
555 Blvd. De la Carriere
Gatineau, QC K1A 0K2
- 1 Canadian Forces Maritime Warfare School
Attention: Commanding Officer
PO Box 99000 STN Forces
Halifax, NS B3K 5X5
- 1 Director General
Institute for Ocean Technology
National Research Council of Canada
P.O. Box 12093, Station A
St. John's, NL A1B 3T5

- 1 Transport Canada / Marine Safety
Tower C, Place de Ville
330 Sparks Street, 11th Floor
Ottawa, ON K1A 0N8
- 2 Commanding Officer (2 hard copies)
U.S. Coast Guard Research and Development Center
1082 Shennecosett Road
Groton, CT 06340-2602
- 1 Ministry of Defence
Attn: Head, Naval Research
Dept. of Naval Architecture and Marine Engineering and Development
P.O. Box 20702
2500 ES, The Hague
The Netherlands
- 1 FOI Swedish Defence Research Agency
Attn: Eva Dalberg
Grindsjon Research Centre
SE-147 25 Tumba
Sweden
- 1 Department of Defence (Navy Office)
Attn: Director Naval Ship Design
Campbell Park Office CPI-505
Canberra ACT 2600
Australia
- 1 Document Exchange Centre
Defence Information Services Branch
Department of Defence
Campbell Park Offices CP2-5-08
Canberra ACT 2600 Australia
- 1 Bundesamt für Wehrtechnik und Beschaffung - SG I 3
Postfach 7360
6057 Koblenz
Germany

1 Dr. John Duncan
Head of Simulation Based Acquisition
Defence Equipment and Support
Abbey Wood
Mail Point 8014
BRISTOL BS34 8JH
UK

1 Michael Dervin
PMO Canadian Surface Combatant
DGMPD(L&S)
NDHQ
101 Colonel By Drive
Ottawa, ON K1A 0K2

1 Frank Patsula
PMO AOPS
DGMPD(L&S)
NDHQ – 101 Colonel By Drive
Ottawa, ON K1A 0K2

1 Ken Holt
PMO JSS
DGMPD(L&S)
NDHQ
101 Colonel By Drive
Ottawa, ON K1A 0K2

1 Mr. M. Tunncliffe
DSTM 7
NDHQ
101 Colonel By Dr.
Ottawa, ON K1A 4516

18 TOTAL LIST PART 2 (16 CDs, 2 HARD COPIES)

29 TOTAL COPIES REQUIRED (25 CDs, 4 HARD COPIES)

This page intentionally left blank.

DOCUMENT CONTROL DATA		
(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated.)		
1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Dynamic Systems Analysis Limited 101-19 Dallas Road, Victoria, BC, Canada, V8V 5A6		2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.) UNCLASSIFIED
		2b. CONTROLLED GOODS (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC JUNE 2010
3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.) Improved Simulation of Ship Mechanical Systems		
4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.) Roy, A.; Steinke, D.; Nicoll, R.		
5. DATE OF PUBLICATION (Month and year of publication of document.) May 2012	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 166	6b. NO. OF REFS (Total cited in document.) 24
7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) Contract Report		
8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.) Defence Research and Development Canada – Atlantic PO Box 1012, Dartmouth NS B2Y 3Z7, Canada		
9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 11ge02	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7707-115188/001/HAL	
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Atlantic CR 2012-093	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.) <input checked="" type="checkbox"/> (X) Unlimited distribution <input type="checkbox"/> () Defence departments and defence contractors; further distribution only as approved <input type="checkbox"/> () Defence departments and Canadian defence contractors; further distribution only as approved <input type="checkbox"/> () Government departments and agencies; further distribution only as approved <input type="checkbox"/> () Defence departments; further distribution only as approved <input type="checkbox"/> () Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)		

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Previous work developed simulation components for ship mechanical systems, including cables, winches, cranes, and payloads. The payload modelling includes treatment of collisions, such as those that can occur when cargo being lifted by a crane strikes the side of a ship. The payload modelling also includes treatment of hydrodynamic forces in calm water and in waves. This report describes improvements made to simulation components to achieve increases in both fidelity and computational speed. Verification and validation of simulation components were performed using test cases of varying complexity. A launch and recovery example demonstrates the usage of crane and payload models when deploying a small boat from a large ship in waves. A towing example demonstrates the usage of the cable model when a naval frigate is towing a tuna clipper.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

collision dynamics; cranes; launch and recovery; ship motions; simulation; time domain; towing; waves

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale



www.drdc-rddc.gc.ca