

Worcester Polytechnic Institute

Digital WPI

Doctoral Dissertations (All Dissertations, All Years)

Electronic Theses and Dissertations

2019-12-03

An Asynchronous Simulation Framework for Multi-User Interactive Collaboration: Application to Robot-Assisted Surgery

Adnan Munawar
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/etd-dissertations>

Repository Citation

Munawar, A. (2019). *An Asynchronous Simulation Framework for Multi-User Interactive Collaboration: Application to Robot-Assisted Surgery*. Retrieved from <https://digitalcommons.wpi.edu/etd-dissertations/566>

This dissertation is brought to you for free and open access by Digital WPI. It has been accepted for inclusion in Doctoral Dissertations (All Dissertations, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.

An Asynchronous Simulation Framework for Multi-User Interactive Collaboration: Application to Robot-Assisted Surgery

by

Adnan Munawar

PhD Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Robotics Engineering

by

December 2019

APPROVED:

Professor Gregory S. Fischer, Committee Chair (Mechanical Engineering, WPI)

Professor Loris Fichera (Robotics Engineering, WPI)

Professor Berk Calli (Computer Science, WPI)

Professor Peter Kazanzides (Computer Science, JHU)

Abstract

The field of surgery is continually evolving as there is always room for improvement in the post-operative health of the patient as well as the comfort of the Operating Room (OR) team. While the success of surgery is contingent upon the skills of the surgeon and the OR team, the use of specialized robots has shown to improve surgery-related outcomes in some cases. These outcomes are currently measured using a wide variety of metrics that include patient pain and recovery, surgeons comfort, duration of the operation and the cost of the procedure. There is a need for additional research to better understand the optimal criteria for benchmarking surgical performance. Presently, surgeons are trained to perform robot-assisted surgeries using interactive simulators. However, in the absence of well-defined performance standards, these simulators focus primarily on the simulation of the operative scene and not the complexities associated with multiple inputs to a real-world surgical procedure. Because interactive simulators are typically designed for specific robots that perform a small number of tasks controlled by a single user, they are inflexible in terms of their portability to different robots and the inclusion of multiple operators (e.g., nurses, medical assistants). Additionally, while most simulators provide high-quality visuals, simplification techniques are often employed to avoid stability issues for physics computation, contact dynamics and multi-manual interaction. This study addresses the limitations of existing simulators by outlining various specifications required to develop techniques that mimic real-world interactions and collaboration. Moreover, this study focuses on the inclusion of distributed control, shared task allocation and assistive feedback – through machine learning, secondary and tertiary operators – alongside the primary human operator.

Acknowledgements

I would like to acknowledge several people that have inspired, motivated and helped me during my research. First, I would like to acknowledge my parents and siblings for their unwavering support in all my endeavors. I would like to acknowledge my advisors and peers from my time back in the School of Science and Engineering (SSE); Dr. S. Masud and Dr. A. Muhammad, Dr. S. Athar, Dr. T. Manzoor, Dr. H. Khan, Dr. Z. Ahmad and Dr. H. Nisar.

I would like to thank my friends and previous colleagues in North Carolina; Alex Maret, Stefan Atay, Dustin Vaughan, Andrew McDaniel and many others for their support during my internships and teaching me invaluable lessons that I have applied in my research.

The number of people that I would like to thank at WPI and AIM lab are just too many so I will name the ones I have most recently interacted with. My colleagues and peers; Dr. Patel, Radian, Yan, Nishan, Anna, Farid, Druv, Vignesh, Nuttaworn, Vishnu, Ehtisham, Ali Hussain, Ali Shah, Ayaz, Arsalan, Hammad, Saad, Ahmad, Sami, Fatima, Najma, Sumeet, Satish, Kene, Tanmay, Katie, Paulo, Tess, Dr. Nycz, Anastasia and many others that have made my time at WPI very memorable.

I would like especially to acknowledge Anton and Dr. Z. Chen from JHU for inspiring me to develop AMBF.

I feel lucky to have been part of the 2012 Fulbright cohort. My peers from this cohort; Mariam, Safyah, Dr. A. Usman, Fazli, Hira, Asma, Kiyas, Hera and many others that have made my time in graduate school very enjoyable.

My committee members, Dr. P. Kazantzides, Dr. B. Calli and Dr. L. Fichera have been phenomenal advisors and have helped me tremendously in achieving this milestone.

Most important of all, I would like to thank Dr. G.S. Fischer for his guidance and

support throughout my M.S. and Ph.D. and allowing me to conduct this research.
I could not have asked for a better advisor or a better research lab.

Contents

1	Introduction	1
1.1	The da Vinci Research Kit	8
1.2	Background	9
1.3	Motivation	18
1.4	Contributions	26
1.4.1	Conceptual Contributions	26
1.4.2	Implementation Based Contributions	27
1.4.3	Community Impact	28
1.5	Organization	29
1.6	Acknowledgement	31
2	Review of Solvers for Dynamic Simulations	33
2.1	Introduction	33
2.2	Mathematical Formulation of Dynamic Simulations	34
2.3	Force Based Methods	35
2.4	Velocity Based Methods	37
2.5	Position Based Methods	40
2.6	Indirect Methods	42
2.6.1	The Combined Jacobian Method	42

2.6.2	The Collision K Matrix Method	43
2.7	Solving the Constraints	44
2.8	Articulated Body Methods	45
2.9	Discussion	47
3	Asynchronous Framework for Collaborative Interaction	49
3.1	Published Work	50
3.2	Introduction	50
3.3	Selection of Software Components	53
3.4	Implementation Details	55
3.4.1	Implementation of Real-Time Dynamic Simulation	55
3.4.2	Asynchronous Control of Multiple IIDs	59
3.4.3	Contextual Viewport Control	63
3.5	Minimal Frame Representation for Input Mapping	64
3.6	Plugin Based Interface for dVRK Masters	78
3.7	Medium for Communication Pipeline	80
3.7.1	Bidirectional Communication Interfaces	84
3.7.2	Communication Pipeline Payloads	84
3.7.3	Normalized Joint Control of Multi-Jointed SDEs	87
3.8	The Python Client	89
3.9	Results and Discussions	93
3.10	Conclusion	103
4	Distributed Format for Robots, Environments and Devices	105
4.1	Published Work	106
4.2	Introduction	106
4.3	The AMBF Description Format (ADF)	109

4.3.1	Anatomy of ADF	110
4.3.2	Interconnected Bodies	112
4.3.3	Convention of Constraint Definition	113
4.3.4	Flexibility of Name-spacing and Resource Paths	116
4.3.5	Resolving Naming Conflicts	117
4.3.6	Support for Soft Bodies	120
4.3.7	Action Based Sensors for Reusability	120
4.3.8	Auto Generation of Communication Instances	122
4.4	Compatibility of ADF with External Software	123
4.4.1	URDF to ADF Conversion	123
4.4.2	Blender ADF Addon	124
4.4.3	Implementation of Multiple View-ports using Camera Data . .	130
4.5	Results and Discussion	132
5	Integration of Soft-Body Simulations	136
5.1	Acknowledgement	137
5.2	Introduction	137
5.3	Problem Formulation	137
5.4	Related Work	139
5.5	Methods	140
5.5.1	Real-Time Simulation of Soft-Body Dynamics	141
5.5.2	Representation of a Soft-Body	142
5.5.3	Visualization	144
5.5.4	Manipulation of Soft-Body	148
5.6	Results	151
5.7	Discussion	153

6	Grasping in Simulation	156
6.1	Acknowledgement	156
6.2	Introduction	157
6.3	Related Work	159
6.4	Problem Formulation	162
6.4.1	Limitations Associated with Rigid-Body Collisions	162
6.4.2	Limitations Associated with Geometric Representation of Rigid Bodies	163
6.4.3	Dynamics Calculation in Physics Libraries	164
6.5	Methods	166
6.5.1	Resistive Sensors for Preemptive Contact Computation	166
6.5.2	Anatomy of a Resistive Sensor	167
6.5.3	Visualization of Contact Forces	171
6.5.4	Automating Sensor Placement	173
6.6	Results	176
6.7	Discussion	179
7	Applications and Use-Cases	182
7.1	Supervised Semi-Autonomous Control with Bayesian Optimization . .	183
7.1.1	Design of Training Puzzle	185
7.1.2	Study Results	186
7.2	Analysis of Collaborative Control for Surgical Training Tasks	189
7.2.1	Setting up the Study	191
7.2.2	Study Protocol	196
7.2.3	Study Results	197
7.3	Conclusion	199

8	Conclusion and Future Work	200
8.1	Review of Contributions	200
8.2	Lessons Learned	201
8.3	Future Work	203
8.3.1	Object Specific Communication Payloads	203
8.3.2	Extension of the Python Client	204
8.3.3	Inclusion of More Sensors	204
8.3.4	Support for Application Program Interface	205
8.3.5	Improved Softbody Simulation Framework	205
8.3.6	Swappable Middleware	205
8.3.7	Incorporating CRTK Specification	206
8.4	Conclusion	206

List of Figures

1.1	Traditional setup for Laparoscopy [1].	2
1.2	The first robot (PUMA-200) to be used for brain biopsies [2].	2
1.3	A robotic system for prostate resection via the trans-urethral route. . .	3
1.4	The commercial Robodoc intended for hip replacement surgeries . . .	4
1.5	(a) The ZEUS Slave Console.(b) The ZEUS system with Master and Slave Console.(c) The automated endoscope called (AESOP)	5
1.6	The da Vinci Surgical System with the surgeon operating the MTM console and the helping nurse at the PSM station	7
1.7	The da Vinci Research Kit at WPI Aimlab.	8
1.8	A lab setup for the Raven II surgical robot.	10
1.9	In 2019 the dVRK is present in 35 sites worldwide.	11
1.10	In 2019 the Raven II is present in around 20 sites worldwide.	11
1.11	The open-source hardware controllers for interfacing with each indi- vidual dVRK manipulator.	12
1.12	The peg transfer puzzle for surgical training.	17
2.1	The simplest form of constraint based on collision between two rigid bodies.	38
3.1	A conceptual view of the Asynchronous Framework	52

3.2	The external components that have been selected to complement various part of the Asynchronous Framework and the AMBF.	54
3.3	(a) Time dilation between Application Clock & Simulation Clock using fixed time-step ($dt=0.001$) (b) Time tracking between Application Clock & Simulation Clock using dynamic time-step	59
3.4	Input devices to interact with a dynamic simulation.	60
3.5	A block diagram depicting the Design of Asynchronous Control Scheme, the Simulated end-effectors and Devices maintain independent and mutually exclusive Data Structures (DS) that are updated on successive writes and are capable of asynchronous reads	62
3.6	These figures show the simulated end-effectors controlled by dVRK Master with clutch/camera foot-pedals enabled. The clutch is used to move the haptic device disengaged, and the camera foot-pedal is used to re-orient the view-direction without affecting the end-effector.	71

3.7	This flow chart represents the internal process of binding an IID to an SDE and a Camera. These parameters are specified using the front end format shown in Figure 3.8. The user has to specify at least one of the two fields “simulated multi-body” or the “root link”. If both the “simulated multi-body” and “root link” are defined, the root link is searched for in the simulated multi-body file. If a “root link” is not set, then the body with the least number of parents in the “simulated multi-body” is treated as the root link. Lastly, in case, the “simulated multi-body” is not defined, it is expected that the “root link” refers to a body already present in the simulation. The field “cameras” is optional and is used to define the controllable cameras from the corresponding IID. If the “cameras” field is not defined, then all the existing cameras in the simulation are added to the device’s cameras. In any case, the first camera in the IIDs list of cameras is used as the device’s FoR.	72
3.8	The specification of an IID and it’s simulation parameters using the front-end specification format. The important parameters for this discussion are the three fields namely “simulated multi-body”, “root link” and the list of “cameras”. The “simulated multi-body” is a description file that defines a proxy simulated multi-body that will be controlled by this IID. The “root link” refers to a body in the “simulated multi-body” or an existing body in the simulation that will be bound to the IID. The “cameras” field is a list of cameras controllable from this IID. The combined use of these three fields is discussed in Flowchart 3.7	73

3.9	A zoomed out view of the components involved in a unilateral or multi-lateral control. The inclusion of the user frame U is important as the user has to deal with the difference between the device base frame and the simulation world's (or camera's) frame as a reference. .	74
3.10	A visual illustration of the SDE frames. These frames are defined for each IID-SDE pair, in-case of multi-lateral control, each pair has its own set of variables. The frames F_{SDE-TO} and F_{SDE-BO} are usually defined at initialization and remain fixed throughout the simulation, while $F_{SDE-REF}$ and $F_{SDE-REF-O}$ change based on the clutching of device position control button.	75
3.11	The frames involved in a generalized representation of an IID. The frames F_{IID-BO} , F_{IID-TO} , F_{IID-B} and F_{IID-E} are meant to handled in the corresponding device drivers (dVRK Arm Plugin in case of the dVRKs) while the frames F_{IID-CL} , $F_{IID-PRE-CL}$ and F_{IID} are handled in the Asynchronous Framework.	76
3.12	The frames associated with the camera which are defined for all IID-SDE-Cam triplets. Each triplet unit has its own set of these frames. .	76
3.13	A flowchart depicting the process of controlling an IID in uni-lateral or multi-lateral control in single device thread. Each IID has its own thread and this flow chart repeats asynchronously.	77
3.14	This block diagram depicts a plugin based interface for dVRK manipulators using ROS as an IPC. The ROS functionality is sealed in the Arm Bridge Class whereas the ARM Interface exposes API for user applications.	79

3.15	A visual representation of the Asynchronous Framework with regards to the C++ AMBF Simulator where each simulated dynamic object is represented as an <i>afObject</i> . The <i>afObjects</i> utilize independent communication pipelines by exposing State/Command interfaces which allow isolated control	81
3.16	Generating grippers such that the joint axes between the left and right fingers (and sub-links) are inverted. This allows a scalar variable to map to multiple joints and allows a generic interface with IIDs having only one pinch DOF.	88
3.17	The Python Client communicates with the AMBF Simulator using ROS as a middle-ware, AMBF ENV retrieves the requested handles for objects from Python Client and provides a GYM compatible interface	91
3.18	The Flowchart depicting the process of throttling the dynamic simulation based on setting the “Enable Throttle” flag by an external application. Once the flag is set, the external application is responsible for providing a clock as shown by the field “External Clock”. The default value of “No. Skip Steps” is set to 5, which the number of simulation steps the physics will take between each clock toggle. This field can also be set dynamically.	92
3.19	Figure (a) and (b) show the haptic update-rate of 5 devices when controlled ‘sequentially’ vs ‘asynchronously’, respectively. Figure (c) and (d) show the corresponding rates for physics update-loops for ‘sequential’ vs ‘asynchronous’ control	94
3.20	Reponse of haptic controllers with degrading dynamic loops frequency	96

3.21	These sub-figures show the progression (top to bottom) of a bi-manual task using the AMBF Simulator. The two SDEs holding the green multi-link puzzle piece are controlled by dVRK Masters (shown as Picture in Picture on top right) and the other two SDEs are controlled via Razer Hydra (shown as Picture in Picture on top left)	97
3.22	Bi-Lateral SISO Control by using a pair to Novint Falcons and a pair of dVRK MTMs to control the same SDEs.	100
3.23	Communication speed of several <i>afObjects</i> for an overloaded dynamic environment. The desired communication frequency is set to 2 kHz Dynamic-Loop's Frequency $\sim 300Hz$, <i>afObjComm</i> frequency $\sim 2kHz$	101
3.24	(a) The histogram showing the communication latency between the C++ AF and the Python Client using message queue size of ~ 10 (b) The green dots show the difference between every successive new message received from the C++ AF by subtracting from the previous packet's embedded time. Similarly, the red dots show the difference between the current time when the message was read from the previous time the last packet was read.	102
3.25	The histogram showing the communication latency between the C++ AF and the Python Client using message queue size of ~ 10	103
3.26	Histogram of the time difference between the embedded time of a received packet and the current time for synchronous communication using Step Throttling	104

4.1	The anatomy of ADF. The yellow tile forms the header and consists of global parameters and header lists which are highlighted with the purple dotted border. The red tile represents a constraint, green represents bodies and blue represents scene objects. The blue text highlights optional parameters.	111
4.2	Densely connected bodies with the corresponding lineage for each body shown on the right.	112
4.3	A subset of robot models already implemented for the AMBF simulator in Blender. These robots include the da Vinci Surgical Robot with multiple parallel mechanisms.	124
4.4	A few features of the Blender-to-AMBF add-on include copy pasting robot models, scaling, altering the pose of any subset of robots/links, visually setting constraints and inertial properties, creating collision meshes and generating/loading created ADF files.	125
4.5	In the sub-figures, the purple and turquoise bodies represent the parent and child with the constraint axes marked with the black ring. In (b), the child body is rotated to form a constraint by aligning $a\vec{x}_p$ and $a\vec{x}_c$. (c) shows the adjustment required in Blender such that the child body is rotated to adjust the constraint axis to default \vec{n}_z followed by (d) to align the constraint axes with parent's axis.	127
4.6	A visual representation of plane offset between the plane formed by shortest angle rotation between parent's and child's constraint axes (purple disk) and the rotation plane of correction axis (green disk). .	128
4.7	A multi-port view of the underlying simulation using 3 frame-buffers which output to separate windows and can be dragged around different monitors.	131

4.8	A simulation with several manipulators running in real-time. The labeled manipulators (ECM and PSM) have two connected closed-loop mechanisms while the MTM has one closed-loop mechanism. . .	131
4.9	Each column shows the joint control of a different manipulator labeled underneath. The last row shows the dynamic update frequency of physics simulation. The ECM's and PSM's 3rd graph depicts a translational joint while all the joints of the MTM are rotational. . .	132
4.10	The loading vs unloading times for simulators with increasing number of complex robot models. The simulators are loaded using the bash terminal.	133
4.11	WPI's Neuro Surgery Robot Model using the Blender-to-AMBF add-on. The robot consists of a 6 bar linkage at the base and an interconnected 8 bar linkage at the top. The robot is controlled using ROS topics at 1 kHz communication frequency.	134
4.12	The dynamic selection and removal of links belonging to interconnected mechanism.	135
5.1	Anatomy of the ADF. The blue tile forms the header and consists of global parameters and header lists which are highlighted with the purple dotted border. The red tile represents a constraint, green represents rigid bodies and yellow represents soft-bodies. The tunable parameters for soft-body dynamics can be set using the config parameter highlighted in red. The defined parameters include $kLST$ = Linear Stiffness Coefficient, kDP = Node Damping Coefficient, kPR = Internal Pressure Coefficient.	141
5.2	Two meshes with similar surface geometry but different internal structure defined using the OBJ mesh format.	143

5.3	Reference image for Algorithm 7. The soft-body fits in the boundary box that is sub-divided into p, q and r blocks along x,y and z axes respectively. Each block is then parsed individually by creating 5 sub-blocks which are a CHK, an IDX and 3 Vertex Triplet sub-blocks.	146
5.4	(a) The original vertex indices that do not account for repeated vertices. (b) The reduced vertex list with the duplicate vertices unified together into a new list.	146
5.5	Proximity sensors can be defined using the same method as bodies, joints, and scene objects in Figure 5.1. The proximity sensor is parented to the desired body with the relative location offset, direction, and range.	149
5.6	Similar to convex hulls used for rigid-body dynamics, a complex soft-body shape can be generated using a compound of simpler shapes. These simpler shapes can be used to perform Boolean operations of mesh subtraction or addition as shown from (a) \rightarrow (b). Finally, (c) shows the simulation and interaction of this mesh in AMBF.	152
5.7	Sequential process of converting a cylindrical primitive to a mesh with coarse surface, creating edges inside for structural stability, and finally applying texture for visual realism.	153
5.8	Examples of Soft-body manipulation using the dVRK MTMs	154
5.9	Real Time Factor for tasks shown in Figure 5.8 (a), (b).	155
6.1	Natural method of grasping for fully-static or quasi-static dynamics. The skin surface in the vicinity of contact points deforms according to the underlying shape of object, thereby providing better surface friction.	158

6.2	Natural Manipulation using either controlled slip, controlled slide or both. The controlled slip and slide is usually assisted by either the weight of the grasped body, using a second hand or leveraging the collision with other objects in the environment.	159
6.3	The penetration depth D_p of three collision shapes (Spheres) with mass = 50 Kg, radius 0.5 m, non static ground plane position at 0 m and drop height = 2 m. The penetration depth was recorded as the difference between the maximum fall distance and the resting position after stabilization.	163
6.4	Visualization of the friction cone to model the natural friction response. The coefficient \vec{n}_W is the contact normal in the world, \vec{F}_R is the resultant force which is expressed as $(\vec{F}_R = \mu * \vec{F}_N)$, \vec{F}_{IMP} is the impending friction and ϕ_s is the static friction ratio.	165
6.5	(a) A visual representation of an individual Resistive Sensor during contact with an external body B (the blue torus). The coefficients are defined in Table. 6.1. (b) Visualization of Resistive sensors mounted on a body (blue cube) before and after penetration into another body (green sphere).	168
6.6	Grasping an object using a simple two finger gripper. The normal force at the contact points is required to compute the normal force \vec{F}_N for static friction computation.	169

6.7	Static friction response of body-mounted with Resistive sensors (transparent blue box) sliding along the plane underneath (shown with the wooden texture) subject to a constant force applied along the direction of the ground plane. The response is calculated as the tangential error (i.e. the difference between the commanded and current position of the blue box).	172
6.8	(a) Primitive Patches for Resistive Sensor Placement. (b) The primitive shapes can also be "skin-wrapped" to match the contours of the underlying complex shape. Figure (c) Sensor placement on the simulated gripper with red spheres representing the $P_{rayStart}$ and the green spheres representing P_{rayEnd} .	174
6.9	A simple prismatic gripper with two links, mounted with an array of Resistive sensors at ends facing each other. The bottom two figures show the grasping of a magenta cylindrical object with an asymmetric posture and the grasping of a yellow object that is composed of multiple collision meshes.	176
6.10	Stability analysis on an inclined plane. $m = 0.5Kg$, $K_s = 5000$, $\sigma_a = 0.001$, $K_n = 1$, $K_D = 50$ and $\mu_v = 0.1$.	177
6.11	Bi-manual manipulation of a screwdriver to rotate the cast assembly underneath	177
6.12	Manipulation of a deformable thread around the gripper jaws and around a puzzle.	179
6.13	The dynamic update frequency of real-time simulation for the two-handed screw driver task and the real-time factor.	180

7.1	(a) The study setup involves the human-subjects looking at the screen where the interactable simulation is being displayed. (b) The goal of the exercise is to pick the peg located at A using the right SDE, handing it over to the left SDE and placing it at B. Then picking back the peg at B and placing it at C using the left SDE. Finally, switching hands to use the right SDE to pick and place the peg back at A. [3].	184
7.2	The supervised semi-autonomous control scheme for assisting the human subjects in surgical task performance [3].	185
7.3	The box plots show the results of semi-autonomous control assistance with and without optimization through Bayesian learning. [3]. . . .	187
7.4	The training environment for getting the study subjects on a similar footing for the user-study. The goal of the environment is to pick and place the puzzle pieces (red, green, blue and yellow bodies) in the purple cast with matching intrusions. The subjects learned the use of clutching control and grasping to perform the task.	188
7.5	(a) Similar to the first user-study, the test subject is required to pick the pegs labeled 1 and 6 using the right SDE, then switching mid-air to the left SDE and placing the blocks in the opposite corners. The pegs can be transferred in any order. (b) This was a more involved task requiring simultaneous control input from both SDEs. The goal was to pick the yellow puzzle from the two handles (dark gray in color) and place it on the base with matching extrusions.	190

7.6	The MTM's wrist platform link is actuated to provide null space control by affixing a virtual plane (translucent green plane) to the platform link. Secondly, a virtual unit vector (red arrow) is affixed to the tip roll link. The angle between the red arrow and the green plane is used in a PD control law to rotate the wrist platform link. . .	193
7.7	The GUI for recording the user-study data.	194
7.8	Task completion time between the two studies A and B and three control modes each.	196
7.9	A example of a human subject performing the collaborative user-study.	196
7.10	Length of the path traversed by IIDs between the two studies A and B and three control modes each.	197
7.11	Clutching frequency between the two studies A and B and three control modes each.	198
7.12	Average of Reduced NASA TLX questionnaire between the two studies A and B and three control modes each	199

List of Tables

3.1	The Description of Transformation matrices used for the XVII Representation	67
3.3	The Description of Frames used for Minimal Frame Representation. These Frames are shown in Figures 3.9, 3.11, 3.10, 3.12	68
3.5	Equations for Multi-Lateral Control of Camera/SDEs with IIDs using XVII Representation. These equations are indexed according to the Flowchart 3.13	70
3.6	<i>afWorlds</i> State Payload	84
3.7	<i>afWorlds</i> Command Payload	85
3.8	<i>afObjects</i> State Payload	85
3.9	<i>afObjects</i> Command Payload	86
4.1	Basic Comparison Between URDF and SDF	109
4.2	Simplifying redundant names using name-spaces rather than suffixes .	117
5.1	Population of Vertex Triplets for the example in Figure 5.4	148
6.1	Symbols used in this Manuscript.	166
6.2	Parametric Data for Specified Tasks	178
7.1	Comparison between the task performance of manual and semi-autonomous supervisory control	187

7.2	Comparison between the task performance of supervised semi-autonomous control with and without Bayesian optimization.	188
-----	---	-----

Chapter 1

Introduction

The use of robots for surgical procedures has increased over the years [4] as they offer enhanced precision, better vision and comfortable control for surgeons. As far as the patients are concerned, depending on the type of surgery, they minimize the procedure area, which prevents unnecessary tissue damage and thus shortens hospital stays, reduces drug dosage and thereby improves recovery times. These surgeries are called robot-assisted minimally invasive surgeries (RMIS) or laparoscopic surgeries. Not all laparoscopic surgeries are performed by robots though, in fact, according to some estimates, robotic surgeries, in general, account for less than 3 % [5] of the total surgeries performed each year in the US. Robot-assisted laparoscopies make up an even lower percentage. Certainly, there is room for improvement.

In traditional MIS surgeries, long and slender shaped tools are pivoted at the port of entry and inserted into the body as shown in Figure 1.1. The tools have various types of end-effectors that include imaging tools, cutting, pinching, grasping and cautery tools. These are the basic categories and within these categories there exist many different types of sub-tools. Depending on the type of surgical procedure, different sets of tools are used, however, all the different tools are inserted using small

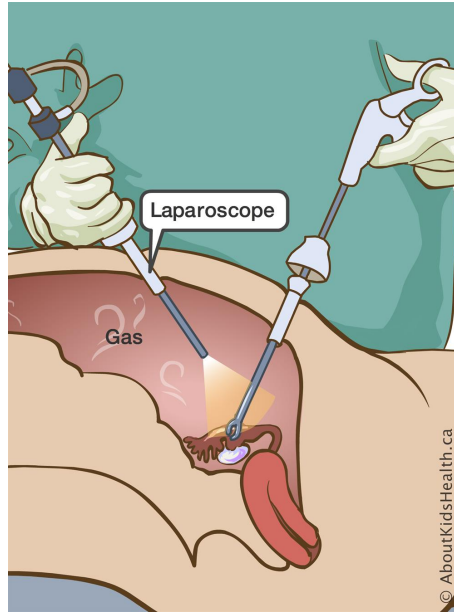


Figure 1.1: Traditional setup for Laparoscopy [1].

incisions on the patient's body. These small incisions are what give laparoscopic surgeries their advantages over open surgery. These advantages are however mostly related to the well-being of the patient. Studies show, that in the surgeries performed laparoscopically, there is less hemorrhaging which directly reduces the need for blood transfusions [6]. Due to smaller and accurate incisions to just the affected area, other organs are not exposed and there is a much lower risk of infection. This is primarily the reason for shorter hospital stays and faster recovery.

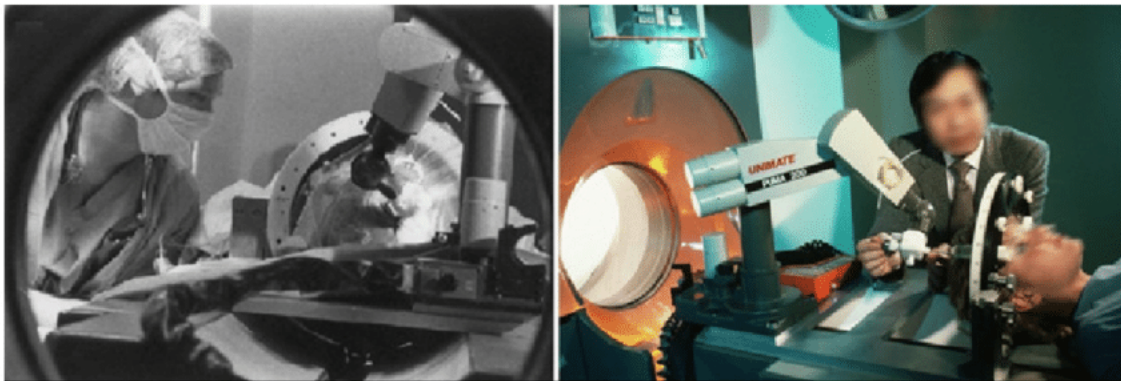


Figure 1.2: The first robot (PUMA-200) to be used for brain biopsies [2].

Since the tools are pivoted at the incision point and are controlled manually, they result in a reflected and a scaled motion. In the context of surgical procedures, this motion is called laparoscopic motion and alters the natural hand-eye or hand-camera (endoscopic) coordination. Furthermore, the tools are usually connected to a rigid shaft thus requiring uneasy motions by the surgeons. These motions are tiring and the setup prevents the surgeons from taking momentary rest without having to remove the tools from the body [7]. This limits the use of laparoscopy for longer surgeries. Even for shorter surgeries, they can cause exhaustion, thereby inducing hand tremors. These tremors are transmitted directly to the surgical area and can be a nuisance at best and fatal in the worst case. These are some of the reasons for which traditional laparoscopy is harder for surgeons.

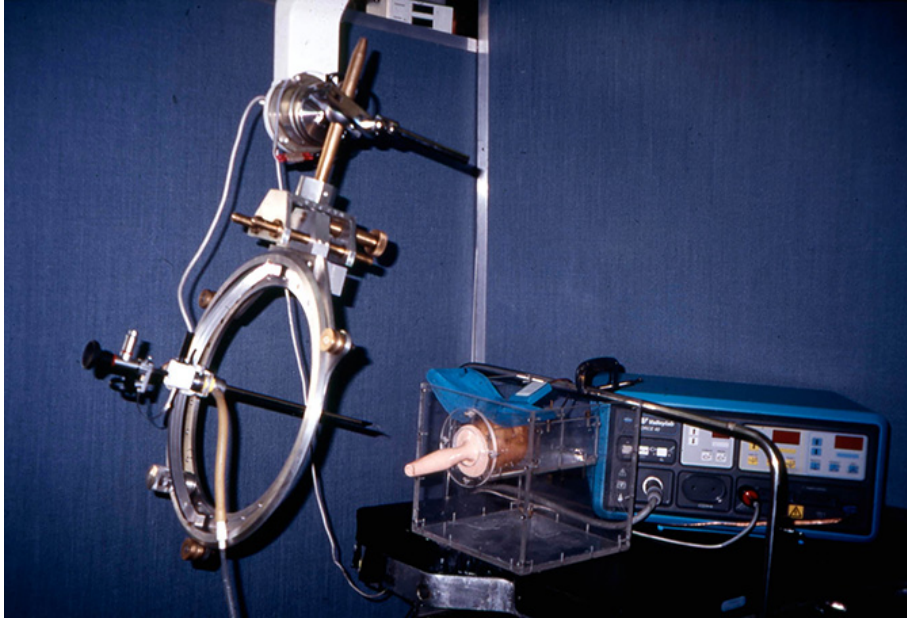


Figure 1.3: A robotic system for prostate resection via the trans-urethral route.

The inclusion of robots, especially teleoperated ones, can mitigate many shortcomings associated with traditional laparoscopy. From a historical perspective, the first robot used for surgery was the PUMA 200 [2], a general-purpose articulated manipulator shown in Figure 1.2. The robot performed a neurosurgical biopsy. A few

years onwards, a similar setup was utilized in performing a transurethral resection of the prostate [6]. Another robot, named Probot [8] (Figure 1.3), was developed at Imperial College London to remove the soft tissues for prostate surgery. However, the breakthrough in robotic laparoscopy happened a little later, in 1994, with a teleoperated robotic setup for cholecystectomy [9].

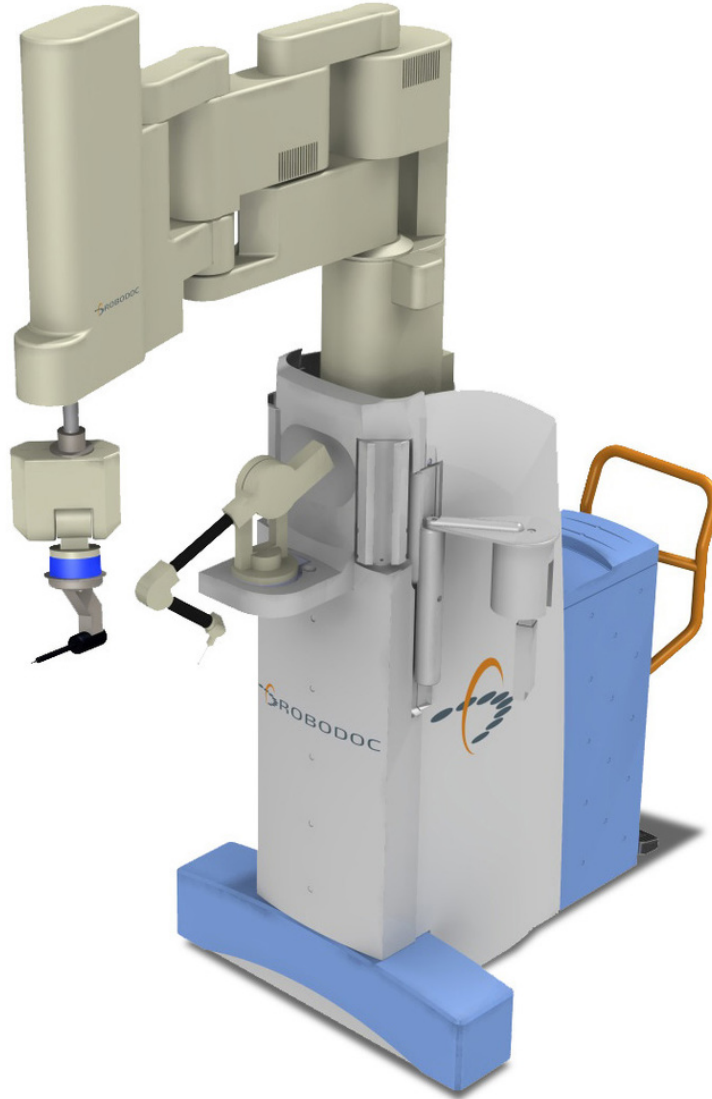


Figure 1.4: The commercial Robodoc intended for hip replacement surgeries

Similar to soft-tissue surgery, the use of robots for hard-tissue related surgeries

was also taking place. A system by the name of ROBODOC [10] (Figure 1.4) was used to perform a hip replacement surgery in 1992. In general, this robot was capable of performing both the revision and total knee and hip Arthroplasty [11]. This system has received several clearances over the years with the most recent one being in 2019 for being able to be marketed for total knee arthroplasty [12]. Interestingly enough, Dr. Kazanzides who happens to be a committee member for this dissertation was one of the co-founders of ROBODOC.

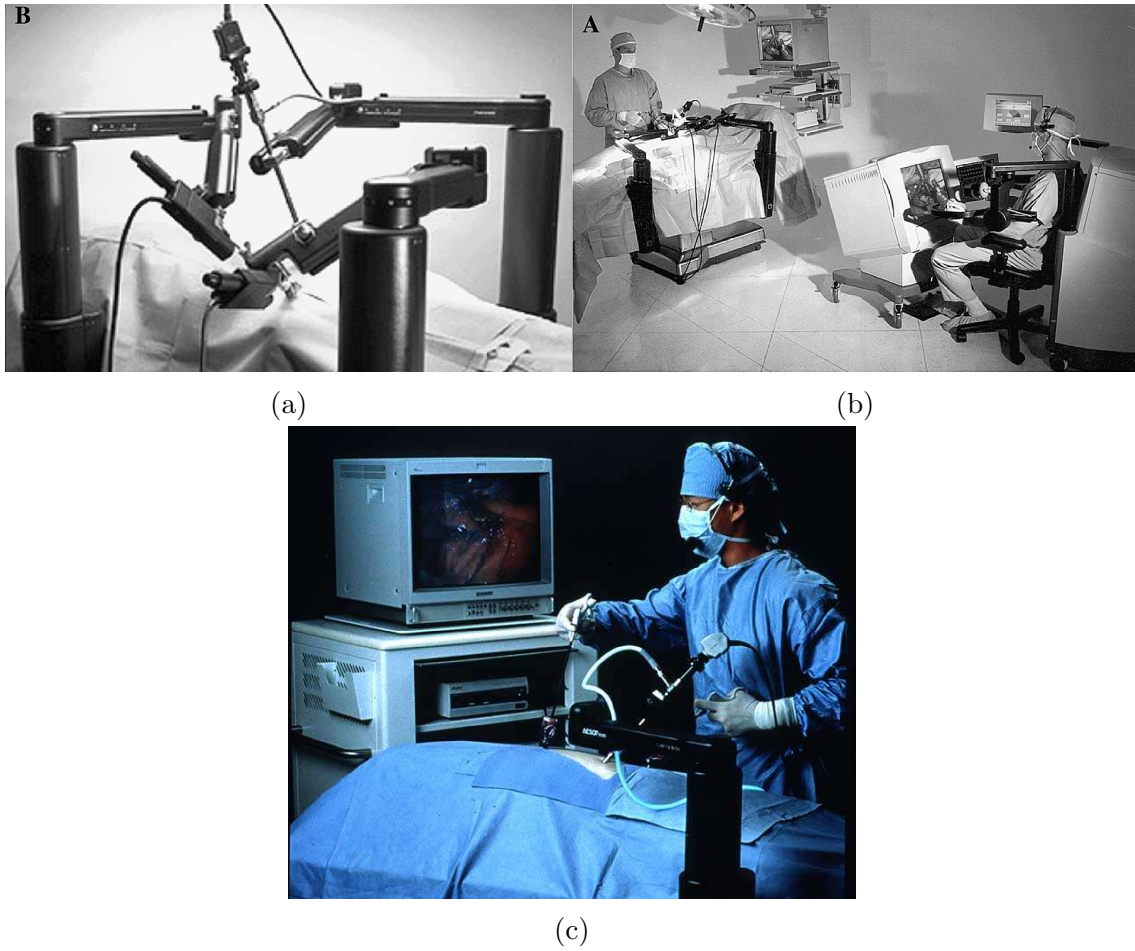


Figure 1.5: (a) The ZEUS Slave Console.(b) The ZEUS system with Master and Slave Console.(c) The automated endoscope called (AESOP)

While the initial robots used for surgery were mostly research projects, it was the commercial enterprises that propelled the field forward. In that respect, a sur-

gical robot by the name of Zeus (Figure 1.5) received FDA clearance for one of its sub-components in 1994. This sub-component was an automated endoscopic module called AESOP. Other than the endoscope, the Zeus system had two teleoperated robotic manipulators that were specialized for laparoscopic and thoracoscopic surgery. The full Zeus system received the final FDA clearance in 2001 [13].

In parallel to the development of the Zeus system, a collaboration between researchers at Ames Research Center (ARC) at NASA and Stanford Research Institute (SRI) [14] resulted in a product that has changed the course of robot-assisted surgery. The project was also joined by the U.S military in its early stages. The military aimed at developing a branched-off sub-project focusing on long-distance teleoperated surgeries. The system resulting from this sub-project was called Mobile Advanced Surgical Hospital (MASH). In terms of human studies, the project never saw the light of day, however, several successful animal studies were conducted [15].

A company by the name of Intuitive Surgical (called Integrated Surgical Supplies at the time) licensed the research project resulting from the collaboration between ARC and SRI. The company filed for the FDA clearance of the system after massive redesign [16]. After a few hiccups, the system finally received the clearance for commercial use in 2001 as a Class-III medical device. This system was called the da Vinci Surgical System and shown in Figure 1.6. The da Vinci Surgical System has transformed the operating room (OR) for the patients, the surgeons as well as the nursing staff. By some estimates, the da Vinci surgical systems have been used to perform more than six million surgeries until 2019¹. While this is still a small number compared to the total number of laparoscopic surgeries, it is quite remarkable in the sense that a single platform (with evolutionary generations) is the major factor in that small number. The da Vinci Surgical System is currently in

¹<https://www.davincisurgery.com/>

its fourth generation with the latest robot called the da Vinci X. Branching off the traditional multi-port setup, the company also has a single port system called the da Vinci SP².

Since 2016, there have been many newcomers to the field of RMIS and surgical robotics in general. Some notable systems include Senhance Surgical System [17] by Transenterix (Morrisville, NC, USA), Hugo [18] by Medtronic (Medtronic Parkway, MN, USA) and Versius [19] by Cambridge Medical (CMR Surgical Ltd, Cambridge, UK) . Additionally, there are several big and small scale companies working towards one or more aspects of surgical robotics which include companies like Verb Surgical (Mountain View, CA, USA), Stryker (Corporate Dr, NJ, USA) and Auris (Redwood City, CA, USA).



Figure 1.6: The da Vinci Surgical System with the surgeon operating the MTM console and the helping nurse at the PSM station

²<https://www.intuitive.com/en-us/products-and-services/da-vinci/systems>

1.1 The da Vinci Research Kit

The da Vinci Research Kit (dVRK) is an open-source version of the first generation of the system. Each first-generation da Vinci surgical system has two master input interfaces called the Master Tool Manipulators (MTMs). These two MTMs can simultaneously teleoperate two corresponding slave manipulators, called Patient Side Manipulators (PSMs). However, the da Vinci system includes not two, but three PSMs. These three PSMs are mounted on a cart which is called the Setup Joint Cart (SUJ). Also on the cart, is an endoscopic manipulator called the Endoscopic Camera Manipulator (ECM). The ECM carries the camera modules that take high-resolution imagery from inside the patient's body. The video from these cameras is viewed by a stereoscopic head-mounted unit that is positioned on top of the two MTMs. Altogether, the set of MTMs and the High Resolution Stereo Viewer (HRSV) Head-Up Display (HUD) is called the Surgeon Console. These components of the dVRK are shown in Figure 1.7.

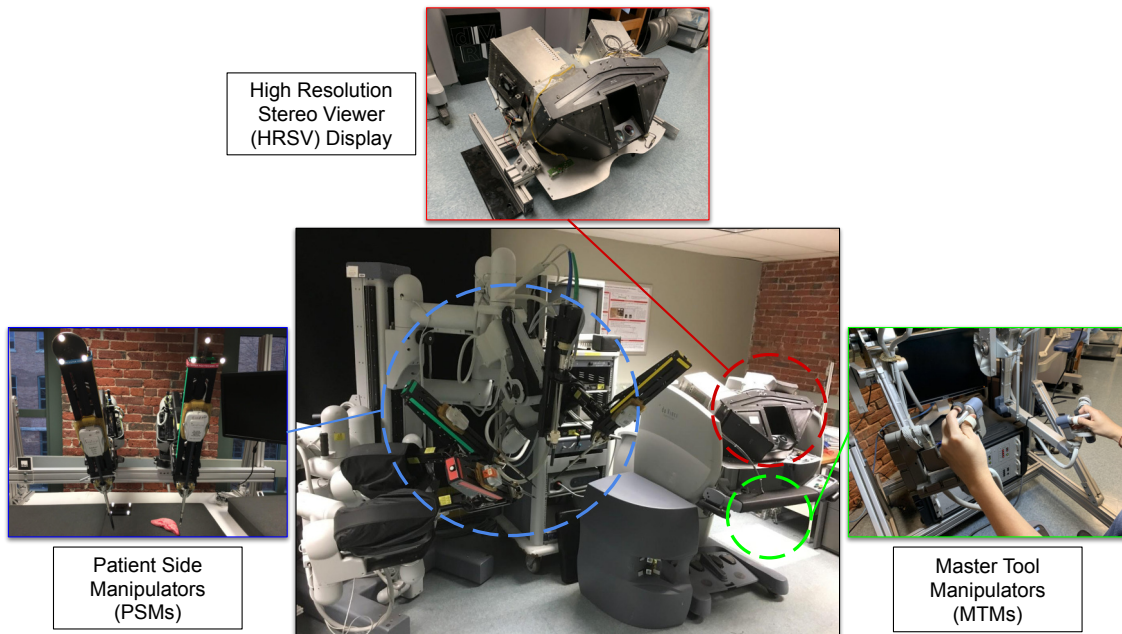


Figure 1.7: The da Vinci Research Kit at WPI Aimlab.

Having laid out the system description and naming terminology of the da Vinci system, it is worth discussing the kinematic and dynamic design of the three dVRK manipulators starting with the MTMs. The MTMs are 7 DOF robotic manipulators with actuated joints. The base of the dVRK MTM has a 4 bar linkage and is also suspended using an assortment of cables and pulleys for reduced distal weight. The PSMs are 6 DOF manipulators with a mechanically constrained remote center of motion (RCM). The mechanical RCM is achieved by the inclusion of two connected 4 bar linkages. Similar to the MTMs, the PSMs have a combination of cables and pulleys that take off the major weight of the end-effector. Abiding by the medical device design practices, the PSMs are designed such that they do not have any actuators at the distal end. Instead, a network of cables, pulleys and couplers is used to control the surgical tools. The ECM looks mostly similar to the PSMs but is mechanically different. Firstly, there are no cables or pulleys to drive the different joints, instead, geared actuators are used. The ECM has only 4 degrees of freedom for spatial positioning and the roll of the camera.

A dVRK setup consists of either a subset of these components or all of these components together.

1.2 Background

The research community focusing on robot-assisted surgery has continued to grow over the years. Specifically for robot-assisted laparoscopic surgery, the main driver behind this growing community is a dedicated effort led by several research institutions as well as companies like Intuitive Surgical (Sunnyvale, CA, USA³) and Applied Dexterity (Seattle, WA, USA⁴). What sets this community apart is that

³<https://www.intuitive.com>

⁴<http://applieddexterity.com/about/>

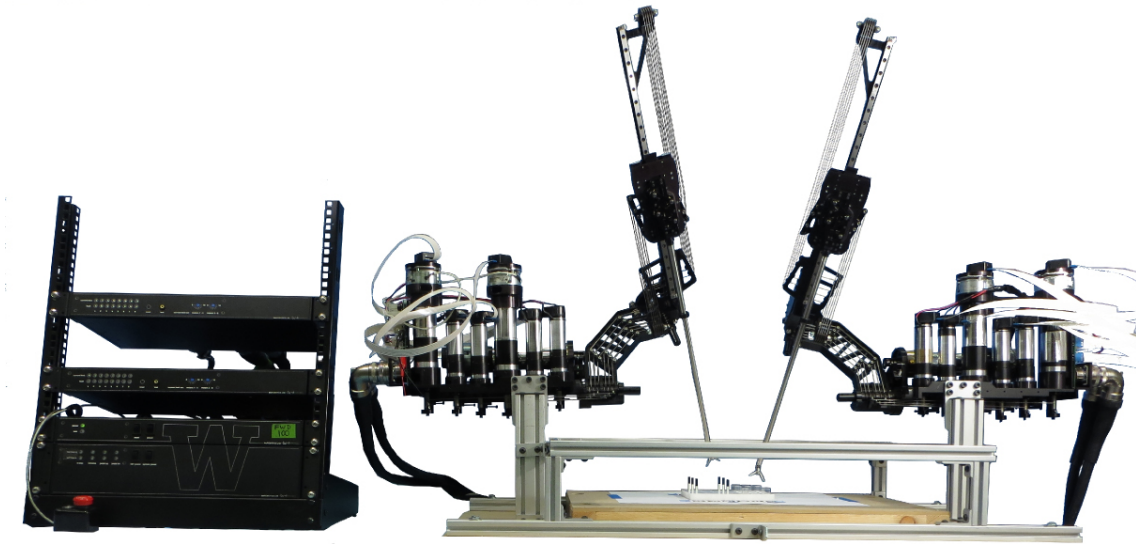


Figure 1.8: A lab setup for the Raven II surgical robot.

some form of informal communication and collaboration keeps happening between different groups and even sub-groups of people working in them. This is on top of the Principal Investigator's (PI) and User Group meetings that are held throughout the year at various robotics conferences.

At the moment, there are more than 50 institutions (> 30 for dVRK and ~ 24 for Raven II) that use the two research platforms. These platforms share some commonalities between their design characteristics and the Raven II platform (shown in Figure 1.8) can use the surgical tools from the da Vinci. In terms of the dVRK, Intuitive Surgical has generously donated the retired clinical da Vinci Surgical Robots to universities not only in the North American region but to many universities in Europe. A geographical map showing the sites is shown in Figure 1.9. Similarly, a map depicting the Raven II sites is shown in Figure 1.10.

The dVRK systems at WPI consist of a pair of Master Tool Manipulators (MTMs), a pair of corresponding Patient Side Manipulators (PSMs) and one Endoscopic Camera Manipulator (ECM). The dVRK (da Vinci Research Kit) systems



Figure 1.9: In 2019 the dVRK is present in 35 sites worldwide.

do not come with any hardware controllers or software support. This is where the role of research institutions comes in and as such, researchers at Johns Hopkins University (JHU) have spearheaded, and the ones at Worcester Polytechnic Institute (WPI) have assisted, in the development, distribution and support of both the custom hardware controllers for interfacing the dVRK Manipulators and the software infrastructure that connects everything. This has enabled all the participating institutions to use the dVRK almost out of the box.



Figure 1.10: In 2019 the Raven II is present in around 20 sites worldwide.

The hardware controllers for interfacing the dVRK include an FPGA based con-

trol unit that communicates with a Quad Linear Amplifier (QLA) board [20] (Figure 1.11). These controllers are open-source both in terms of the schematic design as well as the embedded code that runs on the FPGA, moreover, they are not limited to be used only with the dVRK. Currently, the controllers are in their sixth generation (or sixth revision cycle) which shows continued support over the years by the JHU and WPI researchers.

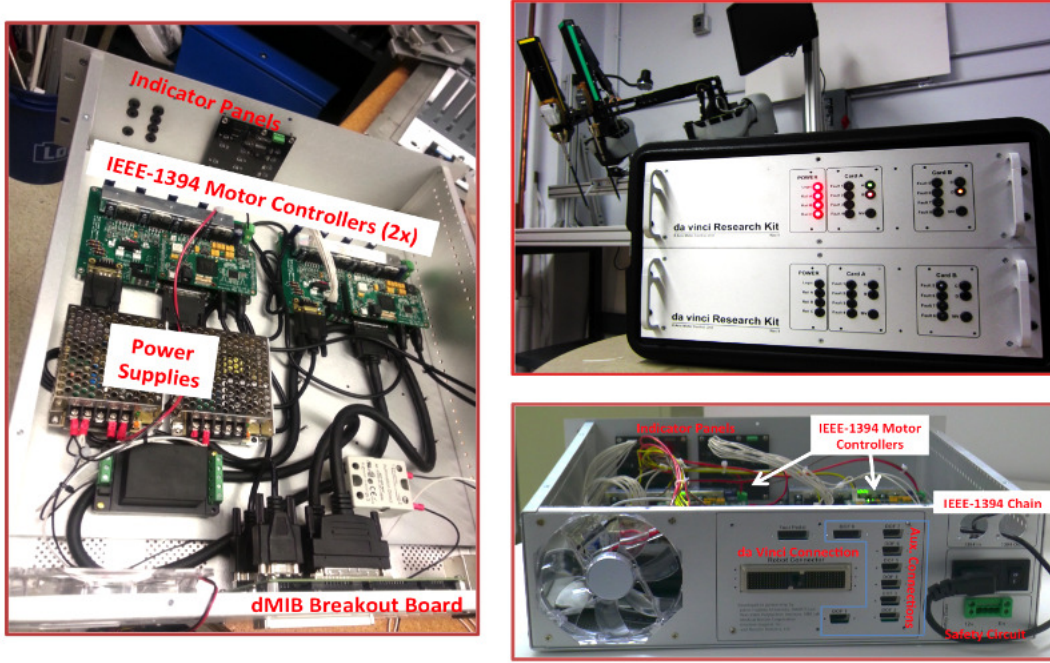


Figure 1.11: The open-source hardware controllers for interfacing with each individual dVRK manipulator.

To interact with the dVRK controllers, extensive software architecture is used. This architecture primarily consists of two separate software packages, namely CISST (Computer Integrated Surgical Systems and Technology) and SAW (Surgical Assistant Workstation). Each package itself contains many stand-alone libraries for various purposes. These packages, similar to the hardware controllers, were developed at JHU as a generic software solution for medical devices, imaging systems and scanners. The customization of these packages to support the dVRK Manipulators

led to an application called **sawIntuitiveResearchKit** [21].

Although the core **sawIntuitiveResearchKit** application was sufficient for tele-operated control of the dVRK Slaves via Master Tool Manipulators, much like the proprietary control hardware and software, it was stumbling block for many nascent researchers (slightly less experienced graduate and undergraduate students in terms of software design) in customizing/extending the software for their specific use-cases and applications. Fortunately, this was recognized early on at a time when dVRK had only been disseminated to a few universities. This resulted in the prototype implementation of a uni-directional bridge (saw-ROS Bridge) that allowed the transmission of specific data to Robot Operating Systems (ROS).

These ROS interfaces were soon used in the implementation of the first da Vinci related simulations in RViz (Robot Visualization) [22]. RViz is the default visualization engine for ROS and used avidly among the ROS community. Apart from being natively built to support ROS, a major reason for its popularity is the simplicity of the interface and the gradual learning curve to visualizing multiple different types of data, in addition to moving robots, all through ROS topics. To name a few, these types include visual markers, point clouds, joint efforts and way-points.

The uni-directional bridge was great in representing the robot state data into robot simulators and data-logging but naturally, this use-case was rather limiting. Hence, not that long after the initial implementation of the uni-directional implementation, the bridge was improved to allow bidirectional communication that supported joint commands from ROS topics. Gradually the bidirectional communication of other types of state-command data was added, the data-types included Cartesian poses, joint efforts and even Cartesian wrenches. The implementation of this extensive bridge was carried out almost in parallel at both JHU and Worcester Polytechnic Institute (WPI). The bridge was also renamed from saw-ROS bridge to

the cisst-ROS bridge.

The bidirectional ROS interfaces of “sawIntuitiveResearchKit” enabled the integration of more advanced libraries that were popular in the ROS/robotics community beyond just mimicking simulators. For instance, a motion planning interface [23], which used MoveIt [24] and Open Motion Planning Library (OMPL) [25], was developed. Similarly, a Matlab interface [26], that utilized the Matlab-to-ROS bridge [27] built for the Matlab [28] software was also integrated. Researchers at the University of British Columbia (UBC) added a similar Matlab support [29] which did not utilize ROS, but instead incorporated “sawIntuitiveResearchKit” directly. This project, however, is now defunct. Nevertheless, the integration allowed the bidirectional control of dVRK manipulators via task-based controllers implemented in high-level software libraries. Despite all the developments happening in high-level software libraries, the development of the core dVRK software continues to date. Due to the size of the community, this multi-level development has the potential of causing conflicts in every new release cycle. To circumvent these commonly encountered issues, a joint effort between the researchers at JHU, University of Washington (UW) and WPI led to the design of a standardized Collaborative Robotics Toolkit (CRTK) [30]. This toolkit provides a “grammar” of sorts for interfacing with robots at all levels of control (low-level, mid-level and high-level). The specifications of this toolkit can be found at [31].

In terms of simulation, RViz met the initial community needs for kinematic testing and evaluation of the dVRK manipulators. However, there was a need for a dynamic simulator to allow more realistic simulations and the implementation of advanced dynamic controllers without putting the physical robots in harm’s way. Although ROS does not have a de-facto dynamic simulator, Gazebo [32], was the default choice for dynamic simulation in the ROS community. Gazebo has its own

visualization interface and various kinds of sensors can be simulated using compiled plugins. Gazebo integrates 4 state-of-the-art physics solver libraries (ODE [33], Bullet [34], Dart [35] and Simbody [36]) for solving the underlying system of equations representing the dynamic bodies and articulated robots. These robots, and environments are specified using an XML [37] based format called the Simulation Description Format (SDF) (<http://sdformat.org/spec>).

The first replicated models for the dVRK robots were created partially, yet independently, at Western University (UWO) in Canada (by a research group headed by Dr. Rajni Patel) and at WPI. More specifically, the models from UWO university include the models for PSMs while the models at WPI included those for MTMs as well as PSMs. While these models replicated the dVRKs visually, the kinematic parameters were taken mostly from visual observations rather than empirical measurements. Due to the innately complex design involving links within links, measuring accurate kinematic data was not possible at the time since the robots would first have to be disassembled and then each link separately measured. Moreover, there was also a lack of publicly available inertial data, thus it was arbitrarily specified in the SolidWorks models. These models were then converted to Universal Robot Description Format (URDF) and from URDF to SDF. The URDF models are XML based files, similar to SDFs, and are used by ROS, in general, to load joint and Cartesian space kinematic representations, and by RViz for visualizations. The SDF models were then modified by hand to include various plugins. Mostly, these plugins allowed the back and forth communication between ROS and Gazebo, and thereby sawIntuitiveResearchKit through the cisst-ROS bridge. Even though Gazebo is fully capable of visualizing all sorts of data using plugins, more often than not, the relevant state data is ported back to RViz due to its simpler ROS topic interface.

With the developments of a working implementation of both kinematic and to some extent a dynamic model for the da Vinci, more research groups started using these simulators with the developed models. However, the initial models of the dVRK were outdated and inaccurate and could not be used in some of the research being carried out in WPI at the time. This led to a re-evaluation of both the kinematic and dynamic parameters. The first study was aimed at carefully constructing, link by link and joint by joint, the accurate kinematic parameters to achieve an implicit closed-loop mechanism, which when solved for, resulted in the end-effector Cartesian pose, numerically identical to the pose calculated by placing optical-tracking markers on the physical da Vincis. For this purpose, the arc lengths of the rotatable links were measured by controlled actuation, while the lengths of non-rotatable links were estimated using approximation techniques (Least Squares Estimation LSE [38]). The results were published in [39] and were used to remake the entire models in Solidworks, then URDF and finally SDF. These models are available at a public repository called “dvrk_env” [40].

Having successfully generated the fully functional closed-loop kinematic model of the dVRK manipulators, attention was focused on calculating accurate dynamic models. This was, of course, more challenging than the kinematic identification, but the effort led by Wang and Gondokaryono at WPI resulted in the estimation of accurate dynamic parameters for all the links of dVRK MTMs and PSMs. Not only that, a generic package was created to allow the parameter identification of other types of closed-loop robots and made publicly available [41]. This work was published in [42].

As the community grew bigger, the need for more advanced simulation software also increased. While the existing simulators (with the implementations of dVRK manipulators) were more or less sufficient in simulating the simplified versions of

dVRK robots, the challenge was that these simulators are not meant for complex closed-loop robots such as the surgical robots. More than just that, a major simulation component used with regards to surgical robotics is the emulation of intractable training environments. While it is true that soft-body simulations are more appropriate for surgical robotics, most of the existing surgeon training tasks, both in reality and proprietary simulators, involve the interaction and manipulation of rigid puzzle pieces in and around rigid body bases. An example training puzzle is shown in Figure 1.12⁵. There are many factors such as reproducibility and lower cost for employing such simpler training environments. More importantly, though, these puzzles can be better for targeted training and evaluation.

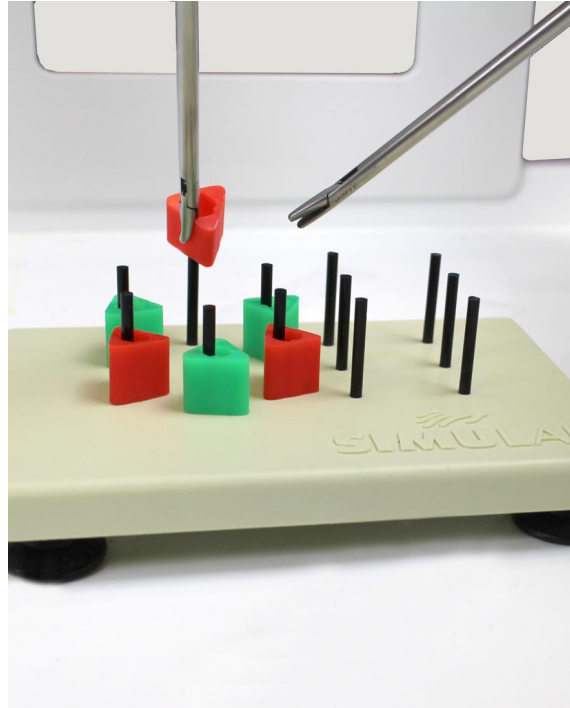


Figure 1.12: The peg transfer puzzle for surgical training.

These limitations, along with several attempts to keep up-to-date with the simulators popular in the community, led to the implementation of a new framework

⁵<https://www.simulab.com/products/laparoscopic-trainers/peg-transfer-board-triangles>

called the Asynchronous Framework (AF). The associated simulator that makes use of this framework is called the Asynchronous Multi-Body Framework (AMBF). A more detailed motivation for the development of this framework is presented in the next Section 1.3.

1.3 Motivation

As discussed in Section 1.2, the models for the dVRK were developed for some popular community simulators (Gazebo and RViz). These models and the associated plugin-based software was kept updated throughout the years to keep in sync with the API changes. The ROS based visualizer, RViz, is meant solely for serial robots (with indirect ways of specifying closed-loop robots). Gazebo, on the other hand, supports closed-loop robots through its specification, but the API for controlling the redundant joints (through various plugins) changed throughout the years or simply dropped support for such robots. Even the core API of Gazebo which was used to write specialized plugins, discussed above [40], changed in less than a year after its release. To understand the support of or lack thereof for closed-loop robots in simulators, one first has to understand the underlying design philosophy of these simulators. One can, for instance, take a look at the popularly used Denavit Hartenberg parameters [43]. DH parameters simplify the specification of articulated robots by allowing a single D.O.F. along each joint axes. This specification works great and is taught and understood commonly, however, it cannot be used for the specification of closed-loop robots in its original form. Similarly, the TF library [44] arguably forms the backbone of ROS and deploys a tree of connected bodies to form a spatial structure. This philosophy mandates that a link (body) must have one and only one parent link (similar to DH). RViz, MoveIt and “ros_controllers” are only

a few ROS packages that build upon TF. While this design choice makes sense for the most common types of general-purpose robots, it is this exact design philosophy that goes against the specification and simulation of closed-loop robots. Since most of Gazebo’s user base utilizes it with ROS and ROS based tools, Gazebo is more or less limited to the design philosophies adopted by these ROS packages. The same is true for VRep.

While this in itself was motivating enough to develop a simulation platform specialized for surgical robots, it was not the primary reason for the origination of the Asynchronous Framework. That motivation was the result of the prevalent need for improving the outcome of robot-assisted surgery. There are of course many different pathways taken by different researchers to address this, all the way from the better design of devices, smaller and more precise instruments, cooperative/shared control and inclusion of force/tactile feedback to the input device. These different research areas have made their way to the Operating Rooms and clinical trials in one form or another.

There is a relatively newer trend of incorporating semi-autonomous agents trained via Machine or Reinforcement learning (ML / RL) into the surgical workflow. Of course, these cannot be applied directly to an actual surgical procedure but instead, on mock setups that use non-clinical versions of actual surgical robots such as the dVRK and Raven. Some notable research in this area includes autonomous algorithms for performing soft-tissue suturing [45], an automated approach for sinus surgery using computer navigation techniques [46], characterization and automation of soft-tissue suturing using a curved needle guide [47] and automation of cutting/creasing sub-tasks while employing learning by observation [48]. Additionally, [49] presents a holistic approach to simplifying the task of manipulator positioning prior to surgeon interaction, and [50] demonstrates a telemanipulated surgical

simulation designed for heart surgery. A trainable infrastructure is presented in [51] with controllable dominance and aggression factors for automating repetitive surgical tasks. Lastly, a shared infrastructure for collecting da Vinci Research Kit (dVRK) manipulators and vision data, primarily for training learning agents by motion decomposition of sub-tasks is developed in [52].

The initial setup for performing these experiments requires the setup of many external sensors including motion capture systems, depth sensors, and uni or stereo vision cameras along with data-collection and labeling from these sensors. In addition to the effort required, these are extremely time-consuming tasks and neither the time spent nor the effort undertaken, reflect in any way in the experimental results. Moreover, in many instances, the experimental setups do not mimic the actual surgical conditions anyway and thus the required time and effort becomes a major stumbling block in repeatable research.

Specialized simulation software can simplify most of these tasks and allow reproducible and shareable research. The state of the art of existing work for surgical simulations is mostly limited to applications catering to specific operating environments. Some notable products include Simbionix⁶, Mimic simulator⁷ and CAE LapVR⁸. These are proprietary applications and designed for interfacing specific surgical robots. A primary focus, in these applications, is the rendition of impressive visuals that mimic the actual surgical scenes, and, the dynamics of soft-tissue interaction for specific use-cases that are highly optimized.

There are also some open-source software that includes SOFA [53] and Open SurgSim⁹ by Simquest (Silver Springs, MD, USA) that can simulate specific surgical training sub-tasks, render visuals and perform soft-body interactions that are almost

⁶<https://simbionix.com/simulators/robotix-mentor/da-vinci-surgery/>

⁷<https://mimicsimulation.com/da-vinci-skills-simulator/>

⁸<https://caehealthcare.com/surgical-simulation/lapvr/>

⁹<http://www.simquest.com/opensurgsim.html>

equivalent to the proprietary software mentioned above. While visual realism and tissue interaction is a useful feature of the surgical training, since these applications are use-case specific, they lack the basic set of tools required by researchers for plugging generic input devices and creating custom environments for different use-cases. What is needed is a simulation framework that is not only robust enough for the specification and simulation of rapidly prototypeable environments but can also handle the complexity associated with interfacing multiple users as well as AI to interact with the simulated environments.

This was, in essence, the motivation behind the research and development of the Asynchronous Framework. Although the motivation was clear, the challenges to the implementation of such a framework had to be first understood and based on these challenges, the requirements had to be specified. The explicit “requirements specification” was good practice not only from the perspective of “Systems Engineering” but also gave a baseline design philosophy for the framework. This design philosophy has been followed to date and all the additions since then have been formulated accordingly.

The requirements for the Asynchronous Framework can be classified broadly according to the following list:

1. Real Time Dynamic Simulation:

Humans perceive the kinematics of everyday objects in a constantly incrementing time-frame. As an example, a grasped body being moved with constant velocity will appear as such to the user. Under non-real time simulated physics, the simulation time does not track the real-world clock and hence from the perspective of a human, the simulation may seem faster in some instances and slower at others. This can also be referred to as “time slippage”. Correspondingly, real-time dynamic simulation is required for interactive training

tasks.

2. Real Time Multi-Device Control:

For the Asynchronous Framework, the user interaction encompasses the use of input interface devices (IIDs) (such as hand controllers, haptics devices, mouse, foot-pedals, etc.) to manipulate simulated bodies. To render the interaction and manipulation more realistic, one can incorporate force feedback for devices that support it. Since such feedback requires high-speed control loops (usually $\geq 1kHz$), a framework that is capable of handling multiple high-speed devices is required. Ideally, the framework should be able to handle a mix of haptic and non-haptic devices, i.e., different drivers with faster and slower update rates.

3. Contextual View Port Control:

In a multi-user training environment, not all users share the same view (camera). This requirement focuses on allowing multiple view-ports to be defined for different users and allowing the distributed control of these view-ports. For example, a specific user can control (re-position or re-orient) their view-port by using a specified button on their IID. Furthermore, multiple different users can share a single view-port and then control the view-port independently. Specifically for orientation control, this gets complicated as one has to make sure that each user can orient the camera in their frame of reference (FoR) (which could be different between the different users sharing the camera) and the camera motion is both continual and smooth. Continual in this context means the change in camera orientation starts from the camera's current state instead of resetting the camera transform to the controlling users FoR and starting the rotation from thereon.

4. Parametric Manipulation and Interaction:

Since interactive simulations have a component of manipulation and grasping, it is important to understand and model contact-based friction in simulation. Due to the underlying mathematical formulation of rigid body dynamics, rendering contact-based friction is both difficult to model and inconsistent [54]. In the existing open-source applications that involve simulated manipulation, the problem is addressed by imposing kinematic and dynamic constraints for grasped objects in relatively simpler environments. For a general-purpose framework, the main idea is to allow the extensible response of simulated objects when interacted upon via IIDs. This response includes the grasping dynamics of simulated bodies, cutting of soft-bodies and collision aided manipulation in and around fixtures. In light of all this, a generic and parametric interaction component is required for the framework.

5. Simulation of Model Accurate Surgical Robots:

Surgical robots are usually the most complex forms of robots and a simulation framework that can cater to requirements for such robots, if implemented correctly, can be used to simulate general-purpose robots. The challenges to the simulation of these robots begin at the specification level. For example, how can an interconnected graph of links, usually true for surgical robots, be represented in a sequential specification? Can such specification be simplified and generalized such that robots that are only serially linked can also be easily specified? Moving on from the specification, how can these robots be simulated such that their constraints are satisfied in a real-time dynamic simulation? These questions result in the requirements for a specialized robot simulator that can also simulate everyday robots.

6. Interface for Training Learning Agents:

A major goal of a developing a simulator targeting real-time, interactive manipulation, is to use this framework for training not only the human operators but to also train artificial intelligence (AI) for performing semi-autonomous tasks. There exist a large number of open-source software libraries for machine learning (ML) and reinforcement learning (RL) that can be used for training purposes. Most of these libraries support the Python language and therefore a compatible interface has to be developed to leverage their API. Developing a Python interface for a real-time simulation framework that is also asynchronous is challenging. One significant challenge is compensating or accounting for the effect of the delay between reading the states, applying a command (action) then re-reading the response (states) of the corresponding command. Regardless of having the support for learning libraries, the advantage of integrating a well designed Python interface is that it offers a low barrier to entry for new users and provides the capability for accelerated testing and deployment of control algorithms.

7. Visualization Engine:

This requirement is rather straight-forward to describe but arguably complex to implement due to the inclusion of the above requirements. The major challenge in the context of the asynchronous framework is to maintain the speed of the visualization loop since the view-ports (cameras) are shared with asynchronous IIDs. Other challenges include the use of advanced rendering features such as textural mapping, multiple light sources and the simulation of soft-bodies.

While existing software libraries cater to one or more of the aforementioned

categories, it is the underlying design of these libraries that limits the inclusion of all these subproblems into a single framework. This is explained in more detail in the chapters to follow but an example is presented here.

Consider the first two items in the above list which are 1) Real-Time Multi-Device Control and 2) Real-Time Dynamic Simulation. A real-time multi-device control requires the consideration of real-time constraints for reading each devices' states, computing the control laws, and feeding back commands (forces for haptic devices). These constraints can be modeled as either hard real-time or soft real-time depending on what Operating System is being used. The control law(s) can be a simple error based force control for joint or task-space or more advanced controllers such as impedance controllers.

These types of control laws can have deterministic compute times, which makes the inclusion of multiple input devices in a sequential control loop (where one device is addressed after the other) possible. Even a parallel implementation (where devices run independently of each other) is trivial as the input devices do not depend on each other for the computation of their control laws.

The addition of dynamic simulation in the mix complicates things since the input devices are no longer independent, instead, they are all interconnected to the dynamic simulation, as well as to each other through their proxy simulated dynamic bodies. The computational time of each iteration of dynamic simulation is never deterministic. This thwarts one's choice of implementing a sequential control approach as the device updates need to be deterministic (real-time) which can no longer be guaranteed because of the iteration of the dynamic simulation. A parallel approach is preferable instead, however, even that has its own set of problems. The requirement number 3) "Contextual View Port Control" compounds these problems further as it requires the sharing of the data from the visual loops in both the

dynamic update and device update computations.

Due to the associated challenges to the requirements listed above, the Asynchronous Framework (AF) (and the associate simulator AMBF) is a ground-up implementation of a parallel design philosophy that allows asynchronous real-time control of multiple input devices and communication interfaces in a real-time dynamic simulation. The list of contributions is presented in the next section.

1.4 Contributions

The work presented in this manuscript can be divided into both conceptual contributions and implementation based contributions. Besides, the open-source nature of the project has made a reasonable community impact. These contributions and the impact on the community is presented in this section.

1.4.1 Conceptual Contributions

A list of conceptual contributions of this dissertation is as follows:

1. A conceptual asynchronous framework for control of multiple haptic and tracker input devices with a real-time dynamic simulation.
2. A minimal frame representation for multi-lateral collaborative control in multiple, continuously varying, and shared frame of references (FoRs).
3. Design of an asynchronous and distributed communication pipeline and payloads for online training with a real-time interactive dynamic simulation.
4. The design of a front-end specification format for robots, environments, input devices and soft-bodies.

5. A framework for the emulation of proximity, contact and resistive sensors using user-specifiable parametric data. The user can also specify the desired contour for the sensor population by providing corresponding mesh shapes. All this data is specified through the front-end specification format.
6. Implementation of a case study to demonstrate and validate the proposed simulation framework.

1.4.2 Implementation Based Contributions

The list of implementation based contributions is presented as follows:

1. The implementation of the Asynchronous Multi-Body Framework (AMBF). AMBF is a versatile simulator for robot and soft-body simulation in addition to being used for multi-user interaction and training.
2. Implementation of plugin based interfaces for the dVRK MTMs, Geomagic Touch¹⁰, Novint Falcons¹¹ and Razer Hydras¹².
3. Design and Implementation of a robust Python client for the AMBF.
4. Implementation of plugins for animation software for the rapid development of robots and environments and the implementation of a converter script that can leverage the largest data-base of existing robot models specified in a different format.
5. The implementation of friction-based grasping for natural manipulation using the resistance based sensors discussed in the conceptual contributions.

¹⁰<https://www.3dsystems.com/haptics-devices/touch>

¹¹<https://github.com/libnifalcon/libnifalcon>

¹²<https://support.razer.com/console/razer-hydra/>

1.4.3 Community Impact

AMBF was publicly released for the surgical robotics community in April of 2019 with a BSD license (<https://opensource.org/licenses/BSD-3-Clause>). To the extent of the author’s knowledge, AMBF has found active user-bases in more than 7 universities to date. These universities include the University of Washington at Seattle¹³, University of Virginia¹⁴, The Hamlyn Center at Imperial College London¹⁵, University of Leeds¹⁶, University of California at Berkeley¹⁷, Politecnico di Milano University¹⁸ and finally the home institution, Worcester Polytechnic Institute (WPI). At WPI, AMBF is being used by several groups of students for directed research, course projects and thesis work. Apart from training applications, some of the recent projects include 1) the control and validation of lower limb exoskeletons, 2) implementation of dynamic force based controllers for the simulated dVRK robots, 3) a generic plugin for forward and inverse kinematics for the surgical robots and 4) augmentation of robotic manipulators for upper limb rehabilitation therapy. Some of these projects have also resulted in the contributions back to the AMBF.

The work presented in this manuscript has also received a couple of awards that include the recognition for “Best for Research Community Award” at the International Hamlyn Symposium 2019 and the finalist for the “Best Application Paper” at the conference for Intelligent Robots and Systems (IROS) 2019.

¹³<https://www.washington.edu/>

¹⁴<https://www.virginia.edu>

¹⁵<https://www.imperial.ac.uk/hamlyn-centre/>

¹⁶<https://www.leeds.ac.uk/>

¹⁷<https://www.berkeley.edu/>

¹⁸<https://www.polimi.it/en/>

1.5 Organization

This manuscript is organized into 8 chapters including the introduction. The remaining 7 chapters are summarized as follows:

2. **Review of Solvers for Dynamic Simulations** The use of computer physics simulation is an integral part of the Asynchronous Framework. There are several different ways of computing the simulated dynamics for these physics simulation problems. A general review of the basic methods is presented in Chapter 2. The review leads to a discussion on which method is preferable over others for being used in the Asynchronous Framework.
3. **Asynchronous Framework for Collaborative Control** This chapter starts by discussing the underlying challenges towards the implementation of a dynamic simulation framework that interfaces multiple input devices, view-ports, distributed controllers and learning agents. Addressing these challenges, the Asynchronous Framework is presented and its implementation details are discussed. The extensive control interfaces are embedded into a novel representation which is called “The XVII Representation” for collaborative and multi-lateral control. The chapter then discusses the design specifications of the communication interfaces for the learning agents in Python. Finally, the evaluation of various performance characteristics along with the discussion of the significance of the results is presented.
4. **Distributed Format for Robots, Environments and Devices** The focus of this chapter is to present a novel description format for specifying and simulating rich environments for the Asynchronous Framework and its simulator, AMBF. The results section of this chapter shows examples of some of the surgical robots. The communication interfaces discussed in the previous chapter

are also extended to the simulated robots and bodies using the description format.

5. **Integration of Soft-Body Simulations** For most robot dynamics simulators, soft-body simulations are an unrelated and therefore un-addressed problem, but for a simulation framework targeting the surgical robotic community, this is not the case. The specification format for robots and environments from the previous chapter is extended to allow the description of soft-bodies. This description is used to simulate rapidly prototypeable soft-bodies in AMBF and a theme of discussion of the chapter.
6. **Grasping in Simulation** A parametric approach for achieving two different kinds of grasping is discussed in this chapter. One approach uses sensor-based constraints to dynamically affix a grasped simulated body which is manipulable by an IID while the second approach models sensor-based friction for a more realistic grasping methodology. The chapter discusses the results of the grasping approach on various multi-manual simulated tasks.
7. **Application and Use-Cases** This chapter discusses two user studies carried out using the AMBF. The first user-study was performed at the Hamlyn Center (Imperial College London) by Junhong Chen and Dan-Dan Zhang in collaboration with researchers from WPI and involved the evaluation of a semi-autonomous control scheme for a simulated puzzle for surgical training. The second user-study was carried out at WPI and analyzed the effect of various forms of collaborative control, using the implementation of “XVII Representation” in the Asynchronous Framework, on the performance of a surgical training puzzle.
8. **Conclusion and Future Work** This is the final chapter and concludes the

manuscript by discussing the proposed future additions as well as the work in progress for the Asynchronous Framework. It also discusses the lessons learned during the development of the framework.

1.6 Acknowledgement

The work presented in this dissertation includes 1) accepted work for publication 2) work under-review for various publications and 3) published work which has not been discussed at length in this dissertation but the results from which have been used. These publications are listed here altogether and then specifically mentioned in relevant chapters.

- Munawar A, Fischer G, “*An Asynchronous Multi-Body Simulation Framework for Real-Time Dynamics, Haptics and Learning with Application to Surgical Robots*”, Intelligent Robots and Systems (IROS), Macao, China, 2019.
- Munawar A, Wang Y, Gondokaryono R, Fischer G, “*A Real-Time Dynamic Simulator and an Associated Front-End Representation Format for Simulating Complex Robots and Environments*”, Intelligent Robots and Systems (IROS), Macao, China, 2019.
- Munawar A, Srishankar N, Fischer G, “*An Open-Source Framework for Rapid Development of Interactive Soft-Body Simulations for Real-Time Training*” In Review for International Conference on Robotics and Automation (ICRA), 2020.
- Munawar A, Srishankar N, Fichera L, Fischer G, “*A Parametric Grasping Methodology for Multi-Manual Interactions in Real-Time Dynamic Simulations*”, In Review for Robotics and Automation Letters (RAL), 2020.

- Wang Y, Gondokaryono RA, Munawar A, Fischer GS, “*A Convex Optimization-based Dynamic Model Identification Package for the da Vinci Research Kit*”, IEEE Robotics and Automation Letters (RA-L), Vol 4, No 4, Oct 2019.
- Gondokaryono RA, Agrawal A, Munawar A, Nycz CJ, Fischer GS, “*An Approach to Modeling Closed-Loop Kinematic Chain Mechanisms, applied to Simulations of the da Vinci Surgical System, Special Issue on Platforms for Robotics Research*” - Acta Polytechnica Hungarica, Vol 16, No 8, pp 29-48, Nov 2019.
- Junhong C, Zhang D, Munawar A, Fischer G S, Yang G Z, “*Supervised Semi-Autonomous Control for Surgical Robot Based on Bayesian Optimization*”, In Review for International Conference on Robotics and Automation (ICRA), 2020.
- Su Y H, Munawar A, Deguet A, Lewis A, Lindgren K, Li Y, Taylor R H, Fischer G S, Hannaford B and Kazanzides P, “*Collaborative Robotics Toolkit (CRTK): Open Software Framework for Surgical Robotics Research*”, In Review for International Conference on Robotic Computing (IRC), 2020.

This research was funded by two research grants which are listed below:

- National Robotics Initiative (NRI) grant: **IIS-1637759** through the National Science Foundation (NSF).
- National Science Foundation (NSF) AccelNet grant: **1927275**

Chapter 2

Review of Solvers for Dynamic Simulations

2.1 Introduction

The work presented in this manuscript involves the integrations of Input Interface Devices (IIDs) into a physics simulation with support for learning and training. The inclusion of the physics simulation in the framework is a major attraction for being used for learning, training, control and manipulation as well as a low barrier to entry, simulation tool. However, the physics solver itself has not been developed as part of this work. There are plenty of choices for competing open-source libraries out there. Based on various considerations among these, the Bullet Physics library [34] has been integrated.

The primary physics solver used in the Bullet Physics library is called the “Sequential Impulse” solver. This solver is a form of “velocity based method” which uses “maximal coordinates”. It is useful to understand the selection of the sequential impulse solver over any other solvers that may exist out there. For this reason,

this chapter briefly discusses the commonly used methods for physics simulation and then compares them to make the case for the inclusion of velocity-based methods for the Asynchronous Framework.

The chapter first outlays the equations of motions representing a free body acted on by constraints and then presents the applications of force-based methods (Section 2.3), velocity-based methods (Section 2.4) and position-based methods (Section 2.5) for solving the underlying system of equations. This is followed by the generalization of these methods to indirect methods (Section 2.6). The limitations of these methods for simulating articulated robots that can be modeled using reduced coordinates are presented next. These limitations lead to the discussion of articulated body methods (Section 2.8) and their shortcomings in modeling closed-loop robots. Finally, a conclusion is presented in Section 2.9.

2.2 Mathematical Formulation of Dynamic Simulations

The simulation of rigid body dynamics has been an area of interest not only for research and development for robotics research but also for entertainment purposes. Due to the influence of the entertainment industry, quite often, the visualization of a simulation is confused with the state of the underlying physics. The gaming industry is the largest market for both physics simulation libraries and visualization libraries. Due to the evolution of computer hardware and the push by the gaming industry, the state-of-the-art for visualization engines has improved tremendously over the years. On the contrary, the physics simulation libraries, especially for games, have seen less drastic of a change. For physics simulation, there exist a large number of libraries, which are all quite capable of rendering rigid-body dynamics. Despite a

large number of libraries, the methods for computing the underlying physics can be grouped into three to four categories, depending upon how the segregation is done.

Any rigid body in simulation is subject to the laws of Newtonian dynamics. The goal is to compute the effective position of everybody at each time-step of the physics simulation. The motion of bodies is restricted by various “constraints”. The term “constraint” is used to broadly classify all the restrictions imposed on the motion of a rigid body. These “restrictions” may include, constraints due to collision, joint based constraints and even springs and dampers can be classified as constraints. The modeling and solution of constraint-less physics simulation are quite trivial and once can simply use the analytical methods [55] for the best results.

On the contrary, incorporating constraints into simulated physics makes the modeling more challenging, and the computation, error-prone. The various methods modeling these problems are 1) Force Based Methods, 2) Velocity Based Methods, 3) Position Based Methods and finally, 4) Indirect representation as Linear Complementary Problems (LCP). LCPs are not always treated as a separate method, as quite often, the velocity-based methods too, can be represented using mixed linear complementary problems (MLCPs). Similarly, the position-based methods can also be re-organized into an LCP formulation. The following sub-sections briefly review these methods individually.

2.3 Force Based Methods

The force based methods are the simplest in terms of modeling the system of equations representing a constraint based physical system. Formulating the equation of motion with the consideration for “constraints” separately from the forces and moments applied by the external controllers, one can arrive at the following form of

the Newton's second law:

$$[\vec{a}_n, \vec{\alpha}_n]^T = [(\vec{F}_{ext} + \vec{F}_{constraints})/m, (\vec{\tau}_{ext} + \vec{\tau}_{constraints})/I]^T \quad (2.1)$$

Wherein \vec{a}_n and $\vec{\alpha}_n$ are the linear and angular acceleration of a body at n time-step. \vec{F}_{ext} and $\vec{\tau}_{ext}$ are the vectors of external force and torque and $\vec{F}_{constraints}$ and $\vec{\tau}_{constraints}$ are the vectors of internal force and torque due to the constraints. m is the mass and I is the inertia of the simulated body.

$$[\vec{v}_n, \vec{\omega}_n]^T = [\vec{v}_{n-1}, \vec{\omega}_{n-1}]^T + [\delta\vec{a}_n, \delta\vec{\alpha}_n]^T \quad (2.2)$$

Here \vec{v}_n and $\vec{\omega}_n$ are the linear and angular velocity at time-step n . The linear and angular acceleration \vec{a} and $\vec{\alpha}$ are the basis for solving the dynamics problem using this approach. These accelerations can be integrated to compute the linear and angular velocities and then, from the velocities, the position and orientation. Both integration steps require the use of temporal states and a time step δt . The integration itself can be carried out using numerical integration techniques which can be divided into explicit and implicit Euler methods [56]. Implicit methods are more accurate compared to explicit methods.

Although the computation of constraint-less dynamics can be carried out using implicit (backward) Euler method, the inclusion of constraints renders stiffness [57] in the underlying ordinary differential equations. Stiffness in this regard refers to the limitation in the choice numerical integration technique (either forward or backward Euler method) to attain convergence without minimizing the time-step δt . Consequently, while implicit methods are theoretically preferable due to their greater flexibility in using a higher value of the time-step Δt , the complexity of implementation and the lack of guarantee for the existence of a solution [58] limits

their use. The other alternative is explicit methods that have an inherent inaccuracy that is driven by a combination of factors including the types of constraint and the length of the time-step dt . The use of the symplectic Euler method [59], also referred to as the semi-implicit Euler method, is a proposed improvement which has better convergence when compared to pure explicit methods. The symplectic Euler method for our problem can be formulated as:

$$\vec{v}_{i+1} = \vec{a}_i \Delta t + \vec{v}_i \quad (2.3)$$

$$\vec{x}_{i+1} = \vec{v}_{i+1} \Delta t + \vec{x}_i \quad (2.4)$$

Due to the use of \vec{v}_{i+1} in the computation of \vec{x}_{i+1} , this method is a mix between implicit and explicit Euler method, with the added advantage of linear computation time. Nonetheless, the only way of incorporating the “constraints” is to include their exerted force. In specific cases, the use of numerical integration makes these methods unstable. This can be observed in even the most simplistic scenarios such as primitive shaped boxes being stacked on top of each other. There are some proposed improvements to force-based methods that work marginally better but overall, these methods are not usually employed by most physics simulators. GEL (<https://www.chai3d.org/forum/gel>) is a soft-body simulation library that uses direct force-based methods for soft-body simulation.

2.4 Velocity Based Methods

Velocity based methods are the most popular in terms of their use in physics computation libraries and as the name indicates, rather than manipulating the magnitude

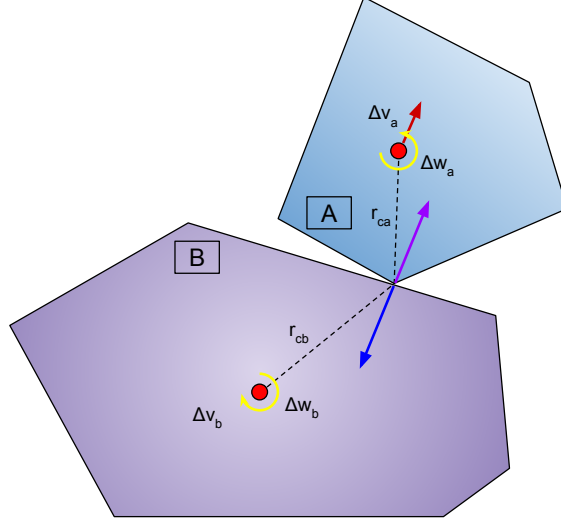


Figure 2.1: The simplest form of constraint based on collision between two rigid bodies.

of force applied by constraints, the change in velocity is calculated directly. The change in velocity is related to the impulse of a dynamic body.

$$P = m_a \Delta \vec{v}_i \implies \Delta \vec{v}_i = P/m_a \quad (2.5)$$

$$L = I_a \Delta \vec{w}_i \implies \Delta \vec{w}_i = L/I_a \quad (2.6)$$

Here, P and L are the linear and angular momentum of the rigid body and I_a is the inertia tensor expressed in body coordinates:

$$I_a = R_a I R_a^T \quad R_a \in 3 \times 3 \quad (2.7)$$

The required change in velocities depends upon the application of the correction impulses P and L . This is the main challenge to a stable implementation of velocity Based Methods. This problem can be explored by using the simplest form of a constraint, which is the collision between two rigid bodies shown in Figure 2.1. The bodies have linear and angular velocities and happen to collide with each other at a

certain time t . The goal of the velocity-based collision constraint is to generate the required impulses that prevent penetration inside each other. This can be accomplished by leveraging the relative velocity between the contact points belonging to each body.

$$\vec{v}_{ca,cb} = \vec{v}_a + r_{ca} \times \vec{\omega}_a - \vec{v}_b - r_{cb} \times \vec{\omega}_b \quad (2.8)$$

Differentiating once to get the change in $\vec{v}_{ca,cb}$:

$$\Delta \vec{v}_{ca,cb} = \frac{\vec{P}_a}{m_a} + r_{ca} \times \frac{\vec{L}_a}{I_a} - \frac{\vec{P}_b}{m_b} + r_{cb} \times \frac{\vec{L}_b}{I_b} \quad (2.9)$$

For the example illustrated in Figure 2.1, the angular momentum L can be written as the cross product of the support vectors and the linear momentum $L = r \times P$. This $\Delta \vec{v}_{ca,cb}$ gives a hint towards the calculation of the correction momentum:

$$P_{correction} = M_{effective} \Delta \vec{v}_{correction} \quad (2.10)$$

The change in relative velocity $\Delta \vec{v}_{ca,cb}$ has the terms P_a and P_b in them. One velocity based method, called Sequential Impulse (SI), applies a unit impulse along the common normal between bodies A and B. For SI, the Equation 2.9 is re-written as follows:

$$\Delta \vec{v}_{ca,cb}^n = \frac{\vec{n}}{m_a} + r_{ca} \times \frac{r_{ca} \times \vec{n}}{I_a} - \frac{-\vec{n}}{m_b} + r_{cb} \times \frac{r_{cb} \times \vec{n}}{I_b} \quad (2.11)$$

According to SI, the $M_{effective}$ is written as:

$$M_{effective} = |P| / (\Delta \vec{v}_{ca,cb}^n \cdot \vec{n}) \quad (2.12)$$

Here, the term $|P|$ is normalized as the unit impulse is applied along the common

normal. Likewise, the term $\vec{v}_{correction}$ is the projection along the common normal with a negative sign to result in repulsion rather than attraction. Based on this, one can arrive at the following expression of $P_{correction}$:

$$P_{correction} = \frac{-\vec{n} \cdot \Delta \vec{v}_{ca,cb}}{\vec{n} \cdot \left(\frac{\vec{n}}{m_a} + r_{ca} \times \frac{r_{ca} \times \vec{n}}{I_a} - \frac{-\vec{n}}{m_b} + r_{cb} \times \frac{r_{cb} \times \vec{n}}{I_b} \right)} \quad (2.13)$$

This presents the basic formulation of the Sequential Impulse constraint solver. This expression results in a correction impulse that prevents two bodies from penetrating each other.

2.5 Position Based Methods

While velocity based methods are promising and popularly used in most physics engines, position-based methods have recently gained traction. Differently from the indirect computation of the position using force or velocity, these methods directly control the position of the bodies at successive time-steps. The position-based dynamics was first introduced by [60] and uses constraints projection to alter the positions of bodies. Constraint projection can be done using the change in body velocities and in that case, it resembles velocity based methods.

The original (PBD) method was used for simulation of soft-bodies however, as stated in [60], the method can be adopted to compute rigid body dynamics. The basic formulation of position based methods is as follows:

$$C(p) + \nabla_p C(p) \cdot \Delta p = 0 \quad (2.14a)$$

$$\Delta p = \lambda \nabla_p C(p) \quad (2.14b)$$

The term $C(p)$ is the constraint, Δp is corrected projection and λ is the control parameters. Similar to the velocity based methods, λ is weighted based on the mass of the particles in contact.

$$\Delta p_i = -s w_i \nabla_{p_i} C(p) \quad (2.15a)$$

$$s = \frac{C(p)}{\sum_j w_j |\nabla_{p_j} C(p)|^2} \quad (2.15b)$$

Here the term s is called the scaling term and the weighting terms $w_i = 1/m_i$ are the inverted masses of particles indexed by i .

For the example of collision correction 2.1, a soft-body particle colliding with a rectangular face (v_1, v_2, v_3, v_4) can be formulated as follows:

$$C(p, v_1, v_2, v_3, v_4) = (p - \frac{\sum_{i=1}^4 v_i}{4}) \cdot \frac{(v_2 - v_1) \times (v_3 - v_1)}{|(v_2 - v_1) \times (v_3 - v_1)|} - f_d \quad (2.16)$$

Position-based methods, although mathematically less accurate than velocity or force-based methods, are reliable for extensive soft-body simulations. This is because the constraint equation is directly specified for each particle and then enforced at each dynamic update step. The enforcement of the constraint is called constraint projection and essentially ensures that the particles obey the constraints rigidly instead of applying corrective velocity or force and then iterating until convergence. This results in dense soft-bodies with a multitude of inter collisions being simulated in a stable simulation.

2.6 Indirect Methods

The purpose of indirect methods is to formulate the problem of simulation dynamics into a Linear Complimentary Problem (LCP). A more generalized expression is called Mixed Linear Complimentary Problem (MLCP) which has the following expression:

$$A\lambda - b \geq 0 \quad A \in nxn, \lambda \in nxm, b \in nxm \quad (2.17a)$$

$$\lambda(A\lambda - b) = 0 \quad (2.17b)$$

$$\lambda \geq 0 \quad (2.17c)$$

Here A and b are known and the goal is to solve for λ . Looking at equation 2.10, velocity based methods can easily be expressed as MLCPs by setting $A := (M_{effective})^{-1}$, $\lambda := P_{correction}$ and $b := \Delta \vec{v}_{correction}$. Additionally, there are other ways to specify the A matrix. The Jacobian J_{ab} with the mass matrix and the Collision K method [61] are two such examples and discussed next.

2.6.1 The Combined Jacobian Method

This method computes the combined velocity Jacobian as such:

$$J_{a,b} = -[\vec{n}, r_{ac} \times \vec{n}, \vec{n}, -r_{bc} \times \vec{n}]^T \quad (2.18)$$

Based on this combined Jacobian method, the mass matrix is simply a sparse

matrix consisting of masses and inertia of body A and body B.

$$M = \begin{bmatrix} M_a & 0 & 0 & 0 \\ 0 & I_a & 0 & 0 \\ 0 & 0 & M_b & 0 \\ 0 & 0 & 0 & I_b \end{bmatrix} \quad M \in 12 \times 12 \quad (2.19)$$

Here the upper case M_a and M_b indicate a 3×3 diagonal matrix consisting of m_a and m_b . The terms I_a and I_b can either be the principal moment of inertia in which case the matrix M is diagonal, otherwise it is sparse. This resulting A matrix is simply:

$$A = J_{a,b}^T M^{-1} J_{a,b} \quad (2.20)$$

2.6.2 The Collision K Matrix Method

The collision K matrix methods has a slightly different approach to modeling the A matrix. This method converts the vectors $r_a \vec{c}$ and $r_b \vec{c}$ to skew-symmetric matrices:

$$S(\vec{r}) = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \quad (2.21)$$

Based on the skew-symmetric matrices, the collision matrix K is as follows:

$$K = M_a^{-1} - S(\vec{r}_{ac}) I_a^{-1} S(\vec{r}_{ac}) + M_b^{-1} - S(\vec{r}_{bc}) I_b^{-1} S(\vec{r}_{bc}) \quad (2.22)$$

And the A matrix is simply computed by cross multiplication of the normal n .

$$A = \vec{n} K \vec{n}^T \quad (2.23)$$

It can be seen that the resulting A matrix from both these methods is identical to the direct specification using the sequential impulse method. Formulating the problem is only part of the problem though, the next challenge is to solve the system of equations.

2.7 Solving the Constraints

Having formulated the constraints into a linear complementary problem, iterative techniques such as the Jacobi [62] or Gauss-Seidel (GS) method [63] can be used to solve the system of equations representing the constrained rigid body dynamics. The Gauss-Seidel method is preferred over the Jacobi method as the successive approximations can make use of the updated value of the prior λ_i . Furthermore, since the trivial GS solver does not account for the inequality in the constraint equation, a slight variation, called the Projected Gauss-Seidel, is used. The following algorithm iterates over the λ at each iteration.

```

for  $itr \in N_{iterations}$  do
   $Error := 0$ 
  for  $i = 1$  to  $n$  do
    for  $j = 1$  to  $n$  do
      if  $i \neq j$  then
         $Error = Error[i] + \sum A(i, j)\lambda(j)$ 
      end if
    end for
     $\lambda[i] = \frac{b[i] - Error}{A(i, i)}$ 
     $\lambda[i] = \text{Clamp}(\lambda[i], min, max)$ 
  end for
end for

```

Situations that render the matrix A singular can result in infinite momentum along the singular axes. This is mostly due to numerical approximations, and large mass ratios are a primary contributing factor. There are various ways to prevent

this from happening, the simplest being to explicitly prevent diagonal elements from going to zero. This is one of the features of the Projected Gauss-Seidel solver.

During each update-step, a variable number of sub-iterations (the term sub-iterations is used to distinguish between the use of iterations in later chapters) are required for convergence to a solution. Increasing the number of sub-iterations prolongs the compute time of each update-step. Conversely, the length of the update-step dt exponentially impacts the required number of iterations. This is a major challenge to achieving a real-time dynamic simulation.

2.8 Articulated Body Methods

The formulation of rigid body dynamics using any of the Cartesian state-based methods (force, velocity and position) relies on the maximal coordinates [64]. The use of maximal coordinates generalizes physics simulations that may involve a combination of both free and connected bodies under generic external forces and moments. The simulation of constraints based on these coordinates has an inherent “slop” in them. Slop refers to the softness in constraints based on external forces. This slop occurs due to the way the corrective impulse (discussed in Section 2.4) is applied. Since the impulse is applied in the maximal coordinates, the convergence is required in these coordinates as well.

Robot manipulators consist of linear and rotational joints which are stiff along all axes except the axis of freedom. There are, of course, compound joints that have multi-axes freedom but they can also be modeled using a combination of multiple linear or rotational joints. Rendering the stiffness along the non-free axes is a challenge for maximal coordinate solvers. To mitigate these problems, articulated robots can be represented in terms of reduced coordinates. The Denavit Hartenberg

(DH) [65] parameters used in robotics can easily be used to formulate the reduced coordinates. Using this notation, each body (called a link in reduced coordinates) is represented by only one free variable \vec{q} . This free variable allows either rotation or translation along the axis of freedom. Finally, the 6 DOF position of bodies can be updated by accounting for the relation between \vec{q} and \vec{x} .

The DH formulation can be converted to transformation matrices between a parent and a child body. These transformation matrices are hierarchical which allows the calculation of connected bodies poses in the world frame. The algorithms that treat the dynamic bodies using reduced coordinates are classified as Articulated Body Algorithms (ABAs) or Articulated Body Methods (ABMs) [66] [67]. The reduced coordinate representation mathematically forces the bodies (links) to obey stiff limits along the constrained axes. In Robotics literature, one often uses the Euler-Lagrange or Newton-Euler method to compute the reduced coordinate representation of articulated links.

$$M(\vec{q})\ddot{\vec{q}} + C(\dot{\vec{q}}, \vec{q})\dot{\vec{q}} + G(\vec{q}) = \vec{\tau} + \tau_{ext} \quad (2.24)$$

Here $M(\vec{q})$ is the $n \times n$ Inertia matrix, $C(\dot{\vec{q}}, \vec{q})$ is the Coriolis matrix and $G(\vec{q})$ is the Gravity vector. The terms $\vec{\tau}$ and τ_{ext} can be separated to distinguish between implicit torque and external torques. These terms also show that the input to the system of equations is only via torques applied along the joint axes. The equation can be re-arranged to calculate $\ddot{\vec{q}}$ as:

$$\ddot{\vec{q}} = \frac{\vec{\tau} + \tau_{ext} - (C(\dot{\vec{q}}, \vec{q})\dot{\vec{q}} + G(\vec{q}))}{M(\vec{q})^{-1}} \quad (2.25)$$

The resulting joint accelerations can be converted to Cartesian accelerations to solve for body positions back in maximal coordinates. Using ABMs for articulated

robot manipulators is preferable over maximal coordinate SI solvers. Some popular libraries that use this reduced coordinate formulation include RBDL [68], Simbody [36] and Dart [35].

2.9 Discussion

The mathematical formulation presented in the prior sections can be summed up as follows. Of the three maximal coordinate-based methods, velocity-based methods perform better than force-based methods in terms of stability and are more accurate than position-based dynamics methods. For articulated bodies, ABAs (utilizing the reduced coordinates) outperform velocity based methods both in terms of accuracy, and stability.

However, all is not well with using purely reduced coordinates solvers. Free bodies that can move along 6 DOF is a simple example which needs maximal representation. More pertinent to this discussion are surgical robots, which often employ closed-loop bodies for constrained motions. These robots cannot be generally specified using ABAs. While there are examples of closed-loop robots using reduced coordinate solvers, these are not generalizeable implementations.

The point of this discussion is to emphasize that velocity-based methods (SI solvers) are the better on average in terms of stability, accuracy and generalizability when compared to other methods. Due to this reason, they can be used to specify any type of environment. For articulated robots, the softness at the joints can be alleviated by allowing a greater number of sub-iterations, reducing the length of the update-step, and more importantly, making sure that two inter-constrained bodies do not have an excessively large mass ratio (Equation 2.19). While these are among several factors that have been discussed throughout this chapter for improving the

stability and accuracy of the physics simulation, the most recurring factor is the length of the update-step dt . Chapter 3 presents an extensive discussion on this variable and the associated repercussions of controlling this variable in the pursuit of real-time dynamic simulation.

Chapter 3

Asynchronous Framework for Collaborative Interaction

This chapter focuses on the challenges of achieving a real-time dynamic simulation with the inclusion of multiple input devices and communication interfaces for control via external applications (Section 3.4.1). To address these challenges, a new framework is proposed which includes a variable number of input devices while maintaining a real-time dynamic simulation (Section 3.4.2). The inclusion of multiple input devices presents additional challenges such as common frame representation and contextual control in a specific frame of reference. This is addressed by the development of a set of 17 frames that are used to abstract the mapping of any device and its corresponding simulated body (Section 3.5). Next, the design of distributed and asynchronous communication pipeline is discussed which leverages the real-time dynamic simulation (Section 3.7.1). To utilize these communication interfaces, a complementary Python client is presented in Section 3.8. Finally, the results and conclusions are presented in Chapter 3.9.

3.1 Published Work

Some of the work presented in this chapter has been published as:

Munawar A, Fischer G, “*An Asynchronous Multi-Body Simulation Framework for Real-Time Dynamics, Haptics and Learning with Application to Surgical Robots*”, Intelligent Robots and Systems (IROS), Macao, China, 2019.

3.2 Introduction

The use of partial autonomy for performing manual tasks with robot collaboration is a popular research area [69]. This area traditionally includes humanoids or nursing robots operating alongside humans to assist in certain tasks. Current research focusing on improving the outcome of robot-assisted surgery, although different in some aspects, has many parallels to traditional human-robot synergy as they both involve collaborative assistance. Concerning robotic surgery, such assistance has a vast scope, which includes, for example, the autonomous positioning of slave manipulators for better workspace, autonomous endoscopic control for improved field of view and visual feedback using markers and visual queues. These examples are forms of assistance that do not directly impact the operator’s control input. On the other hand, examples affecting the user’s control input include active haptic feedback, virtual fixtures for guidance/boundary enforcement and control of a sub-set of slave manipulators in coordination with the human operator. The example of coordinated control is particularly interesting as it is a function of the surgeon’s control input, the sub-task at hand and may require some form of training through deterministic controllers and/or machine-learning. This collaboration may not only affect the teleoperated slave manipulator but also impart forces on the operator’s hand in addition to the feedback from interaction with the real/virtual environment.

The unit carrying out the collaborative assistance can be called an Intelligent Agent with the understanding that the agent can either be a software algorithm trained via neural networks (NN) or another human. In case the agent is a deterministic controller, the term “Intelligence” loses meaning and the agent can instead be called an external or a distributed controller. This chapter focuses on the design and implementation of a framework to achieve assistive collaboration by providing the means to integrate modern surgical robots/haptics devices with high fidelity asynchronous control and haptic feedback. The conceptual design of the framework involves the segregation of the input interfaces into the basic building blocks which can be combined or taken apart to let Intelligent Agents and distributed controllers share the control with primary operators in complex and extensive physics-based simulations.

Real-time simulators that can support collaborative assistance using physical input interface devices (IIDs) can play an important role in improving the outcome of surgical, as well as non-surgical training tasks. The requirements of such simulators have been discussed in Section 1.3. This chapter deals with two of the fundamental requirements, namely, the implementation of real-time dynamic simulation and generic yet extensive device control interfaces for allowing multi-device input. Regarding the simulator requirements, an additional point is addressed in this chapter which is the support for machine learning algorithms through an extensive and distributed communication pipeline. This discussion includes the motivation behind the choice of the underlying message payloads.

While several open-source simulators for robot dynamics such as Gazebo [32], VRep [70] and MuJoCo support different feature-sets ranging from distributed controllers to ML and RL support, these are not built for real-time training applications with support for various haptic and non-haptic IIDs. Among other factors, this is

primarily due to the implementation approach which is “sequential” in nature. As the name indicates, the “sequential approach” performs the internal sub-routines and external methods-calls for supported plugins one after the other. This approach works well for non-real-time tasks that are done purely in simulation. Moreover, this reduces implementation complexity (e.g. since program execution is sequential and deterministic), improves maintainability (e.g. individual components can be blocked or loaded for debugging as their methods are invoked from one point in the sequential code) and allows relatively modular feature expansion through future updates (due to less “moving parts” that each component depends upon).

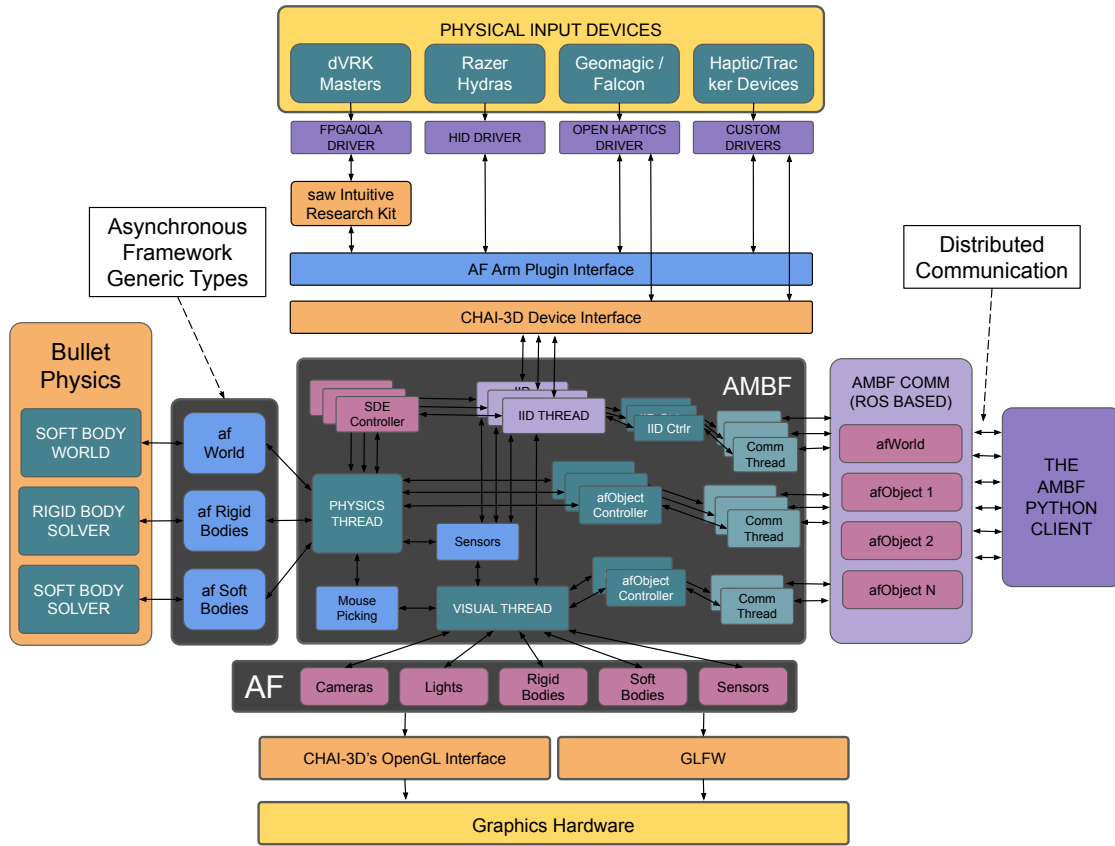


Figure 3.1: A conceptual view of the Asynchronous Framework

On the other hand, a parallel design for heterogeneous simulators (support both physical and simulated input devices and environments) allows the flexibility for

achieving real-time dynamic simulation through asynchronous updates, albeit with the complexity of implementation that spans many different levels. These levels include the lowest building blocks for software development such as data integrity and management, and also the higher levels such as API specification, and feature expansion.

To develop such a heterogeneous simulator, these challenges first had to be understood, as there is limited prior work in this respect. Some of the challenges were harder to identify than others and some were in-fact understood alongside the implementation process. The final outcome resulted in the framework that allows a variable number of IIDs to be included in the simulation with real-time device updates as well as real-time physics simulation updates. The framework is called the Asynchronous Framework (AF) and the underlying simulator that utilizes this framework is conveniently called the Asynchronous Multi-Body Framework (AMBF). This framework abstracts physical devices, simulated bodies, intelligent agents and distributed controllers into independent asynchronous objects that are then handled in a parallel fashion. A simplified description of the Asynchronous Framework is shown in Figure 3.1. This Chapter is dedicated to the challenges faced in the development of this framework in light of the limitations of existing implementations.

3.3 Selection of Software Components

The control of multiple input interface devices alongside a simulated dynamic environment is an essential requirement of the proposed Asynchronous Framework. Section 3.4.2 discusses the challenges associated with this multi-device control. The second, but equally important requirement is the selection of the appropriate soft-

ware components. The selection of these software components is driven by the compatibility in our use-case (CISST-SAW and dVRK – an open-source research kit based on the clinical da Vinci surgical robot – [71] [20]) and the popularity and adoption of the components in the research community. As such, some of the important components which have been chosen to complement the Asynchronous Framework include CHAI-3D [72], Bullet Physics[34], GLFW [73], Boost [74], Yaml-cpp [75], Yaml-py [76], Keras [77], Keras-RL [78], and Open-AI’s GYM [79]. Figure 3.2 shows a holistic view of the appropriate place these external components hold in the pipeline leading from the hardware components to the Python libraries for ML and RL.

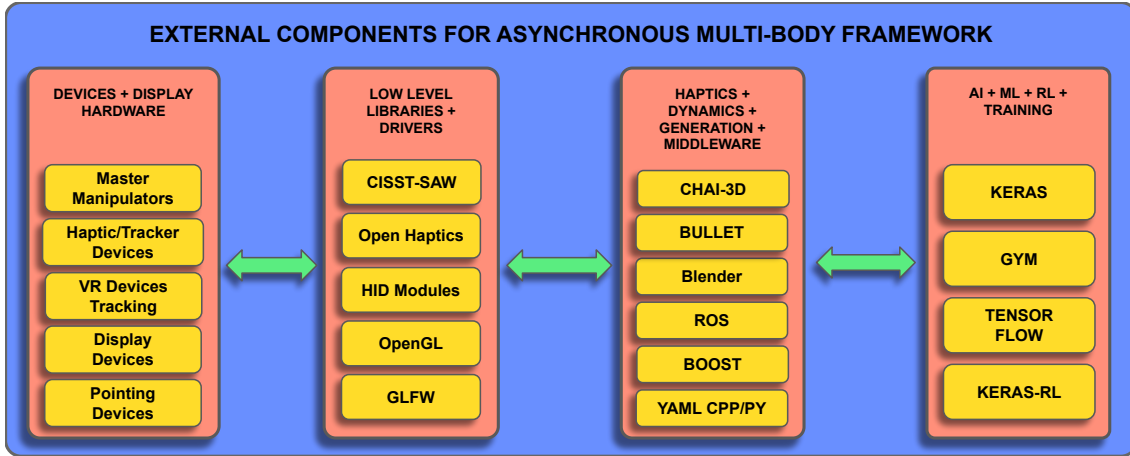


Figure 3.2: The external components that have been selected to complement various part of the Asynchronous Framework and the AMBF.

The motivation behind the selection of each component emphasizes not only the design philosophy of the Asynchronous Framework but also on the process of breaking down a heterogeneous simulator into basic building blocks. In this regard, Bullet Physics[34] and CHAI-3D [72] are the two integral components. Bullet’s Physics is already used in some open-source robotic simulators, including Gazebo - the preferred dynamic simulator for the Robot Operating System (ROS). While Open Dynamics Engine (ODE) is another competitive physics library, Bullet pro-

vides a built-in and general-purpose collision detection library and the support for many different kinds of constraint solvers.

CHAI-3D is an open-source library that supports several commercial haptic devices and offers a device-agnostic interface to applications rendered in Open-GL [80]. CHAI-3D lacks a built-in physics computation library but has preliminary support for Bullet and ODE in the form of basic demo applications. These demo applications show the proof of the concept of integrating with CHAI-3D rather than fully functional support. Yaml-cpp and Yaml-py are essential to the Asynchronous framework but are used for meta-data specification and retrieval and discussed in that context in Chapter 4.

Keras [77] was chosen because of its compatibility with currently popular NN, ML and RL libraries which include Tensorflow, Theano, Keras-RL and OpenAI’s GYM. Both Tensorflow and Theano can be used for defining and solving neural networks. However, neither of them is directly related to the Asynchronous Framework since they are used indirectly through Keras interfaces. Keras-RL [78] provides the implementations of various Reinforcement Learning algorithms and newer algorithms are continuously being added. OpenAI’s GYM allows for the creation of environments and agents that expose an action-state interface for input and output and is the default front-end for utilizing Keras-RL (and consequently Keras).

3.4 Implementation Details

3.4.1 Implementation of Real-Time Dynamic Simulation

Before delving into the implementation of a real-time dynamic (physics) simulation, it is important to define the meaning of the term. A real-time physics simulation means that the simulation clock tracks stepping of the real-world clock. This real-

time clock tracking enables consistent motions of simulated bodies based on contacts, constraints and environmental forces. Lack of a real-time simulation means that some parts of the simulation will be faster than others. As a simplified example, simulated bodies traversing along a trajectory with constant velocity will appear faster at some times and slower at others. For simulations that involve human input for learning and training, real-time simulation is important as humans expect consistent interaction with everyday objects. The response due to interaction doesn't need to be exactly similar to real-world bodies, but it should be consistent within the simulation concerning time. On the contrary, for simulations that require automated training, the real-time simulation doesn't hold much value as the simulation is often sped up for accelerated training. The mismatch between the simulation and the real-world time, in that case, can be adjusted by using simulated time-stamps that also store the actual value of the real clock.

A fundamental control parameter associated with a physics simulation is the update-step. An update-step is the window of time between two discrete states (t_n and t_{n+1}) of the simulated physics and is measured as $dt = t_{n+1} - t_n$. Essentially, at each step of the physics simulation, a new dt is provided to the underlying solver which increments the time of the physics simulation and tries to solve the system of equations. As discussed in the formulation of the system of equations in Chapter 2, a solver of the form of (Guass-Seidel or Jacobi) is used to successively iterate (called sub-stepping, sub-steps or simply sub-iterations) the states of the bodies in the simulation. The successive approximation to the system of equations tries to update the solution such that the residual error between successive states falls below a certain threshold ϵ . Each sub-step takes a certain amount of CPU time (and thus real-world time). As a result, a larger dt , requires more sub-iterations and thus more time to compute. Most offline and simple robot simulators can utilize a fixed time-

step, which allows for stable performance of the underlying simulated dynamics by allowing adequate time for computation of constraints and collisions.

Based on the aforementioned process of solving simulated dynamics, achieving a generic real-time physics simulation is challenging. One major factor is the uncertainty in the amount of time consumed by sequential processes that are sandwiched between each update-step. Ideally, one wants to minimize these time-consuming processes, which requires the understanding - both conceptually and empirically - of factors that take up the most time during each update-step. That being said, not all processes can be eliminated or isolated. Processes such as contact computation, constraint solving and external force resolution are inherent to the physics solver. Of these inherent processes, contact and collision computation can become a major time-consumer. This is where the use of collision primitives comes into play. As such, relatively advanced shapes can be created using a compound of collision primitives (implicit collision). Implicit collision computation is significantly faster and more reliable than explicit collision techniques (GJK [81] and Minkowski Difference [82]), especially for relatively lower update-frequencies of the physics simulation. Some corresponding results regarding this are shown in Chapter 6. While collision primitives are computationally faster, creating collision primitives can be infeasible for complex shapes and instead mesh decimation techniques may be preferred. Examples of external time-consuming processes include blocking delays caused by device drivers, a large number of transform operations, loading plugins and performing plugin method calls.

The Asynchronous Framework achieves real-time physics simulation by dynamically changing more than just the magnitude of the time-step. A non-real-time simulation is also conveniently possible but it is not the focus of this discussion. Each update step has 3 control parameters which include the magnitude of the time-step

δt , maximum number of iterations N_{max} and the magnitude of the default integration step δt_i . Usually, the default integration step δt_i is fixed, and the time-step δt and N_{max} are dynamically controlled. Moreover, Bullet Physics uses an interface called **Motion States**, which allows access to the body states in-directly. This interface saves computational time by interpolating states between the update-steps rather than stepping the solver in case δt meets the following in-equality:

$$\delta t < \delta t_i \times N \quad ; \quad N \in \mathbb{Z}^+ \quad \& \quad N \leq N_{max} \quad (3.1)$$

In the equation above $N \in \mathbb{Z}^+$ stresses that N is a positive integer. Ideally, δt should not exceed $\delta t_i N_{max}$, but this condition can easily be violated, thus, N_{max} needs to be updated accordingly. However, increasing N_{max} also increases the computational time thereby causing higher values of N_{max} in successive iterations. This tends to cause circular deterioration. Based on the empirical evaluation, update frequencies lower than $45Hz$ tend to have a noticeable impact on solution convergence. Finding the right balance between N_{max} and δt is challenging. To mitigate this limitation, N_{max} has to be capped to an upper limit.

The dynamic control of more than just the update step allows the Asynchronous Framework to be more flexible in achieving real-time physics as compared to other robot dynamic simulators. This, however, is just one of the many factors the makes Asynchronous Framework more robust. The other factors include the mitigation of external factors that consume time in between physics update steps. These factors are discussed in Section 3.4.2. Figure 3.3 shows the difference in slippage of the simulation time from the system time (Wall Clock) based on a comparison between fixed and dynamic time-stepping for a simple peg and hole task using one input device.

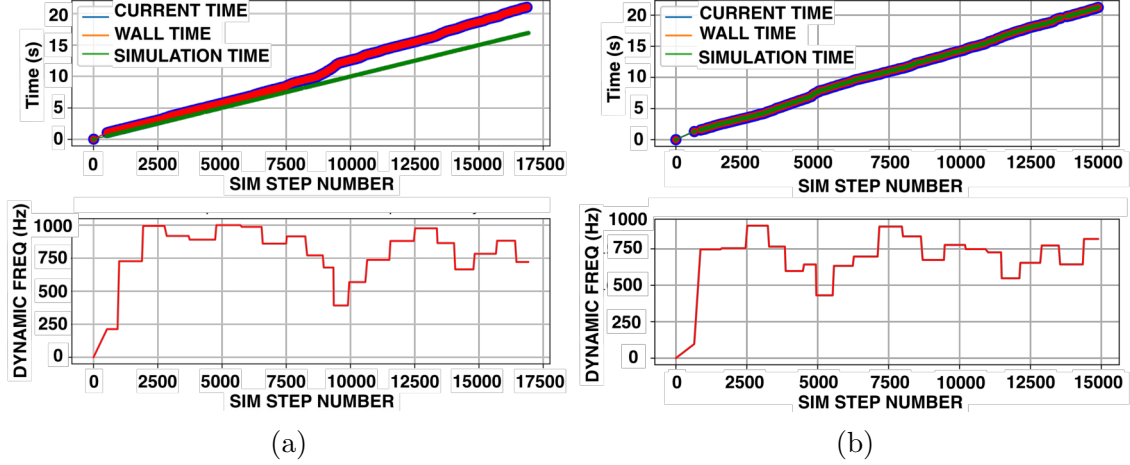


Figure 3.3: (a) Time dilation between Application Clock & Simulation Clock using fixed time-step ($dt=0.001$) (b) Time tracking between Application Clock & Simulation Clock using dynamic time-step

3.4.2 Asynchronous Control of Multiple IIDs

Any input device (with more than 2 degrees of freedom), haptic or not can be called an Input Interface Device (IID). Devices with 2 DOF, for example, computer mice, are not characterized as IIDs. Each IID is represented by a simulated dynamic end-effector (SDE) comprising of links and joints. The term “root link” is a link belonging to the SDE and refers to the base link (body) which is bound to the IID with some transform mapping. In many cases, the root link does not have any parents but can have child bodies such as fingers and child joints. The condition of a parent-less root link is relaxed later in this chapter to present a generic interface for binding any simulated body and its successors to an IID.

For multi-manual interactive collaboration, interfacing multiple IIDs with the physics simulation is required. Each IID needs a recommended update-rate for reading the state data for its device drivers. For haptic IIDs, an update rate of at least a 1 kHz [83] is preferred for reading and writing data (in addition to minimal communication latency). It can easily be seen that a “sequential implementation”, adopted by existing simulators, can only execute the associated device methods in between

each dynamic simulation step. The device drivers for several commercial devices – Geomagic Phantom/Touch (*3D Systems Corp, Rock Hill, SC, USA*) and Falcon (*Novint Technologies Inc., NY, USA*) – impose a blocking delay while commanding forces which further restricts the update-rate of the physics solver. Tracker devices such as Razer Hydra (*Razer Inc., CA, USA*), usually operate at lower update-rates ($\leq 400Hz$) and pointing devices, such as 3D Connexion’s Spacenav mouse¹, operate at even lower frequencies ($\leq 100Hz$). All these devices are shown in Figure 3.4.

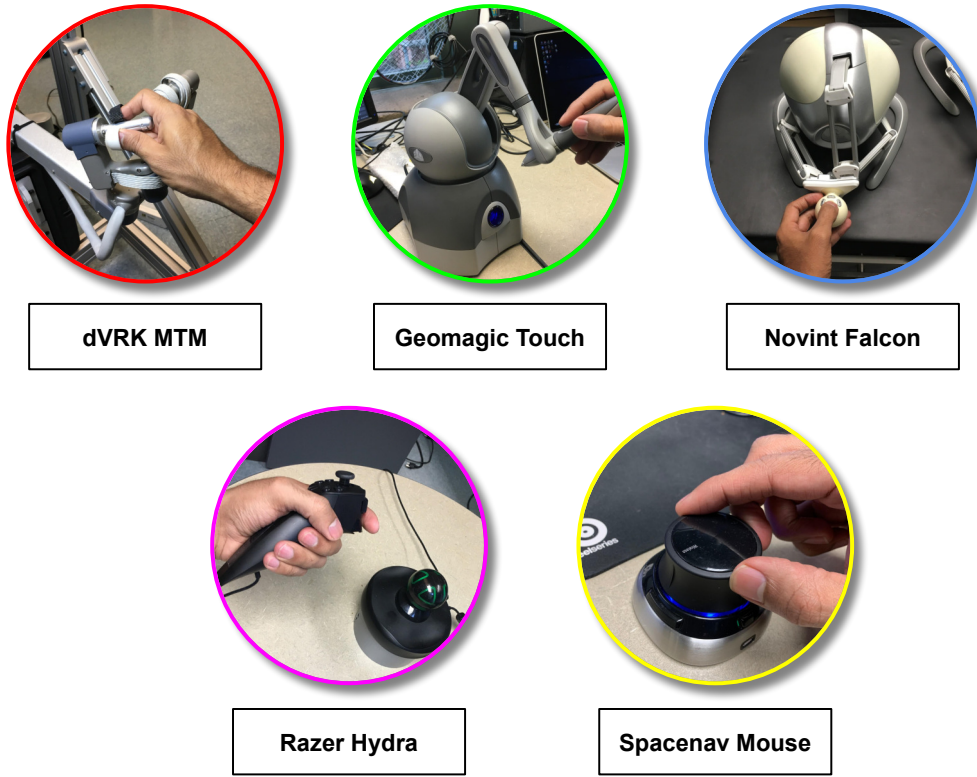


Figure 3.4: Input devices to interact with a dynamic simulation.

To improve the update rate of the physics simulation, a partially distributed implementation was tested. According to this implementation, the control laws for the haptics devices were computed in the simulation loop but the commands were withheld (to prevent blocking sub-routines), and instead, were applied concurrently

¹https://www.3dconnexion.com/spacemouse_compact/en/

in separate threads. This improved the simulation’s update frequency but resulted in unstable control of the SDEs. This was possibly due to a non-deterministic delay between the computation of control laws and the application of output forces as the simulation loop inadvertently swayed back and forth based on the complexity of the underlying physics. The problem was further aggravated when mixing devices with different update-rates. It was concluded, therefore, that a “sequential approach” makes the inclusion of multiple IIDs extremely difficult if not impossible. As a result, an Asynchronous Control scheme was implemented, wherein the dynamic update-loop runs in a separate thread and all of the device update-loops (haptic and non-haptic), in separate threads with indigenous control laws. This control scheme is the basis of the Asynchronous Framework.

An implementation where the IIDs uniquely control their corresponding SDEs is rather straight forward. In such a scenario, the only exchange of data happens between each physical device and its SDEs root link. This exchange can be managed by using mutual exclusion (mutex) [84] locks to prevent race conditions. However, this is not the case with the Asynchronous Framework as data from each thread has to be shared between multiple different threads running at variable frequencies. These threads include the graphics threads, multiple IIDs and communication threads. This is discussed in detail in Section 3.5.

The root link of the SDE is controlled using a dynamic control law based on the motions of the IID. Usually, the states of IID are in the reference frames of the devices themselves, while the SDEs states are in the simulation world frame. Therefore, before the computation of the control law, states have to be converted to a common frame (usually the simulation world frame). Afterward, the control law is used to compute output commands that are then multiplied by two different sets of gains. One set of gains is for scaling the wrench for SDEs and can be called

“controller gains”, while the other set of gains is for controlling the force feedback on the IID and can be called the “haptic gains”. To achieve a truly asynchronous setup, each IID owns a shared data-structure that allows for asynchronous reads and writes. This data-structure maintains the device’s states and has fields to store the commanded forces. A similar, but non-identical, data-structure is defined for each SDE. The control laws are computed and executed independently in the dynamic and haptic threads and applied to the SDE and the IID respectively. A simplified block diagram representing this control scheme is shown in Figure 3.5.

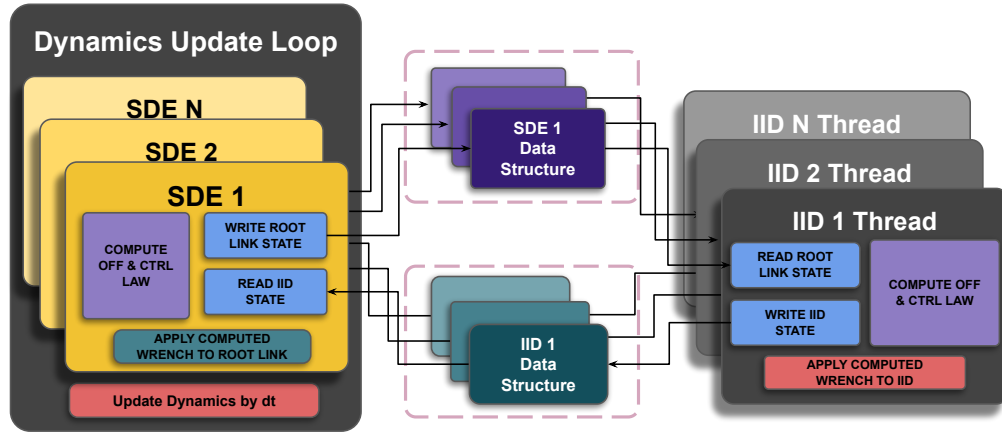


Figure 3.5: A block diagram depicting the Design of Asynchronous Control Scheme, the Simulated end-effectors and Devices maintain independent and mutually exclusive Data Structures (DS) that are updated on successive writes and are capable of asynchronous reads

Such indigenous controllers are used because the execution counters of each thread are different and the commands calculated in one thread do not reflect the state of the encapsulated object (IID or SDE) in another thread. The way a physics simulation is set up, the commands for the simulated bodies are appended as external wrenches. These external wrenches are mixed with wrenches from internal joints and collision constraints. The storage of current states outside the haptic/dynamic update-loops serves as “set-points”. These asynchronously accessible “set-points” allow for the instantaneous computation of control laws and application of commands,

only once per simulation step, which prevents force saturation and instability of the SDEs.

Both the SDE and the IID data-structures own indigenous communications instances that allow input via external controllers and intelligent agents. The SDE is constrained due to its connection with dynamic simulation solver while the IID to its device drivers, however, they can both be commanded asynchronously using the communication pipeline.

3.4.3 Contextual Viewport Control

Distinctive to the Multi-User control, different users may require different point-of-views of the same underlying simulation. Moreover, users might want to control their view direction and position. Thus in terms of visual elements, multiple cameras can be defined and shared among different devices. What this translates to is that an IID device can take exclusive control of multiple cameras (view-ports), share them with other IIDs or even retain exclusive control of some cameras while sharing control of others. Hand-eye coordination requires the definition of several transforms, some of which include the camera pose, IIDs current pose, offset-transforms for clutch engaging/disengaging, mapping transforms between the user and camera and motion scaling. An extensive list of such transformations, in the context of multi-lateral control, is presented in Table 3.1 and discussed further in Section 3.5. The associated frames are described in Table 3.3.

The visualization loops run much slower than the device and simulation update loops and direct data exchange causes concurrency issues. Due to a large number of transforms involved (discussed in detail in Section 3.5), a race-condition is undesirable. This has been addressed, in somewhat a similar manner as the SDEs and IIDs, by utilizing independent copies of a shared data-structure representing

the state of each camera. One such copy encapsulates the communication pipeline which allows the camera control externally. Other copies are paired with each IID and SDE pair that request access to the specific camera. Activating camera position control from any copy of the camera’s data-structure bypasses the other copies of the corresponding camera from being commanded. At the same time, the other copies can actively retrieve the camera’s state to make sure that the hand-eye coordination is not only smooth but resumes from the latest valid pose. This is a unique feature of the Asynchronous Framework.

3.5 Minimal Frame Representation for Input Mapping

The teleoperated control of SDEs using IIDs in a fixed frame of reference (FoR) is rather trivial. The set of equations for a generalized representation for such cases is shown below. Essentially, the linear velocities of the IIDs are mapped based on the fixed orientation offsets for computing the scaled linear velocities for the SDEs, and vice-versa. In terms of orientation control, the use of angular velocities is usually avoided. Instead, the orientation is directly mapped between the IID and SDE with appropriate offset rotation matrices (both pre and post multiplied) to adjust between different FoRs. Contrary to a generalized representation, the similarity transforms are often simplified in terms of raw joint angles and angular offsets, specialized for specific devices. This is done to simplify the underlying complexity associated with similarity transforms. To develop a generic device handling implementation with different types of input devices (custom base transforms and tip rotation offset for SDEs) and different teleoperated output bodies (varying FoRs and Tip Offset frames) such simplifications cannot be applied.

Figure 3.6 shows two IIDs controlling their proxy SDEs. Similar to the haptic IIDs, the SDEs have inertial properties and require dynamic control laws. As discussed in Section 3.4.2, independent control laws are used for the SDE and the IID. Both the control laws produce a 6 DOF Cartesian wrench $\vec{F} = [\vec{f}_{[SDE, IID]}, \vec{\eta}_{[SDE, IID]}]^T$, where:

$$\Delta \vec{P} = (\vec{P}_{[SDE, IID]}^W - (R_{OM}^{[SDE, IID]})^T * \vec{P}_{reference}) \quad (3.2)$$

$$\Delta R = R_{OM}^{[SDE, IID]} * (R_{[SDE, IID]}^W)^T * R_{reference} * (R_{OM}^{[SDE, IID]})^T \quad (3.3)$$

$$[\vec{a}\vec{x}_e, \Delta\theta] = ToAxisAngle(\Delta R_n) \quad (3.4)$$

$$\Delta^2 \vec{\theta}_e = ((\vec{a}\vec{x}_e)_n * (\Delta\theta)_n) - ((\vec{a}\vec{x})_{n-1} * (\Delta\theta)_{n-1}) \quad (3.5)$$

$$\vec{f}_{[SDE, IID]} = \vec{K}_L * \Delta \vec{P}_n * t_s + \vec{B}_L * (\Delta \vec{P}_n - \Delta \vec{P}_{n-1}) / dt \quad (3.6)$$

$$\vec{\eta}_{[SDE, IID]} = R_{[SDE, IID]}^W * (\vec{K}_A * (\vec{a}\vec{x}_e * \Delta\theta) + \vec{B}_A * \Delta^2 \vec{\theta}_e / dt) \quad (3.7)$$

Here \vec{f} and $\vec{\eta}$ are the force and torque, while \vec{K} and \vec{B} are Stiffness and Damping gains. As discussed in Section 3.4.2 different sets of gains are used for the SDE and the corresponding haptic IID. The term $t_s = \frac{dt_f}{dt_d}$ scales the time-step for asynchronous control by taking the fraction of fixed parametric time-step (dt_f) by the dynamic time-step (dt_d). This scaling term is only used in the SDE's control law. It is often the case that $t_s = 1$ (such as the dynamic simulation running at intended

speed), however, for significantly lower update frequencies the time step scaling prevents the saturation of external forces on the SDEs. The term R_{MO} is the mapping offset for the SDE and the IID, while the terms $P_{reference}$ and $R_{reference}$ are the reference values for the control laws. It is the calculation of these reference quantities which allows a flexible control interface by mixing cameras, SDEs and IIDs. The multiple different control loops and contexts in which these reference values are calculated is the basis of the distributed control formulation in the Asynchronous Framework.

A basic set of coordinate frames have been identified that generalize the representation of SDEs, IIDs and Cameras for teleoperated and collaborative control. The underlying transformation matrices, representing these frames, can handle the change in direction of commands based on the change in the cameras transform. Further, these transformations can handle the complexity involved with the shared device and shared view-port control. The use-cases that motivate this extensive frame representation are discussed below:

- **Multi View-Port Control:**

In the context of hand-eye coordination, the control of an IID is usually w.r.t. to a camera's (view-ports) FoR. To make the implementation more general, multiple IIDs can share a camera between themselves. Sharing in this context means that the IIDs are controlled in the FoR of the camera and also control the camera itself using the appropriate switching mechanism. The change in camera transform resulting from one IID should be reflected in the control of the camera sharing IIDs. A slightly similar implementation is used in

Table 3.1: The Description of Transformation matrices used for the XVII Representation

Idx.	Expression	Description
1	T_{IID-B}^U	Input Interface Device's (IID) Base Frame in Users Frame of Reference
2	T_{IID-E}^{IID-B}	IID's End-Effector Frame in IID's Base Frame
3	T_{BO}^{IID-B}	Base offset Frame for IID's end effector, expressed in IID's base Frame. This is used to provide base offset between the corrected IID Frame and the simulation world frame
4	T_{TO}^{IID-E}	IID's Tip Offset Frame in IID's End Effector Frame. This is used to provide orientation offset between the corrected IID Frame and the simulation world Frame
5	T_{IID-CL}^{S-W}	Clutch Frame of IID that moves with the IID when the clutch button is pressed. This is expressed in the Simulation World's Frame.
6	$T_{IID-PRE-CL}^{S-W}$	Clutch Frame of IID prior to the clutch button press. After the button press, the IID-CL Frame moves with the IID
7	T_{IID}^{S-W}	IID Expressed in the S-W Frame after all the IID offset and similarity transforms have been applied
8	T_{S-W}^U	Simulation world Frame in user's eye Frame
9	T_{SDE}^{S-W}	Current SDE Frame in S-W Frame
10	T_{SDE-B0}^{S-W}	Frame used by the IID to impart base offset to the SDE. Pre-multiplied to SDE Frame. Useful in defining an initial offset between the IID and the SDE
11	T_{SDE-TO}^{S-W}	Frame used by the IID to impart tip offset, mostly rotational, to the SDE. Post multiplied to the SDE and is useful in assigning different orientation mapping between the IID and the SDE
12	$T_{SDE-REF}^{S-W}$	Reference SDE frame expressed in S-W Frame. Used to compute the control laws for the SDE
13	$T_{SDE-REF-O}^{S-W}$	Origin Reference SDE frame expressed in S-W Frame. Used to anchor the $F_{SDE-REF}$ as multiple IIDs and Cameras can share the SDE.
14	T_{CAM}^{S-W}	Camera Frame in S-W Frame
15	T_{CAM-CL}^{S-W}	Clutched Camera Transform. This transform only moves with the movement of the IID if the clutch button on the IID is pressed.
16	$T_{CAM-PRE-CL}^{S-W}$	Camera Clutch Transform prior to pressing Clutch, as after pressing the camera clutch, the CAM-CL Frame moves with the IID

Table 3.3: The Description of Frames used for Minimal Frame Representation. These Frames are shown in Figures 3.9, 3.11, 3.10, 3.12

Idx.	Frame	Description
1	$S - W$	Simulation world frame
2	U	User's frame
3	IID	Input Interface Device's (IID) frame
4	$IID - B$	IID's base frame
5	$IID - E$	IID's end effector frame
6	$IID - BO$	IID's base offset frame.
7	$IID - TO$	IID's tip offset frame
8	$IID - CL$	IID's clutched frame
9	$IID - PRE - CL$	IID's pre-clutch frame
10	SDE	Simulated Dynamic End-Effector (SDE) frame
11	$SDE - BO$	SDE's base offset frame
12	$SDE - TO$	SDE's tip offset frame
13	$SDE - REF$	SDE's reference frame
14	$SDE - REF - O$	SDE's reference's origin frame.
15	CAM	Camera's frame
16	$CAM - CL$	Camera's clutch frame
17	$CAM - PRE - CL$	Camera's pre-clutch frame

multi-surgeon robot-assisted surgical procedures that use two da Vinci Master Consoles. In such implementations, a secondary operator (surgical assistant) may take control of the endoscopic camera using a separate set of IIDs. This can be generalized further by allowing any IID to control more than just a single camera with different initial positions, although the IID itself can only be controlled in only one of the camera’s FoR.

- **Multiple IIDs sharing the Control of an SDE**

Another proposed addition to further generalize the shared multi-device control is allowing the possibility of multiple IIDs to share a single SDE. Such a scenario might seem unlikely in the conventional teleoperation sense, however, such scenarios are useful in the context of shared control and autonomy. The user-study presented in Chapter 7 is one such example. Essentially, the applications involving the coordinated control of a single simulated end-effector can be used for supervisory control. This supervisory control can be applied for both teaching and training applications where the supervisor can be a secondary user, a machine learning agent or a deterministic controller interacting in parallel with the primary user. Such a control scenario is more challenging to implement as compared to **Multi View-Port Control** and naturally the combined implementation is even more challenging. One major complication associated with such an implementation is allowing the IIDs to independently switch the control modes (using clutch buttons on the IID for position or camera clutch). The switching essentially disengages the camera or the SDE for the corresponding user, however, the other users are still able to control, clutch and feel the force-feedback without any discontinuity.

The earlier limitation to the binding of the “root link” of an SDE to an IID can

Table 3.5: Equations for Multi-Lateral Control of Camera/SDEs with IIDs using XVII Representation. These equations are indexed according to the Flowchart 3.13

No.	Equation
1	$P_{CAM}^{S-W} = P_{CAM}^{S-W} + S_{CAM} * R_{CAM}^{S-W} * \delta P_{IID}^{S-W} / dt$
2	$R_{CAM}^W = R_{CAM-PRE-CL}^{S-W} * (R_{IID-PRE-CL}^{S-W})^T * R_{IID}^{S-W}$
3	$R_{CAM-PRE-CL}^{S-W} = R_{CAM}^{S-W}$
4	$R_{IID-PRE-CL}^{S-W} = R_{IID}^{S-W}$
5	$P_{IID-CL}^{S-W} = P_{IID}^{S-W}$
6	$R_{IID-CL}^{S-W} = R_{IID}^{S-W}$
7	$P_{SDE-REF-O}^{S-W} = P_{SDE-REF}^{S-W} * (1/S_{IID-WS})$
8	$R_{SDE-REF-O}^{S-W} = R_{SDE-REF}^{S-W}$
9	$P_{SDE-REF}^{S-W} = S_{IID-WS} * (P_{SDE-REF-O}^{S-W} + R_{CAM}^{S-W} * (P_{IID}^{S-W} - P_{IID-CL}^{S-W}))$
10	$R_{SDE-REF}^{S-W} = R_{SDE-REF-O}^{S-W} * R_{CAM}^{S-W} * (R_{IID-CL}^{S-W})^T * R_{IID}^{S-W} * (R_{CAM}^{S-W})^T$
11	$R_{SDE-REF}^{S-W} = R_{IID-BO} * R_{IDD}^{S-W} * R_{IID-TO}$
12	$\Delta P_{IID-(n)} = P_{SDE-REF}^{S-W} - P_{SDE}^{S-W}$
13	$\delta^2 P_{IID-(n)} = (\Delta P_{IID-(n)} - \Delta P_{IID-(n-1)}) / dt$
14	$\vec{F}_{IID}^{S-W} = K_{IID-L} * \Delta P_{IID-(n)} + B_{IID-L} * \delta^2 P_{IID-(n)}$
15	$\Delta R_{IID} = (R_{SDE}^{S-W})^T * R_{SDE-REF}^{S-W}$
16	$\vec{\tau}_{IID}^{S-W} = K_{IID-A} * ToAxisAngle(\Delta R_{IID})$

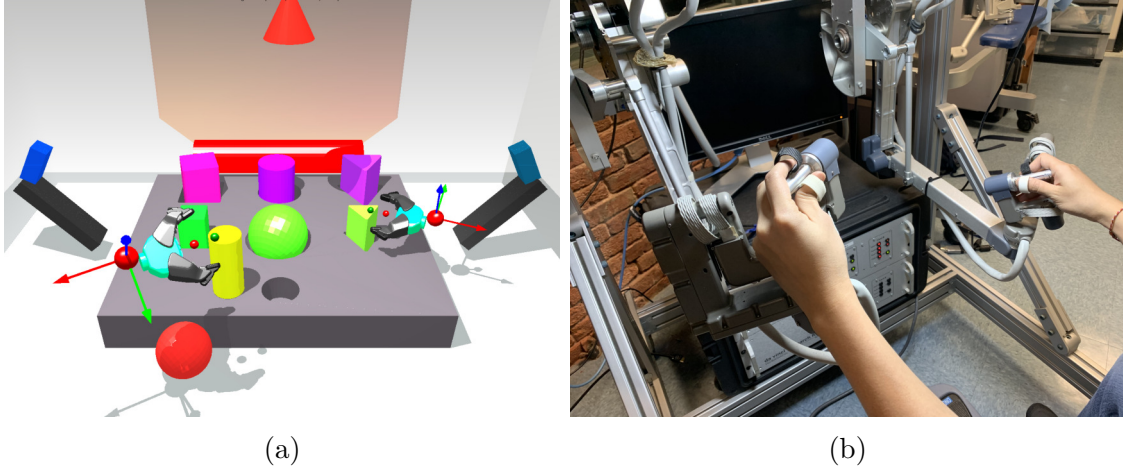


Figure 3.6: These figures show the simulated end-effectors controlled by dVRK Master with clutch/camera foot-pedals enabled. The clutch is used to move the haptic device disengaged, and the camera foot-pedal is used to re-orient the view-direction without affecting the end-effector.

also be relaxed such that any dynamic body, parent-less or not, can be paired to an IID. There is not just a theoretical requirement but in fact there is a use-case for this, which is the pairing of distal bodies (end-effector/graspers) of dexterous robots (such as the PR2 and da Vinci PSMs) to an IID. The control of all the bodies and joints after the root link is handled by the IID whereas the prior bodies and links are controlled independently. The use of the term “independently” could mean either through inverse kinematics, reactive or dynamic controllers. This relaxation also allows the possibility of a chain of connected bodies to be controlled at different points by different IIDs. In the case of humanoid robots, for instance, this can translate to an IID controlling the elbow body while another IID can control the wrist and fingers. Figure 3.7 illustrates a flowchart which is used in the Asynchronous Framework for the selection of an SDE for each IID. The fields are specified in the configuration file shown in Figure 3.8.

The required coordinate frames to carry out such an implementation are shown in Figure 3.9. The subframes for the SDE, IID and Camera are shown in Figure 3.10, 3.11 and 3.12 respectively. The transformation matrices between these frames

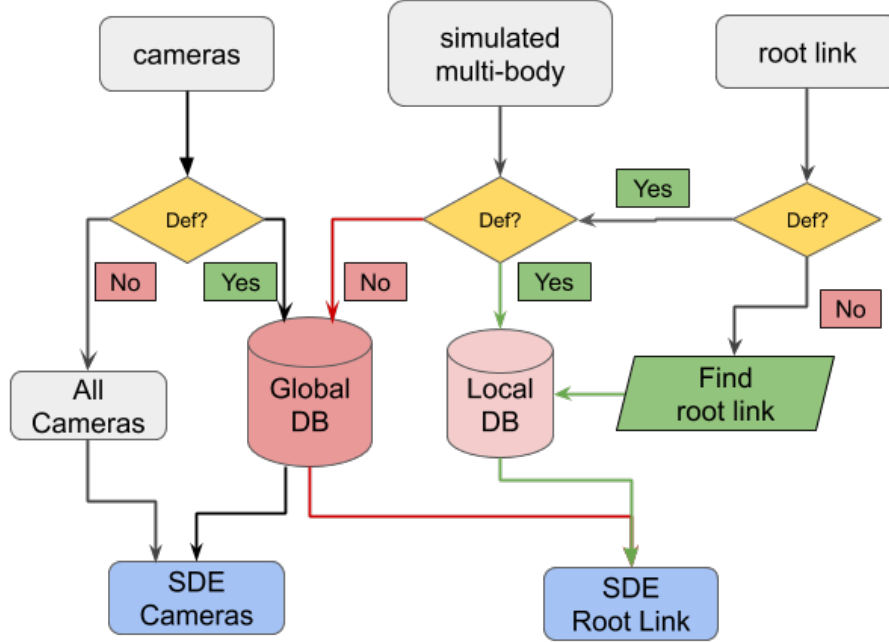


Figure 3.7: This flow chart represents the internal process of binding an IID to an SDE and a Camera. These parameters are specified using the front end format shown in Figure 3.8. The user has to specify at least one of the two fields “simulated multi-body” or the “root link”. If both the “simulated multi-body” and “root link” are defined, the root link is searched for in the simulated multi-body file. If a “root link” is not set, then the body with the least number of parents in the “simulated multi-body” is treated as the root link. Lastly, in case, the “simulated multi-body” is not defined, it is expected that the “root link” refers to a body already present in the simulation. The field “cameras” is optional and is used to define the controllable cameras from the corresponding IID. If the “cameras” field is not defined, then all the existing cameras in the simulation are added to the device’s cameras. In any case, the first camera in the IIDs list of cameras is used as the device’s FoR.

are elaborated in Table 3.5. The flowchart in Figure 3.13 shows the set of equations carried out repeatedly in each IID’s thread. For clarity, the equations are re-written in the Table 3.5 with matching equation indices.

For specific applications, some of the intermediate frames can coalesce together. To achieve various control schemes for shared SDE control while using the generalized XVII frame representation, the controller and haptic gains in the file shown in Figure 3.8 can be set accordingly. Based on the various ways in which theses gains can be set, four different control schemes have been identified:

```

MTMR:
  hardware name: MTMR # Name used for device discovery

  deadband: 0.01 # Threshold value
  max force: 10 # 10 # In Nm
  workspace scaling: 5 # Compared to simulated world

  location: {
    position: {x: -0.5, y: 0, z: 0},
    orientation: {r: 0, p: 0, y: 0}} # Transform of SDE in Sim-W
  orientation offset: {r: 0.0, p: 0.0, y: 0.0} # Tip orientation offset

  haptic gain: {linear: 0.03, angular: 1}
  controller gain: {
    linear: {P: 500, D: 10},
    angular: {P: 50, D: 1}
  }

  simulated multibody: "../forceps.yaml" # ADF file for simulated multibody
  root link: Base # Name of body in simulated multibody
  cameras: [cam1, cam2]

  button mapping: {
    a1: 1, # Action 1 Button Index
    a2: 2, # Action 2 Button Index
    g1: 5, # Button to open and close jaws
    next mode: 3, # Next Mode Button
    prev mode: 4} # Previous Mode Button

```

Figure 3.8: The specification of an IID and its simulation parameters using the front-end specification format. The important parameters for this discussion are the three fields namely “simulated multi-body”, “root link” and the list of “cameras”. The “simulated multi-body” is a description file that defines a proxy simulated multi-body that will be controlled by this IID. The “root link” refers to a body in the “simulated multi-body” or an existing body in the simulation that will be bound to the IID. The “cameras” field is a list of cameras controllable from this IID. The combined use of these three fields is discussed in Flowchart 3.7

- Symmetric Control Input and Symmetric Force Output (SISO)

This refers to the position control of an underlying SDE via multiple IIDs, all of which can control the position and thereby feel the haptic feedback resulting from the interaction as well as the input from other operators sharing the SDE. For this implementation, the controller and haptic gains need to be enabled for the same axes for all the sharing operators. Examples of this implementation

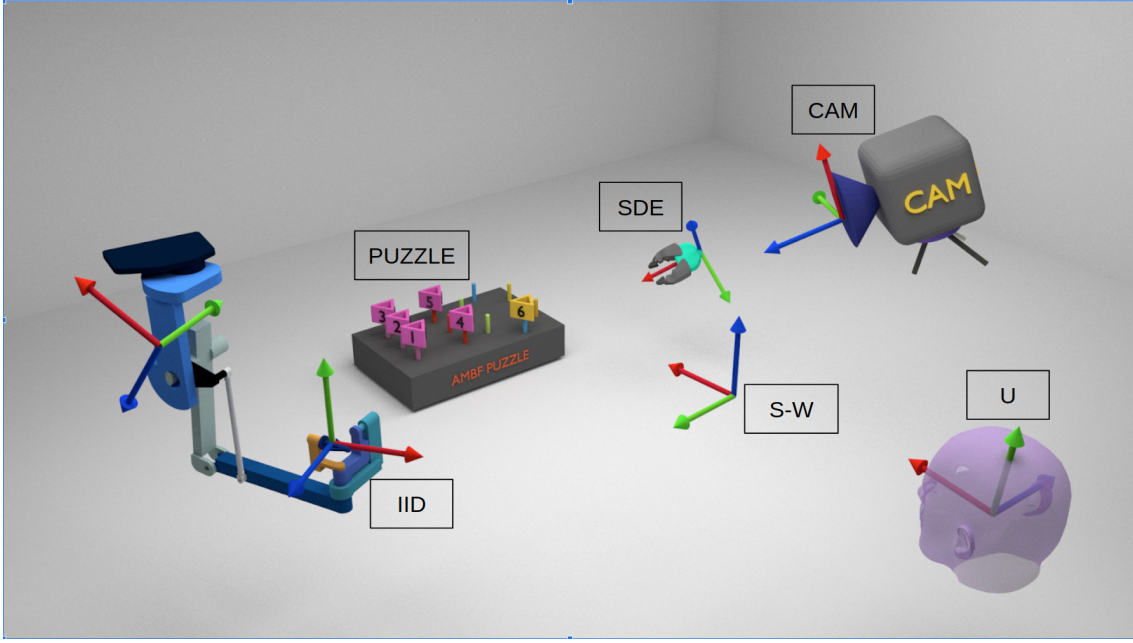


Figure 3.9: A zoomed out view of the components involved in a unilateral or multi-lateral control. The inclusion of the user frame U is important as the user has to deal with the difference between the device base frame and the simulation world's (or camera's) frame as a reference.

include two (or more) operators controlling the position (and orientation) of the same SDEs and each operator can feel the interaction, as well as the force input, of the other operators.

- Symmetric Control Input and Asymmetric Force Output (SIAO)

All the operators sharing the SDE can control the position of the SDE, however, not all users sense the forces in all the degrees of freedom. The force axes can be separated by either linear forces and angular moments or even individual axes of forces and moments. For instance, one operator can sense the force feedback, while the other can only feel the angular moments, or one operator can sense forces in the x,y direction, while the other(s) can only feel the forces along the z axes. The haptic gains are set to zero for axes along which one operator does not sense haptic feedback while setting to some positive value for the other operator(s).

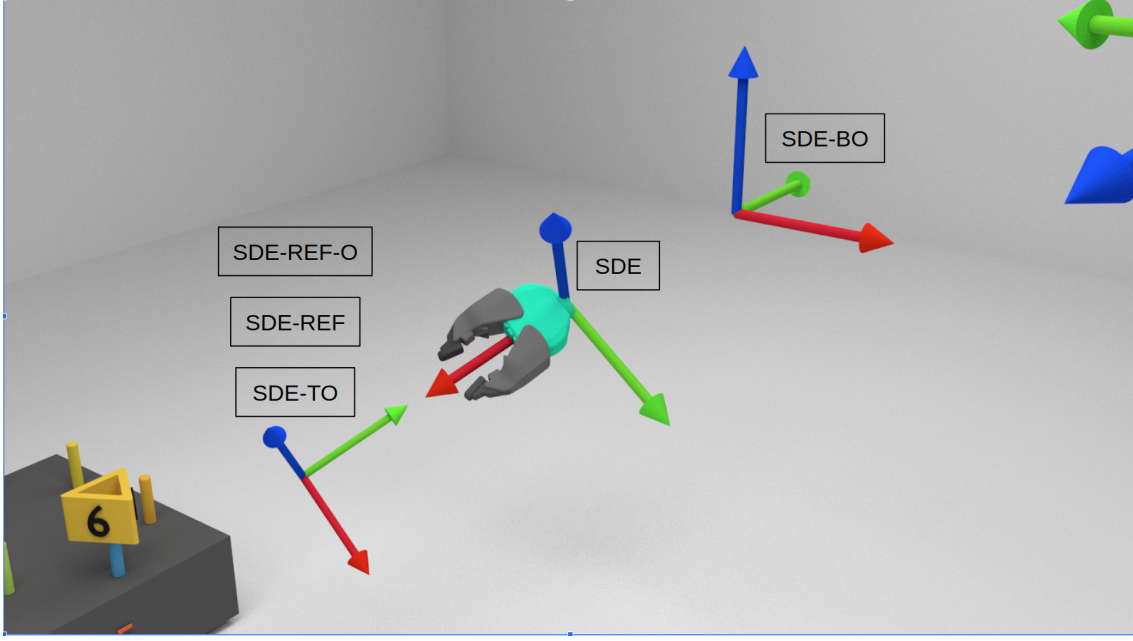


Figure 3.10: A visual illustration of the SDE frames. These frames are defined for each IID-SDE pair, in-case of multi-lateral control, each pair has its own set of variables. The frames F_{SDE-TO} and F_{SDE-BO} are usually defined at initialization and remain fixed throughout the simulation, while $F_{SDE-REF}$ and $F_{SDE-REF-O}$ change based on the clutching of device position control button.

- Asymmetric Control Input and Symmetric Force Output (AIS0)

The operators can only control some axes of freedom while feeling the force feedback along all the freedom axes. The controller gains for some desired axes are set to zero for one IID and set to some positive value for the same axes for other IID(s). The haptic gains are set to some positive value for all the common control axes between the IIDs. An example of this control scheme is two (or more) operators sharing an SDE, such that, one operator can only control position or orientation (or some position axes) while sensing the forces along all the shared axes. The other operator(s) can control the remaining axes and also sense the forces along all the shared axes.

- Asymmetric Control Input and Asymmetric Force Output (AIAO)

The operators can control and sense the forces only along some axes of freedom.

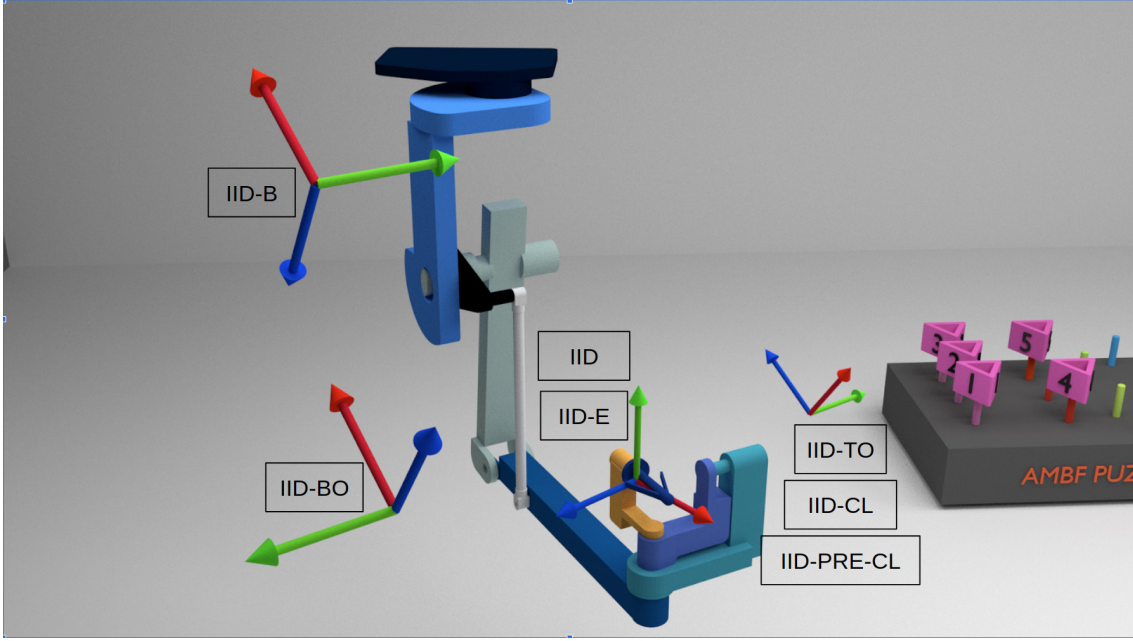


Figure 3.11: The frames involved in a generalized representation of an IID. The frames F_{IID-BO} , F_{IID-TO} , F_{IID-B} and F_{IID-E} are meant to be handled in the corresponding device drivers (dVRK Arm Plugin in case of the dVRKs) while the frames F_{IID-CL} , $F_{IID-PRE-CL}$ and F_{IID} are handled in the Asynchronous Framework.

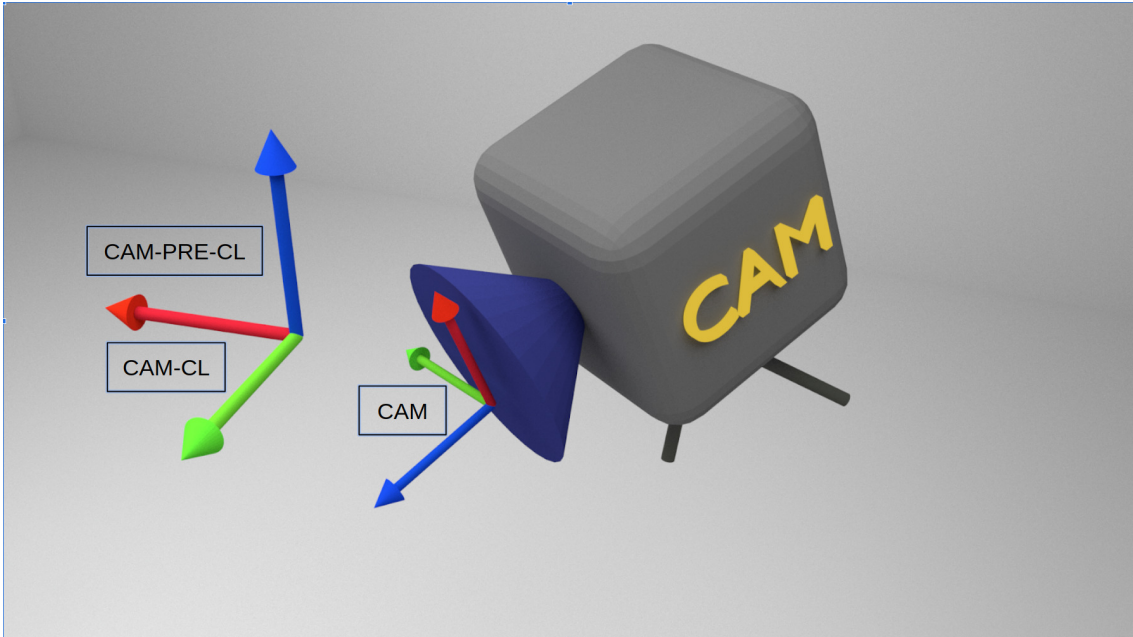


Figure 3.12: The frames associated with the camera which are defined for all IID-SDE-Cam triplets. Each triplet unit has its own set of these frames.

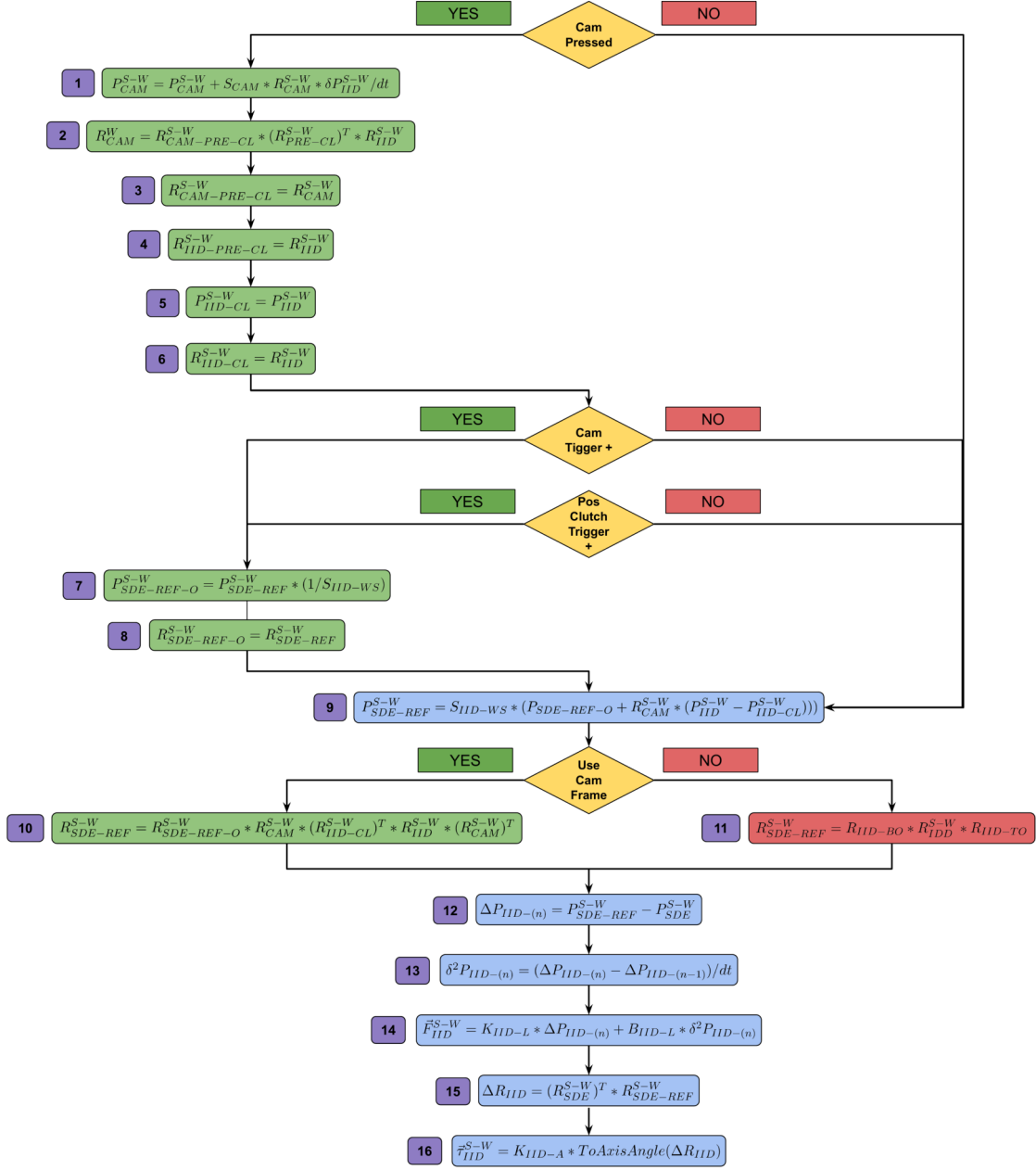


Figure 3.13: A flowchart depicting the process of controlling an IID in uni-lateral or multi-lateral control in single device thread. Each IID has its own thread and this flow chart repeats asynchronously.

The control and force feedback do not necessarily need to be along the same axes. For this scheme, the controller and haptic gains are set to zero for some desired axes for one IID, while the gains along the same axes are set to

some positive value for other IID(s). This scheme can also be referred to as the mixed control scheme, wherein, operators do not have a common set of control and force-feedback axes among themselves.

Two additional control schemes are related to the above four schemes, namely, 5) Symmetric Force Input and No Force Output (SINO) and 6) Asymmetric Force Input and No Force Output (AINO).

3.6 Plugin Based Interface for dVRK Masters

The **sawIntuitiveResearchKit** application [20] provides the state and command data for dVRK manipulators via ROS topics. Due to the convenience of having ROS topic interface, many use-case specific applications can be rapidly developed. Before the development of the Asynchronous Framework, several such implementations were developed [85], [86]. While these applications were easy to prototype, they required the re-writing of mostly the same software in a slightly different manner to cater to the new cases. This was redundant and ultimately a more time-consuming task. This was also recognized by some of the core developers of **sawIntuitiveResearchKit** and as a result, they developed a Python package called “dvrk_python” [87]. This implementation is specific to the dVRK and wraps the ROS functionality internally to provide a method based interface. Due to its ease of use and minimal setup, this package is used avidly by the dVRK community.

Moreover, the wrapping of ROS functionality as such allows the code to be embedded inside various applications without explicitly using any ROS dependencies or ROS launch files. By leveraging the ROS based network, the dVRK hardware can be connected to a different PC in the local network and the target applications can be launched on any other PC. This allows the reduction in system load and

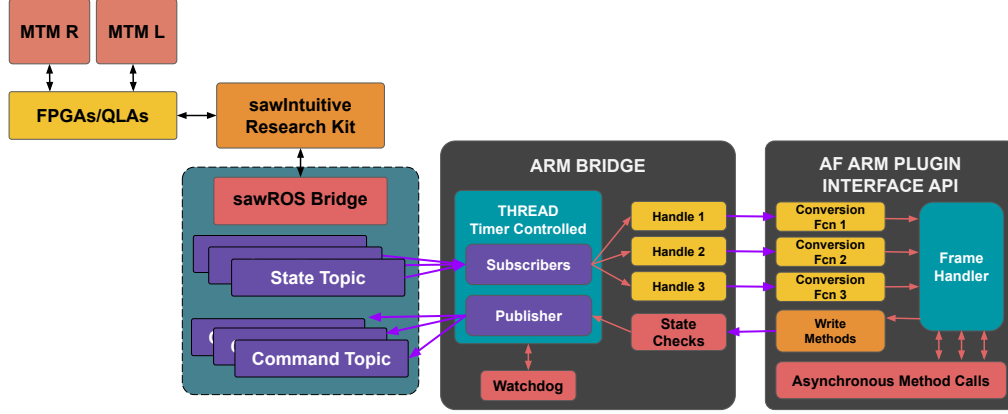


Figure 3.14: This block diagram depicts a plugin based interface for dVRK manipulators using ROS as an IPC. The ROS functionality is sealed in the Arm Bridge Class whereas the ARM Interface exposes API for user applications.

multiple different dVRK systems can be used at once. More importantly, since the dVRK systems have a large footprint, one does not need to be in their vicinity.

Although CHAI-3D does not support dVRK MTMs as input devices, many of the commercial haptic and tracker devices are supported through an extensive API with minimal overhead. To model the dVRK Manipulators according to the CHAI-3D device specifications a plugin-based interface was designed which was written in C++. This plugin, somewhat similar to “dvrk_python”, leverages the ROS topics emanating from sawIntuitiveResearchKit to provide network-based features such as device discovery, control via external PC and asynchronous reads and writes, and also provides device-driver based features such as dynamic linking and watchdog timers for command resetting. On top of that, this plugin handles the F_{IID-TO} and F_{IID-BO} frames described in Section 3.5. The other notable part of this implementation is that it is generic to support other devices that have ROS based communication. For example, Geomagic Touch (haptic device) and Razer Hydra (tracker device) have been used with this interface. This plugin is called the “AF Arm Plugin”. A component view of this plugin is depicted in Figure 3.14.

The “AF Arm Plugin” was initially leveraged to append the gravity compensation for better teleoperated control, however, more recently, the complete dynamic model of the dVRK MTMs has been identified [42] [88]. This dynamic model will potentially be used with the plugin for improved haptic feedback and impedance controllers in a distributed manner.

3.7 Medium for Communication Pipeline

Section 3.4 and correspondingly Figure 3.15, mention the inclusion of a dedicated communication interface for each simulated body, visual entity and IID in the Asynchronous Framework. Having a communication interface allows external applications to control the dynamic simulation by using minimal information contained in the communication payloads. Most of the existing community-based simulators support some form of communication pipelines which makes it easier to interface with different applications instead of having to compile everything together. Many different types of communication mediums exist in modern operating systems that could have been used for the Asynchronous Framework.

A brief review of the commonly used communication libraries is presented in this section and based on their pros and cons, ROS has been selected as a middleware. Although shared memory is the fastest form of data-exchange across different applications on a single machine, it is not scalable nor can the implementation on one platform easily be ported to another. Furthermore, the implementation complexity on a single platform can easily over-shadow the core application for which it is being implemented. Socket communication, although relatively slower [89], is scalable and provides similar implementations across all dominant Linux distributions and even other operating systems. It is also conveniently supported in almost all programming

languages. However, unlike shared memory, socket communication requires the additional step of defining a specification for data serialization and de-serialization since data is transmitted as a stream of characters. Packages such as Zero-MQ [90] and ProtoBuf² simplify this task by outlining specifications for basic data types. These specifications can be incorporated into the application by defining variable names and types in a text-based file which is then included along with the program resulting in programmatically generated code. This generated code represents the data-types and variables specified in the text-based file and can be used at both the transmission and reception ends.

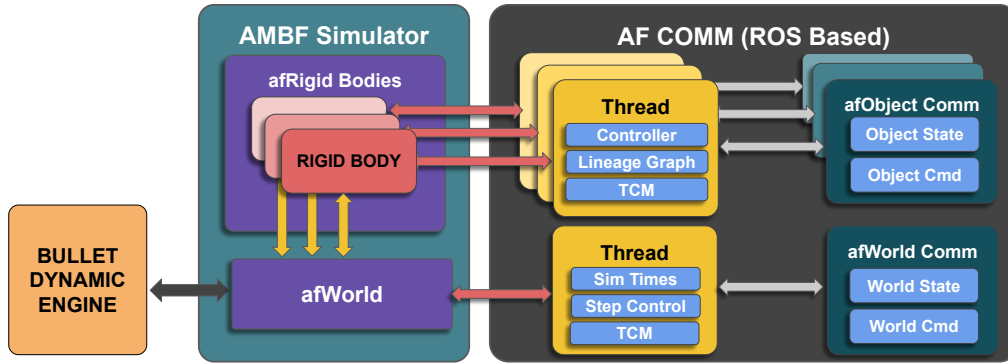


Figure 3.15: A visual representation of the Asynchronous Framework with regards to the C++ AMBF Simulator where each simulated dynamic object is represented as an *afObject*. The *afObjects* utilize independent communication pipelines by exposing State/Command interfaces which allow isolated control

Robot Operating System (ROS) [91] also offers similar features using 3 different types of text-based files which are called messages (.msg), services (.srv) and actions(.action). What sets ROS apart from [90] and Protobuf is the built-in support for several helper tools, both command line and GUI based. These helper tools enhance the ability to debug large scale applications, unlike any other messaging library. ROS also integrates with powerful plotting (RQT Plot [92], PlotJuggler [93]) and logging tools (ROS Bag [94]) which are always useful. These tools do not

²<https://developers.google.com/protocol-buffers>

require any extra setup steps which makes ROS much easier to use and maintain as compared to other socket-based communication libraries.

For communication purposes, all the bodies in the simulation, either kinematic or dynamic (including Cameras, Lights, etc) are called *afObjects*, where ‘af’ stands for ‘Asynchronous Framework’. To maintain a distributed communication structure, each *afObject* owns a separate instance of a communication plugin called *afObjComm*. Likewise, the simulation world is called the *afWorld* and communicates via *afWorldComm*. Unlike multiple instances of *afObjects* that correspond to each simulated body, IID and visual entity, there is only one world and thus only one *afWorld* instance. The communication plugins receive and transmit data asynchronously using indigenous threads but can also share the thread of their parent **afObject/afWorld**.

This design is somewhat similar to ROS Nodelets [95], albeit with some key differences. Unlike ROS Nodelets, the instances of **afObjComm/afWorldComm** distribute/isolate the ROS callbacks using custom callback queues. Custom callback³ are different from the default ROS callbacks. For instance, it is the responsibility of the application to introspect whether new messages have been received and if so, that application has to manage the retrieval. On the contrary, the default ROS callbacks are invoked automatically thus temporarily halting program execution. The advantage of using custom callbacks is that groups of communication interfaces can be managed by the application rather than automatic handling by the ROS communication server. Thus transmission control features that are isolated for each group of communication interfaces can be defined. A conceptual view of this implementation is illustrated in Figure 3.15.

This distributed structure allows each communication instance to define its safety

³https://docs.ros.org/api/roscpp/html/classros_1_1CallbackQueue.html

mechanism (WatchDog timers), speed of communication (publishing and reception frequency) and trigger events (e.g. addition and removal of children data). Even though each communication plugin is isolated from one another in terms of functionality, they are all grouped in a way that they can all be launched dynamically within a single application, without the need for ROS launch files, as is the case with ROS Nodelets. Keeping the Asynchronous Framework isolated from ROS based run-time mechanics while still being able to leverage ROS tools is a distinctive feature of the Asynchronous Framework and provides the means for swappable middleware in the future.

It should be pointed out that although this implementation might seem similar to “AF Arm Plugin”, it is different in many cases. The “AF Arm Plugin” was designed to complement the existing interfaces provided by **sawIntuitiveResearchKit** and ROS based devices whereas the “afCommunication” was developed while keeping in the mind the support for ML, RL and more importantly, a convenient interface for users to interact with the simulation. Moreover, this simplicity should not come at the cost of robustness in handling extensive simulation and communication load.

Due to the asynchronous nature of the communication plugins, a multi-purpose transmission control mechanism (TCM) is built into the design. One component of the TCM is a software-based WatchDog timer which resets the *afCommand* if the timing condition – the invocation frequency of the *afObjComm/afWorldComm* callback – is not met. This adds an extra layer of safety to asynchronous control as it prevents saturation of unsupervised commands to the dynamic simulation and more importantly terminates force commands to actively connected physical haptic IIDs. The TCMs Watchdog timer is re-initiated once a stream of new commands starts to flow in. The secondary function of the TCM is to switch the publishing speed of *afStates* between a minimum and a maximum frequency depending on whether or

Table 3.6: *afWorlds* State Payload

afWorld State	
afState	Description
Msg Num	An incrementing number based on the number of message sent
Server Time	The time read from system clock
Sim Time	The time of the dynamic simulation
Num Devs	Number of Input Interface devices connected to the simulation
Dyn Freq	Frequency of the dynamic simulation

not the WatchDog timer is expired. This reduces the load on computing resources and allows the users to retrieve data at lower frequencies for noncritical tasks.

3.7.1 Bidirectional Communication Interfaces

Both *afObject* and *afWorld* have two interfaces for communication, an ***afState*** for relaying the relevant data outside the simulation environment and an ***afCommand*** for accepting commands to be applied to the underlying *afObject*. These two interfaces implement a scalable input-output design for bidirectional communication through an Inter Process Communication (IPC) medium (Figure 3.15). Based on this design, multiple distributed controllers can be defined for each *afObject*. Moreover, if *afObjects* are connected to each other through the foundational joints (revolute or prismatic), the joints themselves can be controlled using the communication interface of the parent *afObject*.

3.7.2 Communication Pipeline Payloads

The Communication Payloads for the Asynchronous Framework are shown in Tables 3.6, 3.7, 3.8 and 3.9. The payloads are designed to account for external applications with slower execution speeds, such as the Python Client presented in Section 3.8.

The world command has a field called **Enable Throttle**, which is a boolean

Table 3.7: *afWorlds* Command Payload

afWorld Command	
afCommand	Description
Client Time	Time of the client clock
Ena Throttle	Boolean Flag to enable / disable step throttling
Step Clock	Clock to drive the dynamic simulation if Ena Throttle == True
Jump Steps	Number of Simulation Steps to jump at each clock toggle if Ena Throttle == True

Table 3.8: *afObjects* State Payload

afObject State	
afState	Description
Name	The name of the afObject
Sim Step	The simulation step counter number
Wall Time	Time of system clock in Asynchronous Framework
Sim Time	Time of simulation
Mass	The lumped mass of the object
Principal Inertia	The principal inertia
Pose	The pose in the world frame
Wrench	Not implemented yet
User Data	Additional data for debugging or logging purposes
User Data Desc.	Description of user data
Children Names	Name of all the bodies lower in hierarchy
Joint Names	Name of all the joints lower in hierarchy
Joint Positions	Position of all the joints lower in hierarchy

flag and serves to control the flow of dynamic simulation. The other important field in the **ObjectState** and **WorldState** is the field called *SimTime*, which is incremented at each step of the dynamic simulation such that.

$$t_{sim}^n = t_{sim}^{n-1} + dt \quad (3.8)$$

For a real-time dynamic simulation, this time t_{sim}^n matches the system time, as

Table 3.9: *afObjects* Command Payload

afObject Command	
afCommand	Description
Enable Position Controller	Boolean flag to enable/disable Cartesian position control.
Pose	The pose command expressed in the World Frame. If the Enable Position Control flag is true, this pose command will be considered, ignored otherwise.
Wrench	The wrench command expressed in the World Frame.If the Enable Position Control flag is false, this wrench command will be considered, ignored otherwise.
Joint Commands	An array of joint commands to be applied to the children joints
Pos. Controller Mask	This mask is used to choose between position or effort command for Joint Commands array. If this field is not set, all the joint commands are taken as effort control targets. If this array is set, the corresponding Joint Commands with a mask value of true are treated as position control commands
Publish Children Names	Flag to enable/disable the publishing of all the bodies children names
Publish Joint Names	Flag to enable/disable the publishing of all the bodies children joint names
Publish Joint Positions	Flag to enable/disable the publishing of all the bodies children joint positions

recorded by the running application, therefore this information is redundant. As discussed previously, the purpose of using the dynamic time-step is to keep the user-interaction consistent. However, the Asynchronous Framework can also be run based on a fixed time-step by simply specifying the correct command-line arguments. This results in a non-real time dynamic simulation. Similarly, the simulation can be run at a speed greater than the ticking of the real-world clock. Such a scenario is useful in cases where autonomous training and learning are desired without human interaction (using input interface devices) and thus having a separate field for the simulation time is useful for mapping.

3.7.3 Normalized Joint Control of Multi-Jointed SDEs

Training simulations targeting manipulation applications require grippers (forceps, retractors, etc.). Specifically for surgical applications, the IIDs such as dVRK MTMs and even gaming devices such as Razer Hydras have a single DOF for controlling the open and close jaw angle. The SDEs representing these physical input interface devices may consist of articulated rigid bodies connected via sliding or rotating joints. The abstract control of the jaw angle of the SDE is preferable over controlling the explicit joint position or effort for every joint. For this reason, the SDE is designed such that each joint limit is normalized and has an actuation direction that allows it to fully close when the angle is set to 0 and fully open when the angle is set to 1. Based on this design, one can generate a whole class of grippers, an example shown in Figure 3.16, that are controllable via scalar jaw angle input.

Since the simulated bodies in the Asynchronous Framework are represented using maximal coordinates, the joint torques and efforts, which are representations in the reduced coordinates, have to be re-converted to Cartesian Space coordinates. For this purpose, if the axis of freedom of a child body B expressed in parent body A is

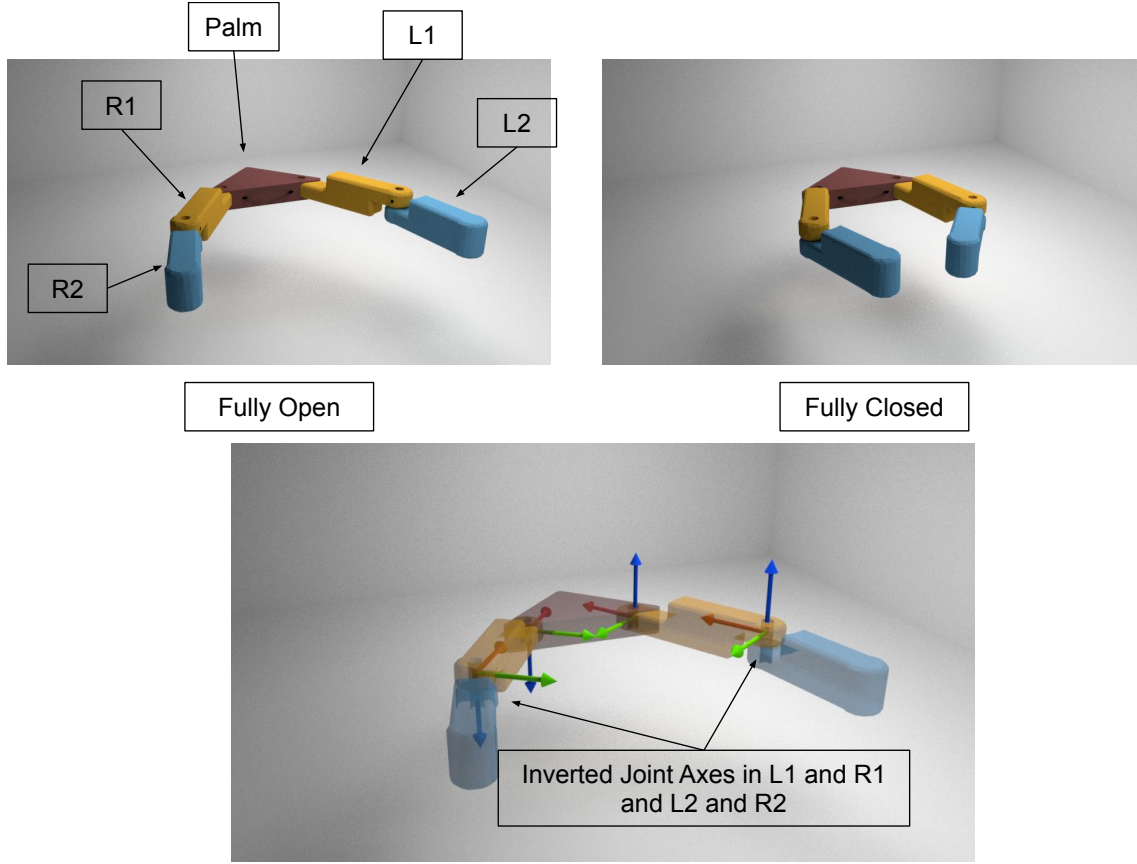


Figure 3.16: Generating grippers such that the joint axes between the left and right fingers (and sub-links) are inverted. This allows a scalar variable to map to multiple joints and allows a generic interface with IIDs having only one pinch DOF.

ax^A , then:

$$(ax^W)_n = R_A^W * ax_J^A \quad (3.9a)$$

$$\Delta R_A^W = (R_A^W)_n * (R_A^W)_{n-1} * (R_A^W)_n^T \quad (3.9b)$$

$$\Delta R_B^W = (R_B^W)_n * (R_B^W)_{n-1} * (R_B^W)_n^T \quad (3.9c)$$

$$\Delta ax_A^W, \delta\theta_A^W \leftarrow To Axis Angle(\Delta R_A^W) \quad (3.9d)$$

$$\Delta ax_B^W, \delta\theta_B^W \leftarrow To Axis Angle(\Delta R_B^W) \quad (3.9e)$$

$$\tau_B^W = -\tau_A^W = K_J ax_n^W (\theta_d - \theta) + D_J ax_n^W (\dot{\theta}_d - \dot{\theta}) + D_C \Delta ax_A^W \Delta \theta_A^W \quad (3.9f)$$

3.8 The Python Client

As discussed in the introduction of this chapter, many of the popular libraries for learning and training have Python interfaces (Keras, GYM, Tensorflow/Theano, and Keras-RL). In alignment with these preferred interfaces, a stand-alone Python client was developed to complement the Asynchronous Framework. The design of the communication interfaces (Section 3.7.1) and the implementation of the Python Client were done alongside each other. Python tends to have slower execution speeds when compared to compiled applications. To build on the robustness of the Asynchronous Framework, the following requirements were identified for a complementary Python client.

- **Online and Offline Training based on Deterministic Data**

Offline training of data can easily be implemented for any dynamic simulator and examples of such implementations can be found in [96] and [79]. Generally, online training has a broader set of requirements since the external application has to account for the round-trip communication overhead and asynchronous

data update. Offline training is relatively simpler and can be covered under the requirements of online training.

- **High-Speed Closed-Loop Control**

High-speed closed-loop control is slightly related to the previous point. The challenge here is to manage a large number of *afObjects* while still being able to keep a high communication speed and low round-trip latency.

- **Distributed Handling of Objects**

This requirement builds upon the previous requirement. Since the C++ implementation of the Asynchronous Framework treats *afObjects* in a distributed and asynchronous manner, a sequential implementation in the Python client would nullify the associated advantages. Therefore the Python client should replicate the underlying design philosophy of the C++ Asynchronous Framework and allow parallel handling of the underlying objects.

Based on the listed requirements, the Python Client makes use of the bidirectional communication of **afObjects** and then creates callable instances of *afObjects* and *afWorld*. These instances have encapsulated ROS publishers and subscribers and are grouped only for dissemination purposes by the Python Client, other than that, they are isolated from one another. Similar to their C++ counterparts, the Python **afObjects** are asynchronous. Data sequencing techniques and payload time-stamps are used to keep track of states, actions and rewards, thus allowing deterministic data management for machine learning applications. A distinctive feature of the Asynchronous Framework is the minimal initialization time and the Python

client replicates this by directly using ROS topics (names and message types) for implicit discovery, initialization and dispatch of `afObjects`.

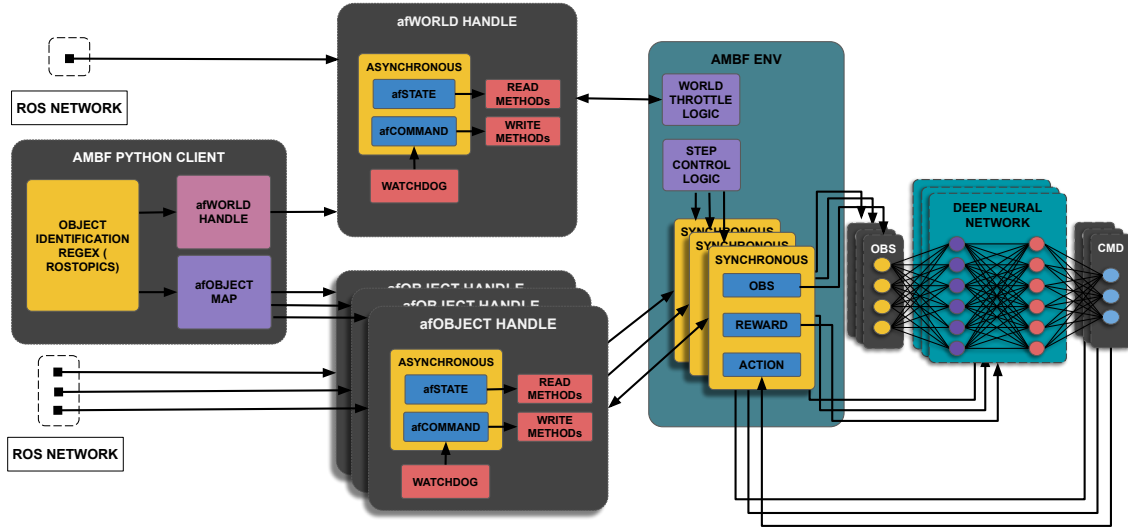


Figure 3.17: The Python Client communicates with the AMBF Simulator using ROS as a middleware, AMBF ENV retrieves the requested handles for objects from Python Client and provides a GYM compatible interface

Another useful feature of the Python client is building upon the joint control interface exposed by the `afCommand` message. All the children's joints of any parent `afObject` can be accessed using either integer indices or the actual joint names. In this regard, it is possible to control the joint positions and efforts using instantaneous and non-blocking method calls. Essentially, the desired position or effort keeps publishing without withholding the program control. This makes the testing and debugging of position and force-based controllers using the Python Client very easy. The client still has an option to change this behavior and reset the commands based on a Watchdog timer which is encapsulated with each Python's instance of the `afObject`. Similar to the C++ TCM, this timer enforces command resetting and thereby provides an extra layer of safety. The overview of the internal workings of the Python Client and its connection with learning interfaces is shown in Figure 3.17.

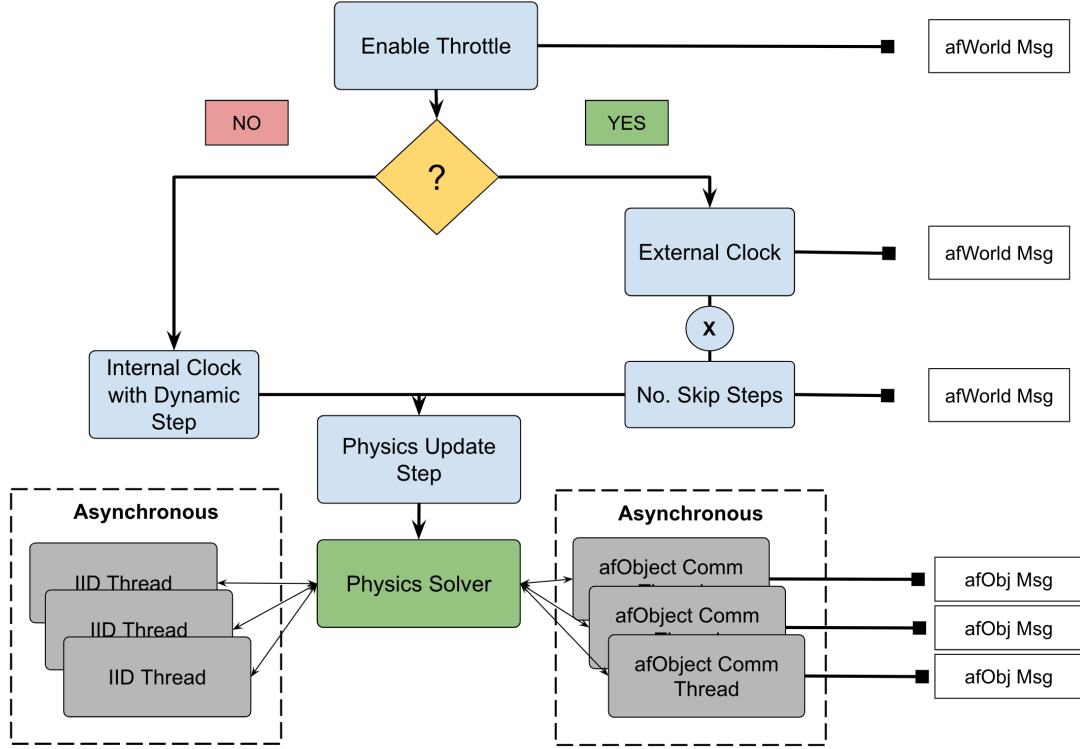


Figure 3.18: The Flowchart depicting the process of throttling the dynamic simulation based on setting the “Enable Throttle” flag by an external application. Once the flag is set, the external application is responsible for providing a clock as shown by the field “External Clock”. The default value of “No. Skip Steps” is set to 5, which is the number of simulation steps the physics will take between each clock toggle. This field can also be set dynamically.

As discussed in the requirements for the Python client, there needs to be a mechanism to account for the round-trip latency between the Client and the Asynchronous Framework for ML and RL applications. At the very least, this mechanism requires that the dynamic simulation be paused (throttled) to give the underlying neural network adequate time to process the **afStates** and compute the next **afCommands** and vice-versa. The requirement for this throttling comes from the action-reward pair for the valid Markov States in Reinforcement Learning problems [97]. This, in effect, mandates the states to have associated rewards. These rewards are only defined as a function of the action taken. In the case of Asynchronous Framework which has a distributed and asynchronous communication implementa-

tion, the time delay between an action being applied to the reward being retrieved makes the action-reward mapping meaningless. To circumvent this, the simulation can be throttled between the update-steps of training (forward and backward pass of the Neural Network).

To achieve such a throttling, the Python Client can leverage the **afCommand** of the **afWorld**. The client firstly enables a throttle flag which forces the dynamic simulation to stop auto-stepping and instead, progress on an external trigger. This trigger can be provided by the communication medium, and therefore, also the Python client, in the form of a clock signal. A flow-chart representing this take-over of the dynamic simulation scheme is presented in Figure 3.18. The asynchronous design allows one to dynamically change the physics simulation frequency while the connected input interface devices can still run in real-time threads. Such a setup would not be possible with a “sequential implementation” as throttling the simulation frequency would throttle the update of the device drivers and affect the force feedback for haptic devices.

3.9 Results and Discussions

A PC setup consisting of an Intel(R) Core(TM) i7-3770 CPU (3.40GHz), Fujitsu 32 GB DDR3 RAM (1333 MHz) and an Nvidia GTX 1060 (8 GB RAM) GPU running Ubuntu 18.04 was used for the demonstration of results.

To demonstrate the robustness of the Asynchronous Framework, the difference between the “sequential” and “asynchronous” implementation was analyzed by recording the dynamic and haptic update-rates of multiple IIDs connected to a real-time dynamic simulation. The devices include five haptic controllers, of which two are Novint Falcons, one Geomagic Touch, and two Master Telemanipulators

(MTMs) from *Intuitive Surgical Inc., Sunnyvale, CA, USA*. As shown in Figure 3.19(a), (c) in the sequential implementation, the update-rate never meets the 1 kHz set-point. On the other hand, in Figure 3.19(b), and (d), the device update-rates stay close to 1 kHz but the dynamic update-rate can swing depending upon the complexity of equations for the physics solver.

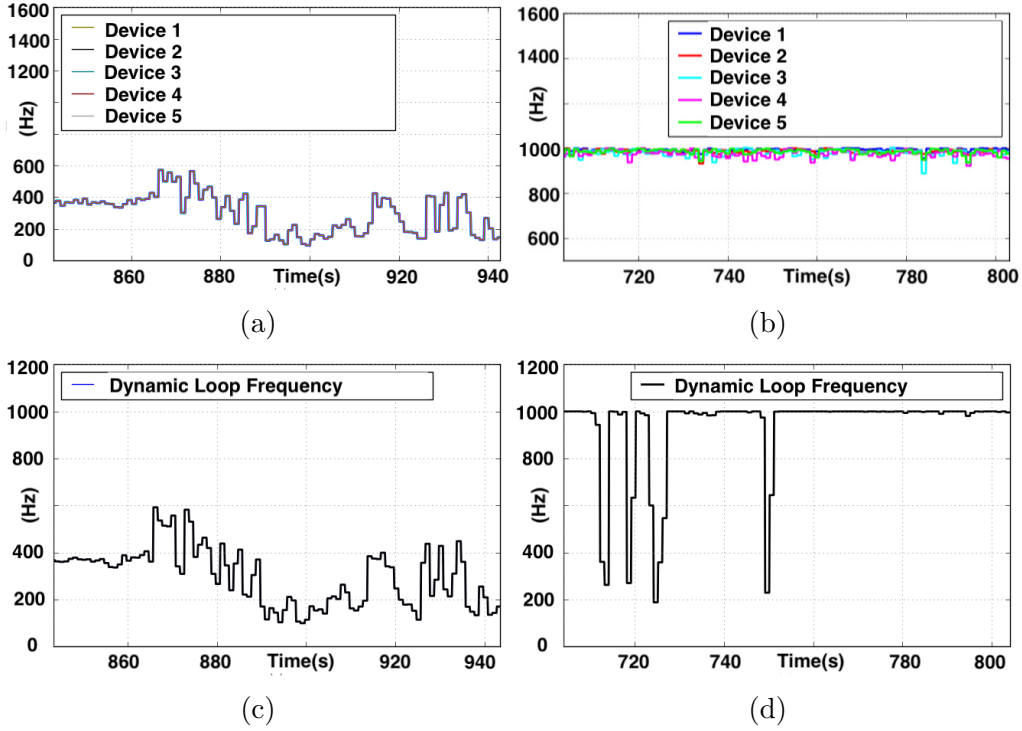


Figure 3.19: Figure (a) and (b) show the haptic update-rate of 5 devices when controlled ‘sequentially’ vs ‘asynchronously’, respectively. Figure (c) and (d) show the corresponding rates for physics update-loops for ‘sequential’ vs ‘asynchronous’ control

Having demonstrated the performance of device updates with a varying physics simulation frequency, the controller performance of the Simulated Dynamic end-effectors (SDEs) was analyzed in response to varying dynamic update frequency. Since one cannot deterministically reduce the physics update frequency by only using appropriately complex simulation environments, the reduction was induced by leveraging the “step throttling” functionality discussed in Section 3.7.2. The **AF Arm Plugin** interface (shown in Figure 3.14) was used to spawn two input devices

(MTM-R and MTM-L) and controlled based on a parametric trajectory described by the following equation:

$$P_{input} = P_{off} + [a_p \sin(t_c t), b_p \cos(t_c t), c_p \sin(t_c t)]' S \quad (3.10)$$

Here, P_{input} is the commanded position of the Input Device while the R.H.S consists of offset P_{off} , time constant t_c , scale S , system time t and a_p, b_p, c_p are the major/minor axes in the 3 Dimensional space. In order to generate a high velocity, the variables were set as follows, $t_c = 4.0$, $S = 0.1m$ and $a_p = 1, b_p = 1, c_p = 2$. A script systematically throttled the dynamic update frequency and recorded the controllers' performance as the magnitude of error from the set-point. Figure 3.20 shows the output of the controllers performance for $n = 5000$ readings. As visualized in the graph, the controllers' response began to suffer as the dynamic-loop's frequency fell below a threshold frequency of $\sim 50Hz$.

While this result is not significant in itself as it only shows the performance of the corresponding PD control, it is significant because an external application with slower execution speeds can be used in conjunction with human subjects to perform collaborative tasks. One particular example of such slower applications is a trained agent (NN trained through ML or RL) which would need to throttle the dynamic simulation to have adequate time to process each state and generate the correct response command. This throttling would not affect the connected users as long as the dynamic simulation frequency stays above a certain threshold ($> 50Hz$).

This result is also significant in the sense that a complex environment that is unable to run at the desired simulation-frequency would also not impact the haptic IIDs as long as it keeps above the threshold frequency. Such a simplistic test can be used to analyze the haptic response due to the use of shared data-structures discussed in 3.4.2. Essentially, the set-point error (both in terms of the position and

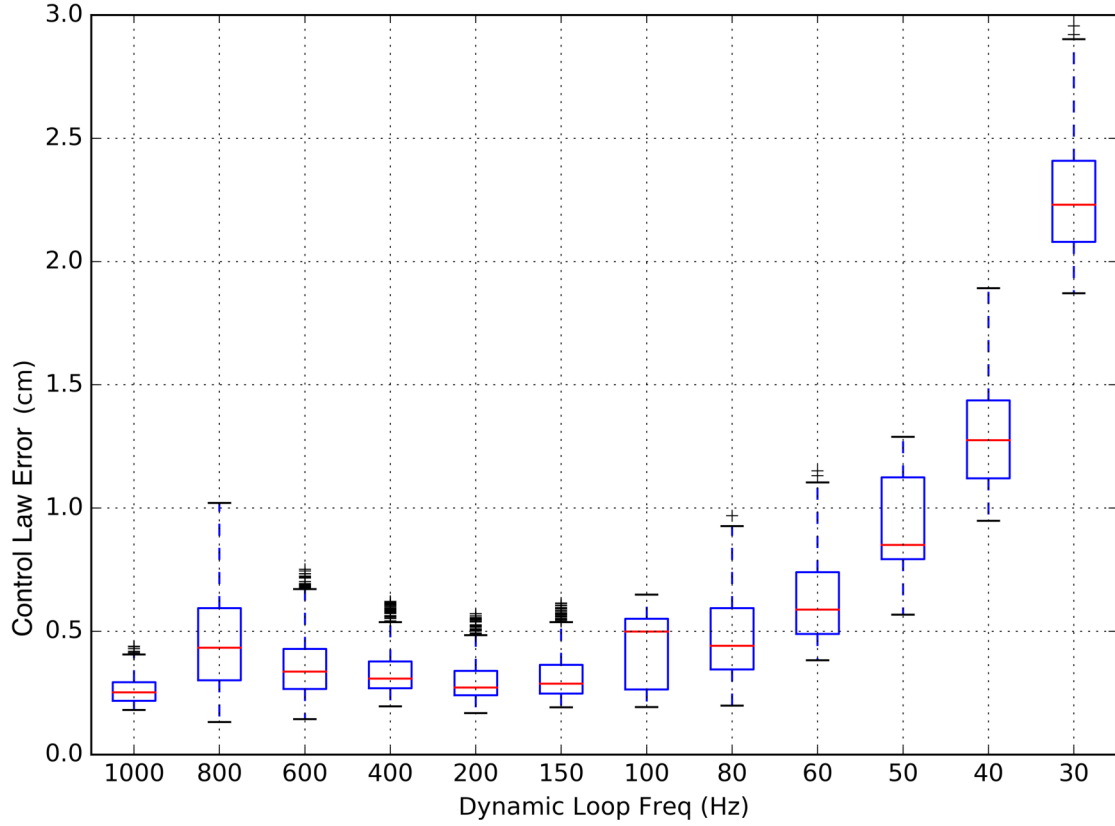


Figure 3.20: Reponse of haptic controllers with degrading dynamic loops frequency

velocity) is used, in the correct Frame of References (FoRs), for the PD controllers of each SDE and IID. Especially for haptic IIDs, the PD controller can be incorporated inside more advanced controllers such as the generic Impedance controller or specific Inverse Dynamics Controllers. Better yet, these advanced controllers can also be run separately and their output can be incorporated by using the communication pipeline for each IID.

Moving on, Figure 3.21 demonstrates an application involving two users controlling a pair of IIDs each. Both users have their independent view-ports and can control them individually. The user controlling the dVRK MTMs (visible as PIP on the top right) has force feedback as well as visual feedback while the user controlling the Razer Hydras (PIP on the top left) only has the visual feedback. This setup

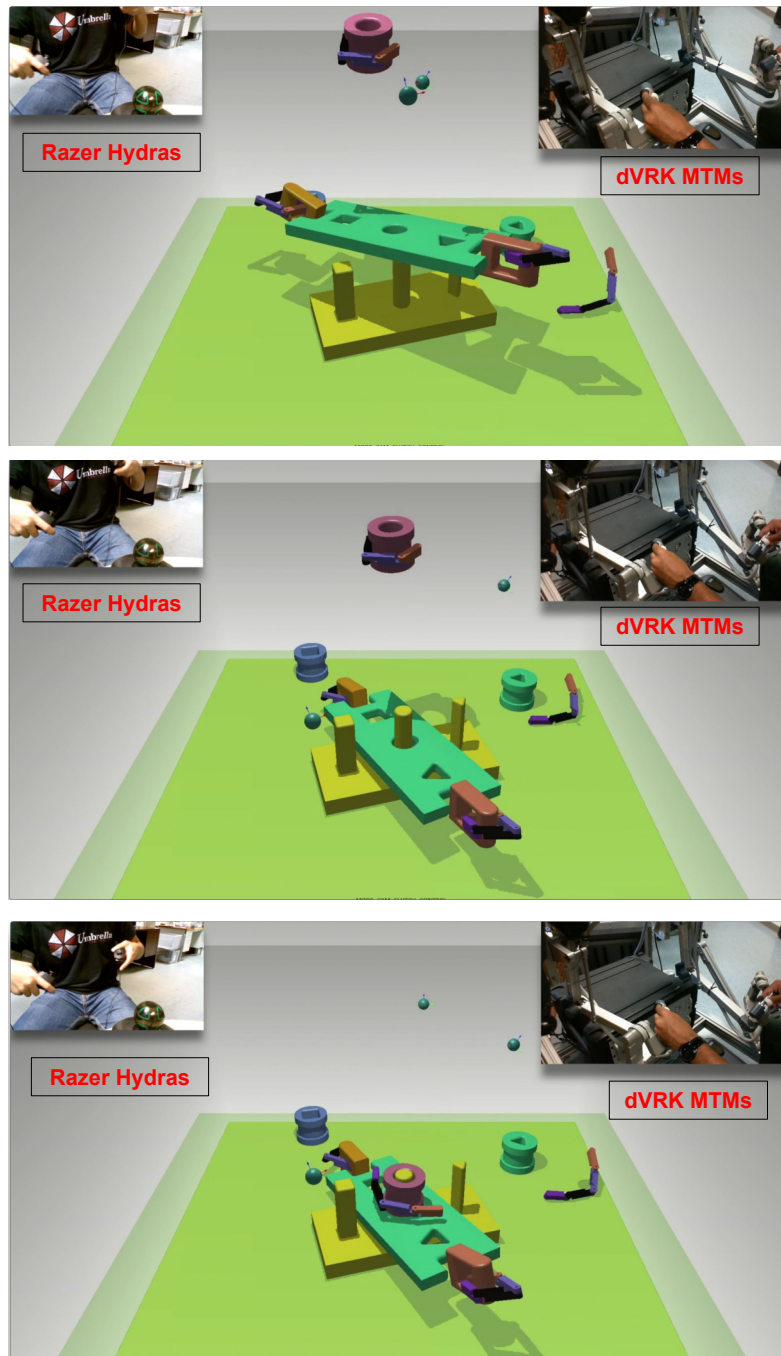


Figure 3.21: These sub-figures show the progression (top to bottom) of a bi-manual task using the AMBF Simulator. The two SDEs holding the green multi-link puzzle piece are controlled by dVRK Masters (shown as Picture in Picture on top right) and the other two SDEs are controlled via Razer Hydra (shown as Picture in Picture on top left)

shows the inclusion of IIDs with different update rates, as the dVRKs update at 1 kHz whereas the Razer Hydras update at $\sim 300Hz$. The goal of the setup is to place the puzzle pieces including three single link rigid bodies puzzles (the Triangle Piece (Green cylindrical shape), the Square piece (Blue cylindrical shape), the Circle Piece (Red cylindrical shape)) and the multi-link chain (Green Plate with Orange Handles) on the Puzzle Base (Yellow Mesh). The Puzzle base and the multi-link Puzzle have a matching set of extrusions and cut-outs respectively, while the three rigid body puzzles have matching cut-outs for the three individual extrusions of the puzzle base. It is important to note that all of the grasping interactions in the simulation are purely dynamic, therefore, they involve a combination of friction due to contact geometry, grip force, slip, and slide. No simplification techniques were applied for appending the grasped object. While this dynamic grasping helps provide a natural feel by allowing gripping slack, it makes puzzle-solving more challenging. Manipulation and grasping in the simulation have their own set of challenges that are addressed in the context of the Asynchronous Framework in Chapter 6.

The sub-figures 3.21 show the progression of a sub-task that involves manipulation of the multi-link puzzle. The multi-link puzzle requires at least two inputs for it to be lifted and placed on the Puzzle Base. This is followed by the single link puzzle pieces being placed on top. All of the four simulated end-effectors can interact with each other and the remaining puzzles. The closed-loop constraint formed by the multi-link Puzzle is felt by the dVRK Masters which constrains the range of motion, which in this case allows better control and manipulation.

The goal of this demonstration is to show multi-user control, therefore the puzzles weren't designed to any particular sub-task (surgical or not). The meshes for the puzzle were created in Solidworks and then imported, scaled and placed explicitly using the C++ interfaces of CHAI-3D that are ultimately wrapped by Asynchronous

Framework. The corresponding lower-resolution collision meshes were generated using mesh-decimation techniques via Blender [98]. The lower resolution collision meshes help maintain stable dynamic update frequency.

A more interesting demonstration is the multilateral (bi-lateral in this case) control of the SDEs. Using the XVII Representation discussed in Section 3.5, two different users control the same underlying pair of proxy SDEs. The haptic and controller gains can be set to achieve 1) SISO, 2) SIAO, 3) AISO or 4) AIAO. In Figure 3.22, a SISO scheme is shown. The user holding the Falcon IIDs can only command the Position (as the Falcon has no orientation sensors) while the dVRK User can command both the Position and Orientation of the SDEs and the moment feedback is disabled. Both the users can feel the forces imparted by each other’s corresponding hand. Similarly, the two users can interact within their own FoRs.

Switching from evaluating the control characteristic, the performance of the distributed communication pipeline is tested. For this purpose, around 300 cubes (primitive shapes) were programmatically spawned in the simulation to achieve both a burdened simulation as well as an extensive communication load by setting the minimum and maximum communication frequency to $[100Hz, 2kHz]$. The distributed communication interfaces exposed the states of each dynamic body as an *afObject* which were read asynchronously to determine the characteristics of the communication pipelines. The ROS introspection tools were used to probe the frequency of *afStates* for a few boxes. The introspection tools pushed the communication frequency of the *afObjects* to the max frequency by commanding zero efforts at frequencies greater than the pipeline’s WatchDog timer frequencies. As shown in the Figure 3.23, due to the excessive load, the dynamic update frequency dropped to around 300 Hz (still real-time), however, the communication speed for all *afObjects* was $\sim 2kHz$. This result emphasizes the utility of segregating every possible

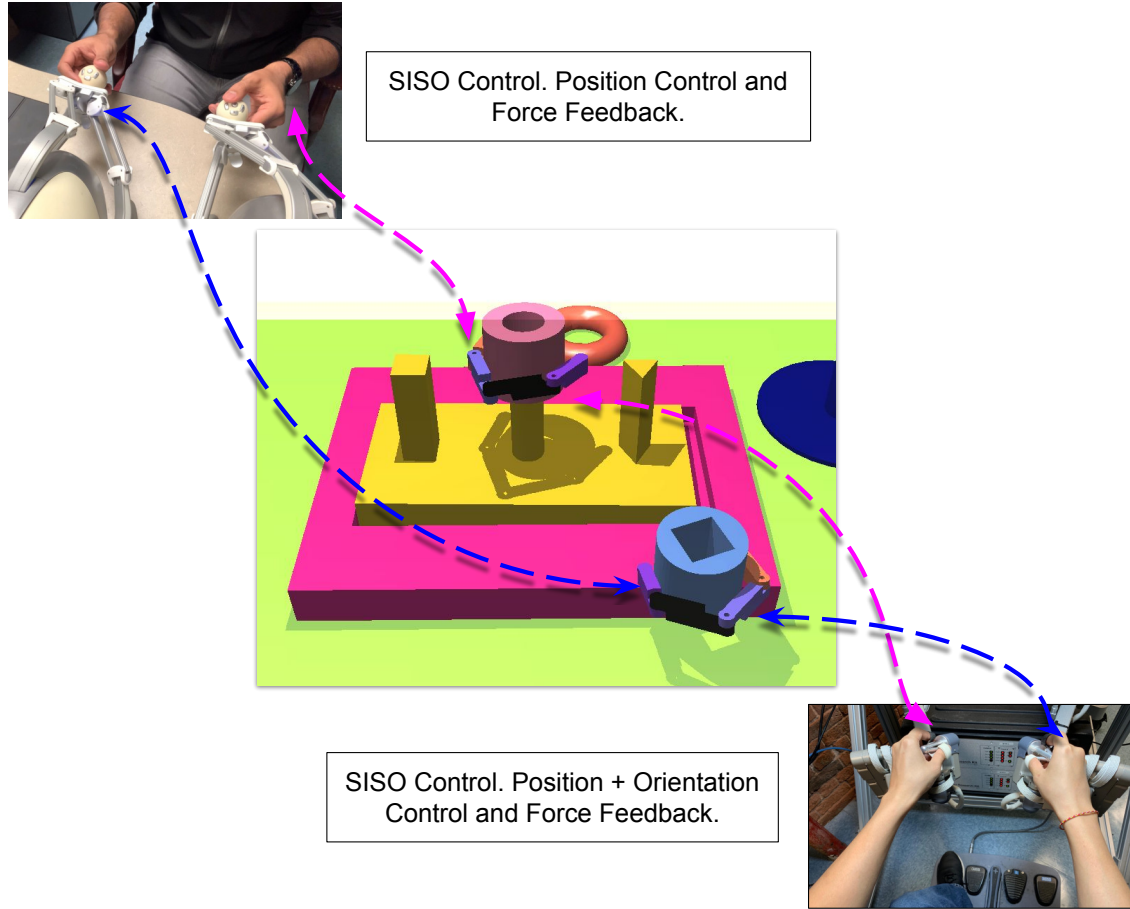


Figure 3.22: Bi-Lateral SISO Control by using a pair to Novint Falcons and a pair of dVRK MTMs to control the same SDEs.

computation from the dynamic update loop to better achieve real-dynamic dynamic simulation.

The communication result, discussed above, was analyzed using the built-in ROS introspection tools, namely “rostopic info” and “rostopic bw” [99] which are written in Python. For more realistic scenarios, the Python Client itself will be used instead for external control. To achieve Synchronous control under the umbrella of Asynchronous control, a stepping control mechanism has been implemented in the Client. This mechanism uses data sequences and time stamps for making sure that the correct data is mapped for the state, action and reward triplets (already

topic (name)	rate(Hz)	min_delta(s)	max_delta(s)	std_dev	window
/chai/env/Box_55/State	1.967e+03	7.868e-06	0.01539	0.0006213	8516
/chai/env/Box_56/State	1.956e+03	8.106e-06	0.01444	0.0006179	8516
/chai/env/Box_57/State	1.962e+03	8.821e-06	0.01361	0.0005808	8480
/chai/env/Box_58/State	1.961e+03	7.868e-06	0.01471	0.000605	8480
/chai/env/Box_59/State	1.965e+03	6.914e-06	0.01263	0.0005788	8478
/chai/env/Box_60/State	1.954e+03	8.821e-06	0.01649	0.0005974	8478
/chai/env/Box_61/State	1.959e+03	9.06e-06	0.01357	0.0005925	8378
/chai/env/Box_62/State	1.954e+03	9.06e-06	0.01312	0.0005545	8378
/chai/env/Box_63/State	1.963e+03	7.868e-06	0.0136	0.0005484	8360
/chai/env/Box_64/State	1.958e+03	7.868e-06	0.01304	0.0005565	8360
/chai/env/Box_65/State	1.955e+03	7.868e-06	0.01275	0.0005779	8286

Figure 3.23: Communication speed of several *afObjects* for an overloaded dynamic environment. The desired communication frequency is set to 2 kHz Dynamic-Loop’s Frequency $\sim 300Hz$, *afObjComm* frequency $\sim 2kHz$

discussed in Section 3.8). Apart from measuring the communication frequency, it is also important to measure the communication latency. Various tests were written for this purpose and supplied publicly with the AF source code. To measure the latency for each message, the current system time-stamp is embedded at the Asynchronous Framework’s end before transmission. In the Python Client, this embedded time is compared to the current time. Since both the C++ AF and the Python Client run on the same machine, the system time can be used as a ground truth.

One can set a queue size of messages in the ROS C++ and Python implementations. The messaging queue size refers to the number of messages that will be stored for retrieval. If the queue is full and new data is incoming, it will be discarded. Figure 3.24 shows the latency characteristics at $\sim 1kHz$ of communication frequency and the messaging queue size of 5 at the Python’s end. It can be seen that the Python Client (or the Python implementation of ROS) does not seem to be able to maintain a stable latency. It was worth discovering whether or not the value of the queue size had any effect on this. For this, a different test was carried out where AF published a topic at $100Hz$ while the Python queue size was set to 5. It can be seen that in Figure 3.25, even with a reduced communication frequency, a

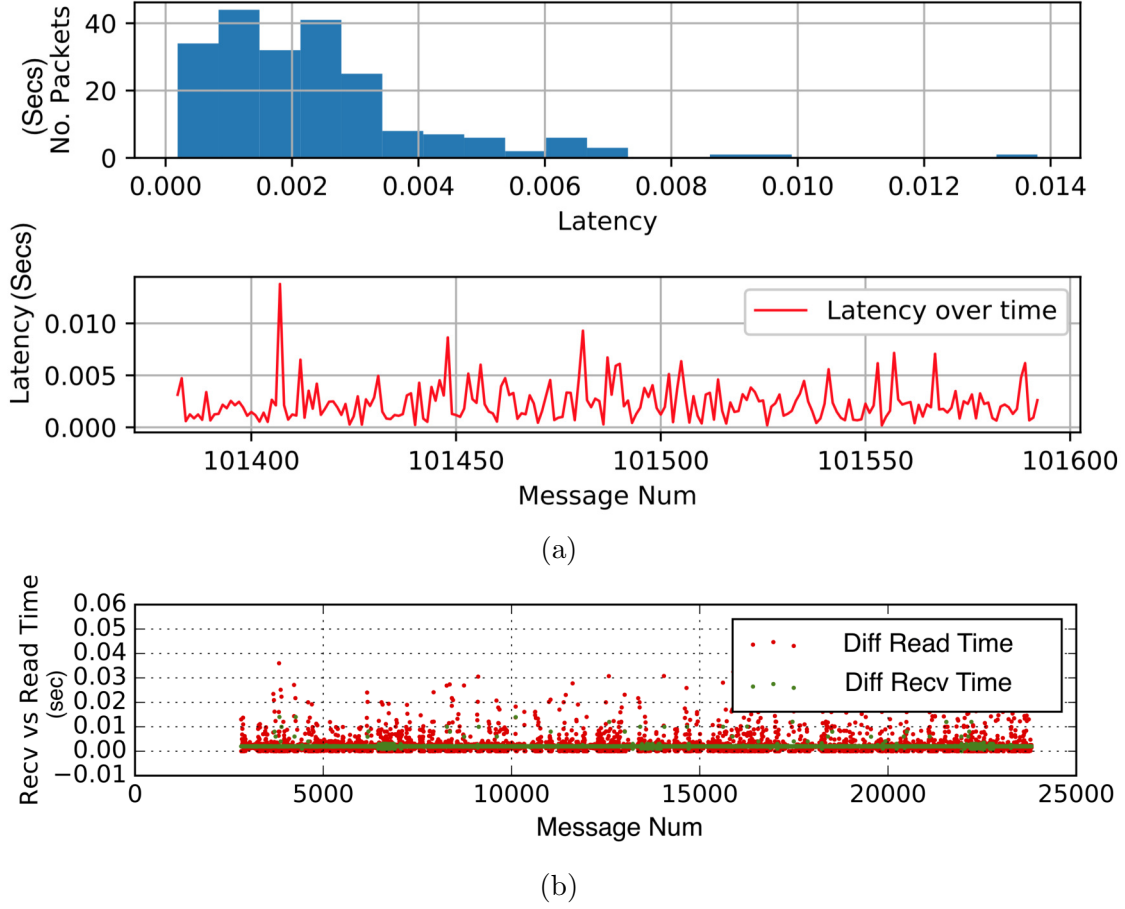


Figure 3.24: (a) The histogram showing the communication latency between the C++ AF and the Python Client using message queue size of ~ 10 (b) The green dots show the difference between every successive new message received from the C++ AF by subtracting from the previous packet's embedded time. Similarly, the red dots show the difference between the current time when the message was read from the previous time the last packet was read.

longer queue aggravates the latency.

For synchronous control, one is concerned with the latest data, and thus the queue-size can be set to 1. This was tested alongside a communication frequency of $1kHz$ and as shown in Figure 3.26, the latency histogram performs much better. For offline training applications, one might need access to more than just the recent data and thus it might make sense to keep a higher value of the queue size. In conclusion, it should be pointed out that in our experiments, the C++ ROS implementation does not suffer measurably to higher values of the queue sizes and thus the bottleneck

is almost always at the Python's end.

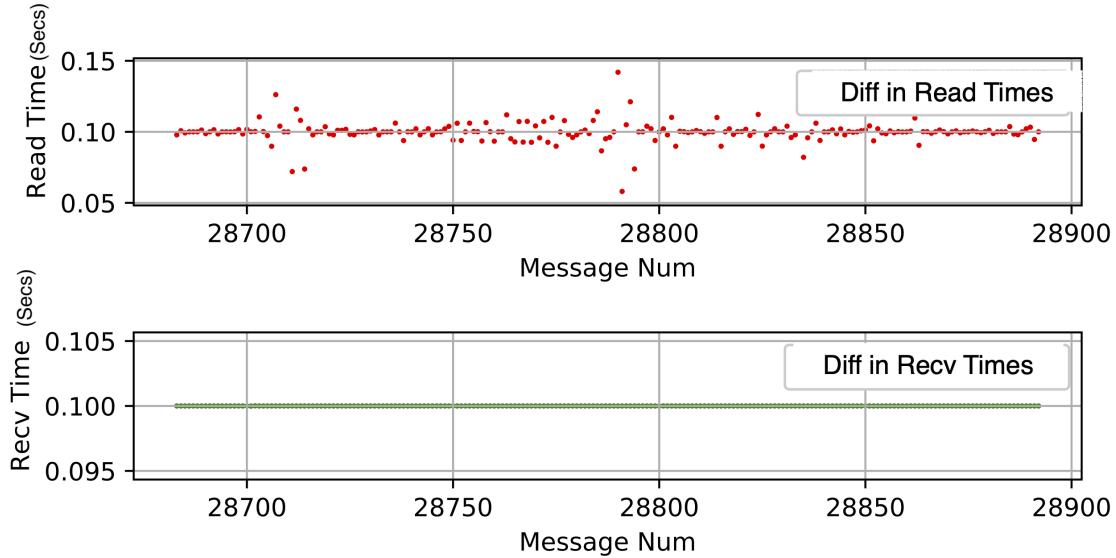


Figure 3.25: The histogram showing the communication latency between the C++ AF and the Python Client using message queue size of ~ 10 .

3.10 Conclusion

This chapter demonstrated the design and development of the Asynchronous Framework for allowing the inclusion of a variable number of IIDs in a real-time dynamic simulation. The framework is especially useful as it allows extensive mapping between cameras, IIDs and SDEs without having to develop temporary applications. All this could be achieved using the configuration file shown in Figure 3.8. The same file can be used to allow the coordinated control modes discussed in Section 3.5. These control modes are of interest as they allow supervised and task performance both with and without force feedback. The performance of the Asynchronous Framework for an actual multi-user and multi-manual tasks was demonstrated in Figures 3.19 and 3.23. These results show a promising implementation that separates dynamic simulation from the control of multiple IIDs and communication

interfaces.

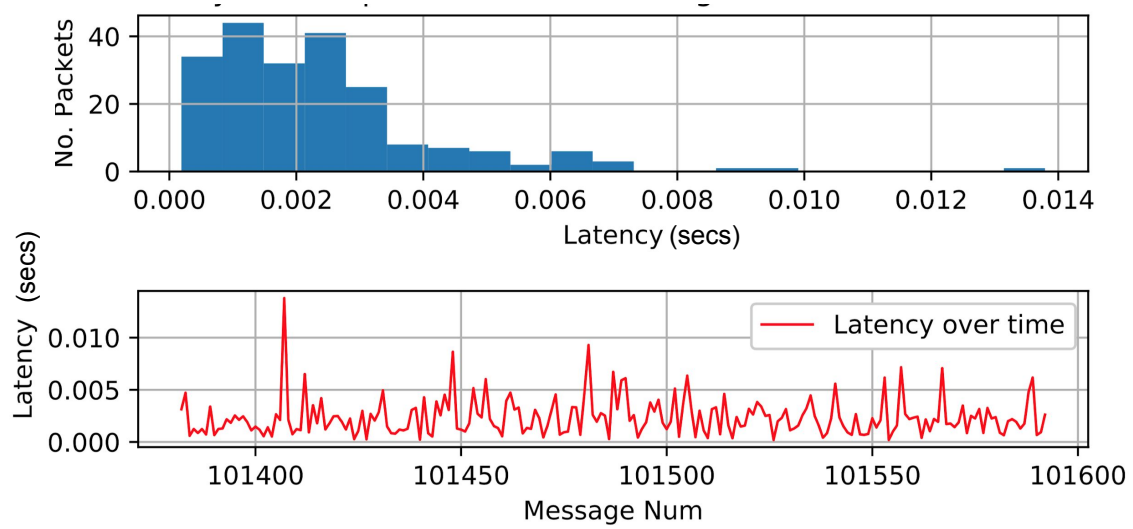


Figure 3.26: Histogram of the time difference between the embedded time of a received packet and the current time for synchronous communication using Step Throttling

Chapter 4

Distributed Format for Robots, Environments and Devices

This chapter discusses the simulation component of the Asynchronous Framework which is called the Asynchronous Multi-Body Framework (AMBF). To enable the simulation of complex robots and environments a viable specification format is required. Section 4.2 presents the limitations of the most commonly used robot representation formats in terms of simulating closed-loop surgical robots. To overcome these limitations, a novel distributed specification format has been developed which is discussed in Section 4.3. This section also discusses the additional features of this specification format that are not present in existing formats. To make this format adoptable within the community, several addons and scripts have been developed that bridge the gap with the existing format. These scripts and addons are discussed in Section 4.4. Finally, the results and conclusions pertaining to AMBF are presented in Section 4.5.

4.1 Published Work

Most of the work presented in this chapter has been published as:

Munawar A, Wang Y, Gondokaryono R, Fischer G, “*A Real-Time Dynamic Simulator and an Associated Front-End Representation Format for Simulating Complex Robots and Environments*”, Intelligent Robots and Systems (IROS), Macao, China, 2019.

4.2 Introduction

Achieving real-time control of multiple haptic input devices with real-time dynamic simulation, presented in Chapter 3, is a novel contribution of this thesis. The contribution presents a framework for using multiple devices and distributed control algorithms together rather than individual application based implementations which are both limited in terms of scalability and require unnecessary re-writing of code. The framework is written in C++ and has a Python client for incorporating intelligent agents and distributed controllers. While the inclusion of IIDs and external controllers was formalized into the framework, the examples of SDEs being controlled as well as other simulated dynamic bodies, were hard-coded in the application itself. It is possible to program many different sets of simulated environments as well as SDEs but it can be argued that such a stiff interface defeats the purpose of the Asynchronous Framework. This could potentially also limit community adoption. This limitation was recognized early on in the design of the Asynchronous Framework and led to the in-depth study of existing representation formats that could be leveraged to define various environments for the Asynchronous Framework.

The Universal Robot Description Format (URDF) is one of the most widely used representation formats for robots. There exist other formats that are either driven

from URDF (Standard Description Format (SDF) [32]) or allow conversion from URDF (such as MuJoCo [100] format and V-REP Simulator [70]). Arguably, URDF played a pivotal role in the success and community adoption of Robot Operating System (ROS) [91] and is tailored to serial manipulators and robots. While there are ways to visually achieve redundant mechanisms using **mimic** tags, realistic closed-loop constraints are not possible as the limitation broadly comes from the design philosophy of URDF. The idea of a robot, as envisioned by URDF, is a spatial tree of bodies wherein the joints are essential parts of the links. While this philosophy is the foundational building block of kinematics and visualizations using the default ROS simulator RViz (and derivative software such as MoveIt), it thwarts the ability to define unconnected, sparsely and densely connected combinations of bodies.

The Simulation Description Format (SDF) is employed by the Gazebo Simulator [32] and is similar to the URDF in many core aspects while defining serial robots. SDF addresses the key limitation of the URDF in defining closed-loop mechanisms, however, the latest Gazebo simulator (9.0) does not support direct control of parallel linkages using ROS. While URDF can only define a single robot per description file, SDF can support the distributed description of robots. Moreover, SDF is designed for more general-purpose use with support for environment entities such as lighting, scene-objects, and sensors. Scene objects are relatively straight-forward to describe so in this discussion, the role of representation formats is limited to defined robots and inter-connected mechanisms.

Both the URDF and SDF (and even MuJoCo) use XML language, which although historically has been used to store and transmit configuration and description data, is not known for human readability. This limitation has somewhat been the reason behind the development of other markup languages such as JavaScript Ob-

ject Notation (JSON)¹ and Yet Another Markup Language (YAML)². While XML retains its place as the back-end tool for data storage, YAML and JSON are gaining wide adoptability in front-end applications. In addition to the readability component, both JSON and YAML are feature-rich as compared to XML. For example, YAML provides inherent support for macros in the form of **anchors**, which tend to be useful for the specification of properties. Moreover, vectors are also supported in a better manner in YAML.

Gazebo [32] is supported across major operating systems (e.g., Microsoft Windows, Mac OS, and Linux), however, it is used most commonly with ROS (Linux). While Gazebo is feature-rich and allows for robust support for a large number of sensors as loadable plugins, its support with URDF, and consequently external control via ROS-topics is complicated and non-robust. The process of going from a URDF to SDF, and eventually loading joint controllers, communicable using ROS-topics/ROS-services, is lengthy and repetitive even for advanced users. Some of this complexity can be attributed to **ros_control** and **ros_controllers** packages which form the backbone of control via ROS. Even after a successful bridge between ROS and Gazebo has been established, joint control for connected bodies requires extra steps since the joints must be controlled independently using messages and services. There are of course ways to simplify the segregation of joint controllers by using wrappers, such as the Gazebo plug-in for da Vinci Surgical Robot [40]. While this might not pose an issue for simpler robots with a limited number of joints, it creates unnecessary complexity for real-world surgical robots. A general comparison between URDF and SDF is presented in Table 4.1.

¹<https://www.json.org/>

²<https://yaml.org/>

Table 4.1: Basic Comparison Between URDF and SDF

URDF	SDF
ROS uses URDF	Gazebo uses SDF
XML Markup Specification	XML Markup Specification
Each link defined w.r.t. previous joint	Each link specified in world frame
Each joint defined w.r.t. it parent links frame	Each joint defined w.r.t. child link frame
Does not support close loop interconnection	Sort of supports closed loop kinematics
Joint and link dynamics are optional	Joint and link dynamics are mandatory
URDF supports Xacro (an XML Macro)	No Macro Support
Conversion to SDF trivial	Conversion to URDF not trivial (sometimes not even possible)

4.3 The AMBF Description Format (ADF)

Based on the limitations of the robot description formats, and consequently, robot simulators elaborated in the introduction to this chapter, the following metrics are outlined for the proposed Asynchronous Multi-Body Framework Format (ADF):

- **Human Readability:** One of the design motivations behind the Asynchronous Multi-Body Framework Format (ADF or AMBF description file) is human readability, and consequently modification by hand. ADF's design philosophy places robot description at the front-end for creating, modifying and distributed testing of multi-bodies.
- **Distributed Structure:** All the relevant data for a single body/constraint/environmental object should be contained in the relevant definition block. Removal of the data block should not affect any other body/constraint.
- **Constraint Definition:** A body could have multiple constraints (joints), and each constraint is defined independently of other constraints. The addi-

tion/removal of a constraint should not alter any other constraint except for the physical/dynamic implications.

- **Controllability:** In this context, controllability refers to the ability to apply forces on the body internally or externally from the running simulation independent of the other bodies. The connected bodies react passively based on the type of constraint they share.
- **Communicability:** This refers to the ability to relay information about all aspects of every dynamic body independent from each other. This information can include the constraints this body forms with all of its connected bodies but not necessarily the information of bodies themselves.
- **Dynamic Loading and Unloading:** This defines the ability to add/remove bodies at run-time and even define constraints between newly added bodies with existing bodies.

4.3.1 Anatomy of ADF

The AMBF simulator was designed around the ADF to demonstrate its capabilities. The AMBF simulator uses several external packages that include Bullet Physics [34] and CHAI-3D [72]. The types of data in the ADF can be separated into various types that include World Data, Rigid Body Data, Soft-Body Data, Constraint Data, Lighting Data, Camera (View-port) Data and Input Device Data. The flexibility of the ADF allows not only for the definition of multiple robots and multi-bodies in one description file but also for the separation of a single robot/multi-body in multiple description files, which is in line with the *Distributed Definition* metric. As an implementation example, all the body data for one robot can be defined in

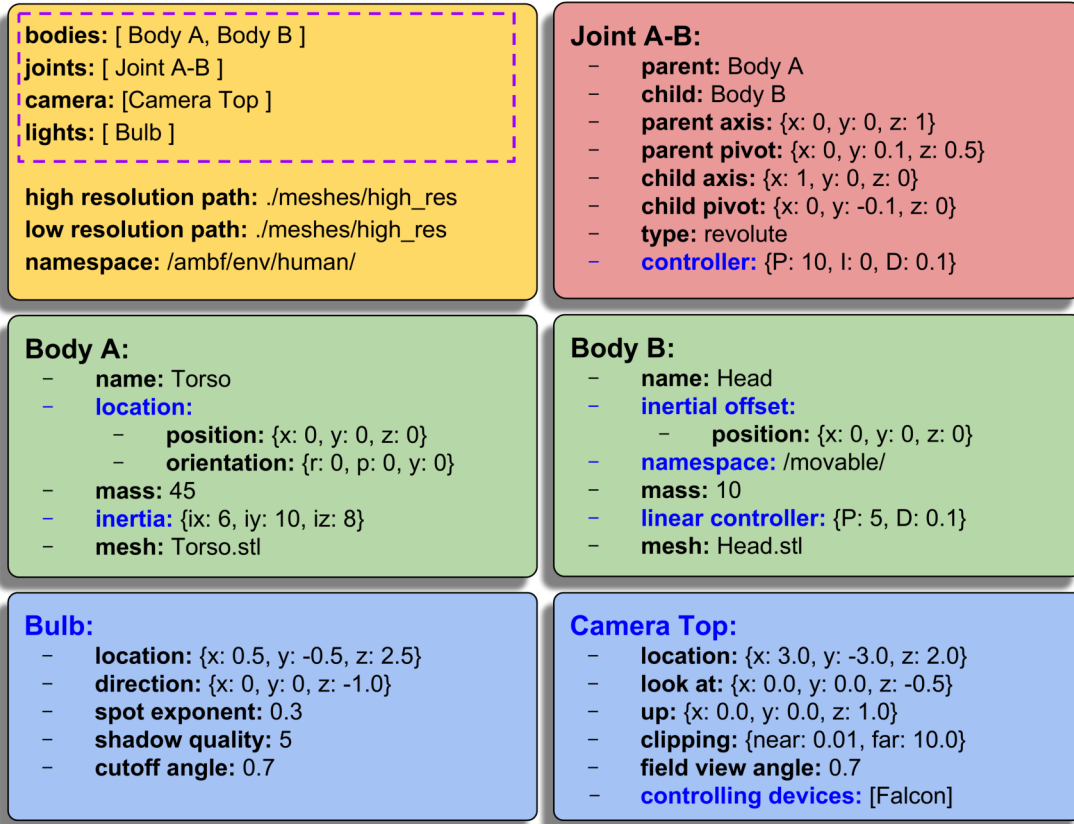


Figure 4.1: The anatomy of ADF. The yellow tile forms the header and consists of global parameters and header lists which are highlighted with the purple dotted border. The red tile represents a constraint, green represents bodies and blue represents scene objects. The blue text highlights optional parameters.

one or more description files, whereas the constraint (joint) data can be placed in a separate file(s). The ADF files are written based on the ADF. Figure 4.1 outlines the components of the ADF file (placed in tiles for emphasis but are written sequentially). The contents of the yellow tile are placed at the top and consist of global parameters applicable to the rest of the description file. Debugging robot/multi-body models by ignoring certain sub-components of the model is often an overlooked and understated design feature of robot description formats. Commenting out parts of the robot description is helpful, not only for debugging but also for testing sub-components of a model in isolation. To ignore certain objects from loading in URDF or SDF, the required object's description spanning several lines needs to

be commented out. AMBF's design specification uses header lists (emphasized by the dotted purple border in the yellow tile in Figure 4.1). The header lists are the entry point of the document such that bodies, visual elements and constraints are processed based on the content of these lists. Instead of having to comment out multiple lines of object data, it is sufficient to remove the object from the header list of its type. The ignored description block does not affect the loading of any other body or constraint since the AMBF simulator, it's derivatives and the ADF are implemented while considering the *Distributed Definition*, *Constraint Handling* and *Dynamic Loading* specifications.

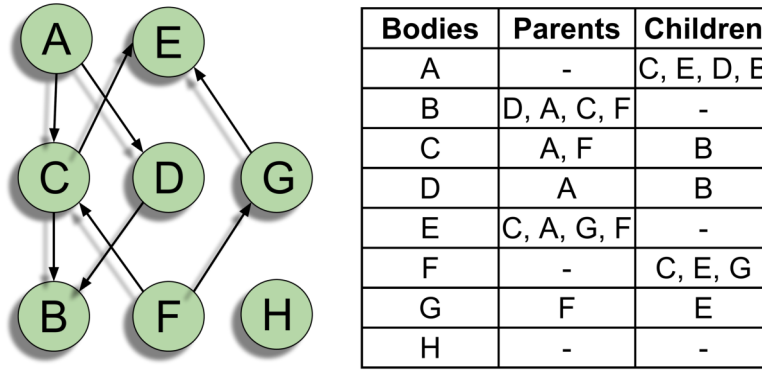


Figure 4.2: Densely connected bodies with the corresponding lineage for each body shown on the right.

4.3.2 Interconnected Bodies

To generate non-connected, semi-connected or densely connected bodies, a combination of a graph network and a densely interconnected tree structure is employed. Figure 4.2 illustrates an example of this composite structure. Unlike other formats where the parent refers to the immediate predecessor body, this requirement is relaxed by classifying all the predecessors of the body as its parents. While the relaxation of such parent hierarchy might seem counter-intuitive in traditional robot

representation formats, this relaxation is essential to meet the defined metrics of AMBF.

The Table in Figure 4.2 shows the resulting population of each body’s lineage. It should be noted that the lineage path from one body to another may lead from multiple routes as is the case between bodies ($A \rightarrow B$) and ($F \rightarrow E$). In such cases, adding children/parents redundantly to a body’s lineage is restricted. To achieve the fully connected tree, an upward and a downward pass for each added constraint is used. Algorithm 1 sums the process of adding a constraint. At the end of all passes, each object maintains references to all the successor joints while all the children register references to all the predecessor bodies. It is important to note that in the case of diverging leaf nodes at a specific body, the predecessor bodies contain the references to all the children in every leaf node, however, the successors in leaf nodes are unaware of bodies in other leaves.

4.3.3 Convention of Constraint Definition

Constraints are used to connect two bodies in certain ways that limit their relative motion. In robot applications, the constraints can broadly be classified into two foundational types, the rotational constraint (revolute and hinge) and the translational constraint (prismatic and slider). Other constraints such as springs, cams, gears and 6 DOF joints can be built with the combination of foundational types. Fixed constraints present a special case, but can also be implemented with either of the foundational types.

In alignment with the design philosophy of the Asynchronous Framework, constraints are defined in a slightly different manner as compared to URDF or SDF. In URDF, the joint is treated as the origin of the child body or vice-versa, and two additional fields are used to set the offset of the child body’s visual mesh and the col-

Algorithm 1 Add Constraint Algorithm

```
1: function ADD JOINT(joint, parent, child)
2:    $p := \text{parent}, c := \text{child}$ 
3:    $c.Parents \leftarrow p, p.Children \leftarrow c, p.Joints \leftarrow \text{joint}$ 
4:   UpwardTreePass(p)
5:   DownwardTreePass(c)
6: end function
7: function UPWARD TREE PASS(body)
8:    $P := \text{body}.Parents, C := \text{body}.Children$ 
9:   for  $p \in P$  do
10:    for  $c \in C$  do
11:       $p.Children \cup C \ \& \ p.Joints \cup c.Joints$ 
12:    end for
13:  end for
14: end function
15: function DOWNWARD TREE PASS(body)
16:    $P := \text{body}.Parents, C := \text{body}.Children$ 
17:   for  $c \in C$  do
18:    for  $p \in P$  do
19:       $c.Parents \cup p.Parents$ 
20:    end for
21:  end for
22: end function
```

lision mesh. Furthermore, these visual and collision offsets are defined in the body's definition, while the child origin (joint origin) is defined in the joint description. This distribution of data breaks the Asynchronous Design, since, to get a complete specification of the interconnection between two bodies, it is necessary to parse the data beyond what is just defined in the constraint description.

In the AMBF Constraint definition, a body's origin is always treated as the base frame of its representative mesh. The way AMBF's constraint definition differs from URDF or SDF is by treating the constraint origin as independent of the child's or parent's body origin. As a result, two fields - namely pivot and axis - are used for the parent and the child. The pivot defines the location of the constraint from the body's origin in Cartesian space, and the axis defines the free axis in the body's frame. This convention requires less parameters to fully define an interconnection ($13 = [\text{parent's pivot (3)} + \text{parent's axis (3)} + \text{child's pivot (3)} + \text{child's axis (3)} + \text{offset (1)}]$) as compared to URDF or SDF ($15 = [\text{joint's XYZ (3)} + \text{joints's RPY (3)} + \text{joints's axis (3)} + \text{child's offset XYZ (3)} + \text{child's offset RPY (3)}]$). While this description is sufficient in constraining the two bodies, an extra scalar parameter is required to define the rotational offset along the parent axis between the two bodies. This offset is discussed in detail in Section 4.4.2. The best part about the way ADF defines a constraint is that one can easily switch the parent and child definition (name, pivot and axis) and the joints will stay where it is. This is handy for complex interconnected robots where it is difficult to determine who the parent/child is.

The direct use of parent/child axes to build constraints emphasizes the front-end nature of ADF, which consequently makes the specification of robots and multi-bodies easier. This, however, adds more work at the back-end where the constraints are actually parsed and processed. Internally, joint transforms w.r.t. the parent

body and child transforms w.r.t. to the joint have to be computed. A unified convention to define the rotation represented by axes in the parent/child body frame is required. This convention utilizes the plane formed by the two axes ($\vec{a}x_p$ and $\vec{a}x_c$) to define a rotation matrix. This rotation is trivial except for the case where the two axes are parallel to each other since there exist an infinite number of rotation planes. To address such cases, Algorithm 2 is adopted across the AMBF Framework, AMBF Simulator, Blender-to-AMBF add-on (4.4.2) and the URDF-to-AMBF converter (4.4.1). In the Algorithm, S denotes the Skew-Symmetric matrix, $\vec{a} \times \vec{b}$ denotes the vector cross product, $I_{3 \times 3}$ is the 3×3 Identity matrix and \vec{n}_x , \vec{n}_y , \vec{n}_z are the three unit vectors.

Algorithm 2 Convention for Rotation Between Two Vectors

```

1:  $\vec{a} = \vec{a}/\|\vec{a}\|$ 
2:  $\vec{b} = \vec{b}/\|\vec{b}\|$ 
3: if  $abs(\vec{a} \cdot \vec{b}) \simeq 1$  then  $R_{\vec{a}}^{\vec{b}} = I_{3 \times 3}$ 
4: else if  $\vec{a} \parallel \vec{b}$  then  $\triangleright \vec{a}$  is not parallel to  $\vec{b}$ 
5:    $R_{\vec{a}}^{\vec{b}} = I_{3 \times 3} + S(\vec{a} \times \vec{b}) + S(\vec{a} \times \vec{b})^2(1 - \vec{a} \cdot \vec{b})/(\vec{a} \times \vec{b})^2$ 
6: else if  $\vec{a} \parallel \vec{n}_x$  then  $R_{\vec{a}}^{\vec{b}} = AxisAngle(\vec{a} \times \vec{n}_x, \pi)$ 
7: else  $R_{\vec{a}}^{\vec{b}} = AxisAngle(\vec{a} \times \vec{n}_y, \pi)$ 
8: end if

```

4.3.4 Flexibility of Name-spacing and Resource Paths

The foundational structure of ADF allows for the use of multiple namespaces in a single description file. This is accomplished by overriding the description file's global namespace with the local name-space parameter in the respective body(ies) as shown for **Body B** in Figure 4.1. Name-spacing is not required for joints as their parents and children are searched in all the listed name-spaces. This feature of the AMBF not only allows multiple robots and multi-bodies to exist in one description file but also the ability to create different name-spacing for sub-structures of a single robot.

Table 4.2: Simplifying redundant names using name-spaces rather than suffixes

URDF & SDF	ADF
/body/limb_⟨left right⟩	/body/⟨left right⟩/limb
/box_⟨one two⟩_lid_⟨top down⟩	/⟨one two⟩/box/⟨top down⟩/lid

One practical example is shown in Table 4.2 where identical bodies are distinguished by name-spaces rather than the addition of suffixes to their names. Among other advantages, this allows for the convenience of disseminating distributed controllers using name-spaces rather than breaking down the link names.

A redundant aspect of URDF or SDF is the specification of resource paths as it is often the case that a robot’s visual and collision meshes are located in a single OS directory. However, a qualified path for the mesh needs to be defined for each link. This is somewhat simplified by the use of “package” or “model” tags as base names, which are resolved to the base folder of a “package” or the “.gazebo/model” folder respectively. The ADF simplifies this by separating the mesh’s name from its path. Towards this end, two global resource paths are defined in the ADF’s header shown in Figure 4.1 (for visual and collision geometry). Similar to the global name-space parameter, the mesh resource paths can be overridden locally in the body’s description, thus allowing multiple paths in a single description file. Additionally, the mesh path can either be relative or absolute. This greatly improves the readability and manageability of the ADF files.

4.3.5 Resolving Naming Conflicts

As of now, there are three different ways of spawning ADF files into the AMBF simulator, these include:

- **Using Launch File:**

Appending the desired ADF file path to the base launch file and using the

corresponding index of the appended file to start the simulator.

- **CLI Specification**

Using a CLI argument $-a$ and then adding to that flag, the file-path of the desired ADF file.

- **Drag and Drop**

Dragging and dropping the ADF file into a running instance of the simulator.

Using all the above three ways of spawning an ADF file, multiple different files can be loaded at the same time. Even the same file can be loaded multiple times. Since the ADF is designed to support a distributed definition of bodies and joints, a naming conflict may occur among different files or will occur while launching the same file multiple times. Although the naming conflict can be avoided by altering the global or local name-space parameters in ADF files or expecting the user to prevent any name repetitions, nonetheless, the AMBF Framework is integrated with a safety mechanism to detect naming conflict, re-assign names systematically, resolve joint look-ups and load communication plugins without breaking the running instance of the simulation.

For this purpose, if an ADF file contains a fully defined object name (body / camera / light or sensor name) that already exists in the graph, the newly added object is appended with a renaming index starting at 1. Adding objects with identical names keeps increasing the renaming index. It is worth noting that a mechanism for rectifying naming conflicts, for objects specified at random times, is more cumbersome as compared to batch copy/pasting and renaming, as it includes breaking down the object names as strings, and then finding if the name contains any trailing characters representing the ASCII character for digits and if a number exists, this number is incremented by 1 and appended to the name of the newly added object.

On the other hand, if a trailing number doesn't exist, the renaming index is set to 1 and appended to the newly added object name. This newly added name is also reflected in the communication instance of the object.

Since a multi-body can be divided into multiple description files a joint (constraint) can have an interconnection between two bodies that do not exist in the same file. While this makes the ADF files shorter, simpler and manageable, it introduces complexity in case a naming conflict between two objects (bodies) results in the renaming of the original bodies, as any joint looking to connect to the bodies with the original name might fail. For this reason, for each joint, the parent and child are first searched in the local scope. The local scope refers to all the objects defined in the current description file. If an object with the required name is not found, it is searched for in the global scope. Global scope refers to all the objects that have been added to the graph until that moment.

This still doesn't address the case where identical ADF files (containing both bodies and joints) are loaded in succession. The first ADF file will result in bodies being spawned at first and eventually the joints are loaded, connecting these bodies. When the second ADF file is loaded, all the bodies will be renamed by appending a "1" at the end. Now the joints of this newly added ADF file will already have been defined by the previous file, so these joints will also be renamed by appending "1". The appended character is now used in combination with the parent and children names to search in the local scope thereby resulting in the correct interconnection between the newly added bodies.

In case the second added ADF was not identical to the first ADF file, i.e. all the bodies and most of the joints were of different names except some joints with the identical names to joints in the previous ADF file, then after the first local search with the appended "1", which will fail, the search is repeated in the global scope

with all the appended combinations up til the “renaming_idx”. This will allow these joints to correctly connect to the bodies in the first ADF file. If one wants to connect to a prior body belonging to the group of renamed bodies, they should explicitly provide the correct name along with the renamed index in the ADF file.

Although this safety mechanism is cumbersome to explain and equally complex implementation wise, it makes it possible to quickly copy, paste and spawn multiple robots and environments in the AMBF simulator. More importantly, this allows specialized robots to have changeable tools and end-effectors while the simulation is running.

4.3.6 Support for Soft Bodies

The ADF provides support for soft bodies in addition to rigid bodies. Soft-bodies are defined as almost identically to rigid bodies except for additional solver data. The AMBF simulator uses the Bullet’s soft-body solvers for simulating the interaction. The discussion of soft-body support deserves a lengthy discussion and for this reason, it is presented in Chapter 5.

4.3.7 Action Based Sensors for Reusability

An interactive training simulation involves several forms of interactions that result in the change of state of the interacted objects. For example, a user may interact with a door or a latch resulting in the state of the door being “open” or “closed” or neither. Likewise, in the context of user-training, the task may involve solving a puzzle, and a sub-task is considered successful if the correct puzzle piece is placed in the desired position (and/or orientation). In these examples, the outcome of the interactive task is a boolean flag which indicates “success” and “failure” or simply “true” and “false”.

For the specific examples, the problem here is to determine if the door is open or closed and whether the puzzle piece is sitting in the right place. These problems can, of course, be solved using brute force methods. In the example of the door open/close, a brute force method may be simply testing the joint angle θ_{hinge} of the door hinge and explicitly defining whether the angle is between the threshold for a door being open or closed $\theta_{correct}$. Likewise, in the example of puzzle placement, the transform of the puzzle base T_b^w and the puzzle piece T_p^w need to be calculated to finally calculate the transform between the piece and base T_b^p . Afterward, a predetermined transform $T_{correct}$ needs to be defined, which is compared to T_b^p to find out if the puzzle piece sits where it needs to. While these problems are easier to solve, they require the explicit programming and calculation of correct $T_{correct}$ and $\theta_{correct}$ for each door and puzzle piece. One can envision using dynamically placeable simulated sensors to address these problems without explicit programming. These sensors, similar to their real-world counterparts, trigger based on either proximity or contact and output a boolean state as well as the ID or name of the triggering bodies. For the puzzle-solving task, pairs of sensors can be used, each for the puzzle piece and the corresponding location on the puzzle base. These sensors can then be assigned a matching key, to identify if the correct piece sits in the right place.

For more involved problems, one may require the output of sensors to be actions instead of just a boolean state. This comes in handy for manipulation problems, such as affixing the desired body to an SDE with a closed grip once the body is between the jaws. It can be easily argued that the sensor-based approach makes this problem generic to solve as compared to the brute force approaches for each different type of SDE. Similarly, more challenging problems like cutting soft-bodies using simulated knives and blades can be approached using sensor-based approaches even if the initial implementation is more challenging. In literature, the term “sensactors” [101] or

“sensoriactuators” [102] have been used for action based sensors.

The goal of an action-based sensors approach is to solve these problems by first reformulating them, and then classifying them into smaller generic sub-tasks. The first challenge lies in determining whether an event has taken place in the context of the sub-task. This can be accomplished by using different types of sensing elements. One such sensor is implemented by utilizing the ray-tracing algorithm which can be used to calculate range, scan areas, detect contacts and determine the information of the triggering body(s). This multipurpose sensor can be conveniently used for implementing contact-based grasping, cutting, magnetic constraints and contact validation for puzzle design.

The design and implementation of these sensors are inspired by actual sensors used in robotics. The advantage of using such a formulation is that sensors are defined in the front end ADF. This description includes the type of sensor and also what object are these sensors parented to. This makes it convenient to develop training simulations with reusable elements rather than explicit programming of all scenarios. A generic grasping methodology using these sensors is presented in Chapter 6.

4.3.8 Auto Generation of Communication Instances

Unlike other robot dynamics simulators, the AMBF simulator does not require any intermediate steps to prepare for bidirectional communication. The bodies defined in the ADF are designed to satisfy the *communicability* and *controlability* requirement and spawn instantaneously after the relevant ADF file is loaded. Each body initiates a thread for its bidirectional communication using an Inter-Process Communication (IPC) medium (via ROS topics). The outgoing communication provides information about the body’s state and is conveniently called the `afState` message, while the

incoming message is called `afCommand`. Unlike multiple communication instances for bodies, there is a single instance for the world's states/commands. The payloads for these communication instances have already been discussed in Section 3.7.2.

4.4 Compatibility of ADF with External Software

4.4.1 URDF to ADF Conversion

A significant number of robot models haven already been defined using the URDF format, and arguably, newer robots would continue to be represented in URDF. To take advantage of the existing work and community support for the URDF, a URDF-to-AMBF converter has been developed in parallel with the design of the ADF and simulator. The source code of this converter is available at [103]. The converter uses internally implemented XML parsing to reduce the reliance on external ROS parsing packages for portability outside Linux operating systems. As mentioned in the previous sections, URDF is constrained by design to limit the links to a single parent. From a design point of view, this deadlock is enforced by the use of visual and collision offset data in the link description. These offsets are taken from the joint frames of relevant links. For the ADF, this visual offset data is used in conjunction with joint data to develop AMBF constraints based on Algorithm 3.

Algorithm 3 URDF Joint to AMBF Joint

- 1: **if** *JointType* := *Fixed* **then** $\vec{a}\vec{x}_j = \vec{n}_z$ **else** $\vec{a}\vec{x}_j = \text{joint.Axis}$
 - 2: $T_j^{pv} = (T_{pv}^p)^{-1} * T_j^p$ $\triangleright pv = \text{ParentVisual}, p = \text{Parent}$
 - 3: $pvt_p = P_j^{pv}, \vec{a}\vec{x}_p = R_j^{pv} * \vec{a}\vec{x}_j$ $\triangleright j = \text{Joint}, ax = \text{axis}$
 - 4: $pvt_c = P_j^{cv}, \vec{a}\vec{x}_c = P_j^{cv} * \vec{a}\vec{x}_j$ $\triangleright pvt = \text{Pivot}$
 - 5: $ambfR_c^p = \text{RotBetweenVectors}(\vec{a}\vec{x}_c, \vec{a}\vec{x}_p)$
 - 6: $urdfR_c^p = (R_{pv}^p)^{-1} * R_j^p * R_{cv}^c$
 - 7: $R_{jo} = (ambfR_c^p)^{-1} * urdfR_c^p$ $\triangleright jo = \text{JointOffset}$
 - 8: $\vec{a}\vec{x}_{jo}, \theta_{jo} = \text{toAxisAngle}(R_{jo})$
-

4.4.2 Blender ADF Addon

The default simulator for ROS (RViz) does not have the capabilities to generate robot models. The most recent versions of Gazebo provide limited support for generating robot models, but its interface is experimental. Hence, SDF files are often generated using URDF through script converters. URDF files can be created using Solidworks (Solidworks Corp., MA, USA) via **Solidworks2URDF** converter [104]. This versatile converter has been in active development and the tool of choice for anyone creating URDFs without handling XML by hand. While ROS and its derivatives are designed to be free for research purposes, Solidworks is not. Not only that, Solidworks lacks support for Linux, which is the OS of choice for ROS related development. It is worth mentioning that the Solidworks2URDF converter lacks bidirectional support in Solidworks (i.e., the generated URDF file cannot be reused to load the corresponding Solidworks assembly). Arguably, this can be attributed to the restrictions posed by Solidwork’s plugin API rather than the converter itself.

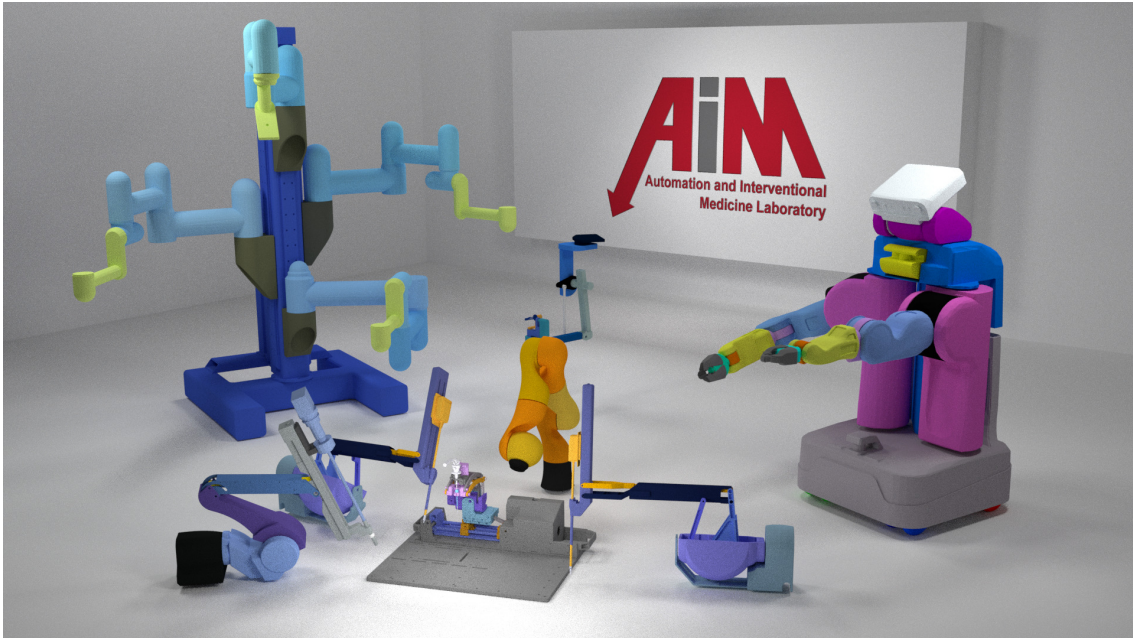


Figure 4.3: A subset of robot models already implemented for the AMBF simulator in Blender. These robots include the da Vinci Surgical Robot with multiple parallel mechanisms.

Even though the ADF has a front-end interface to allow for the easy creation of simple robots and mechanisms, a graphical user interface is always helpful in fine-tuning and creating complex robots and multi-bodies. Existing software that can be leveraged for this purpose was sought out. A few specifications are outlined for the selection of the corresponding software, which includes a “free to share” license, bidirectional API to generate and load models, community support and optionally Linux portability. Based on these specifications, Blender [98] is selected as the graphical interface for creating ADF files (Figure 4.3). Notably, the overall user interface of Blender might offer a relatively steeper learning curve to users unfamiliar with animation software.

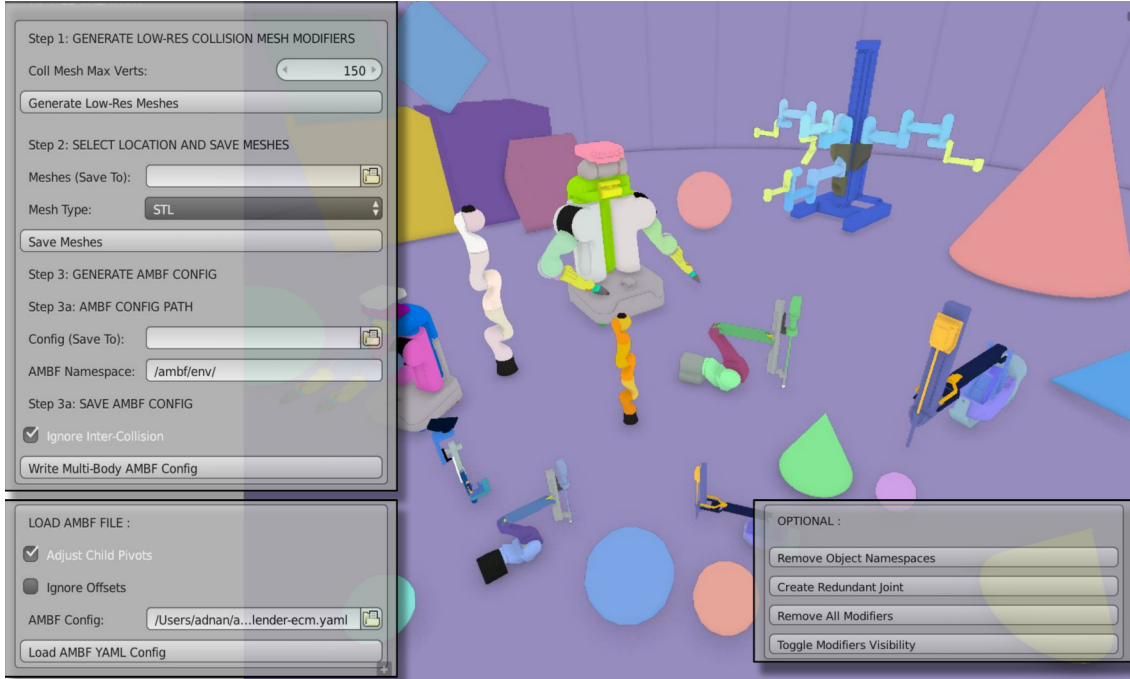


Figure 4.4: A few features of the Blender-to-AMBF add-on include copy pasting robot models, scaling, altering the pose of any subset of robots/links, visually setting constraints and inertial properties, creating collision meshes and generating/loading created ADF files.

While Blender enjoys huge community support for graphic designers and hence offers extensive features for such, it has not primarily been used for modeling dexterous robots and bodies with a significant number of interconnected constraints.

Blender includes basic support for Bullet Physics for defining constraints and rigid bodies. This support has been leveraged to create a plugin for generating and loading ADF files (Figure 4.4). To include bidirectional usage with Blender, a few simplifications of the ADF are required which are addressed in the following subsections. The source code for Blender-to-AMBF add-on is available at [105].

Loading ADFs

Child body's n_z and n_x are the default constraint axis for rotational and translational joints respectively in both Blender and Bullet Physics, while ADF and simulator do not impose this limitation. To enable the same model to be circularly compatible with Blender, the Blender-to-AMBF add-on provides the necessary functionality to alter the multi-body description by adjusting for child body pivots and axis. Adjusting a body axis and pivot is not trivial as all the successor bodies must be accounted for. Since the design philosophy of AMBF separates constraints from bodies, all the body data (meshes) are imported first followed by joints, which in turn connect bodies and enforce world transforms. The pivot and axis correction involves two algorithms which are necessary to make sure that the entire connected structure is bidirectionally compatible:

Algorithm 4 Adjust and Store Child Offsets

```

1: if  $JointType := Rotational$  then
2:    $\vec{ax}_j = \vec{n}_z$ 
3: else  $JointType := Translational$ 
4:    $\vec{ax}_j = \vec{n}_x$ 
5: end if
6:  $R_j^{c^{adj}} \leftarrow RotBetweenVectors(\vec{ax}_j, \vec{ax}_c)$ 
7:  $T_j^{c^{adj}} := [R_j^c, \vec{pvt}_c]$   $\triangleright adj = Adjusted$ 
8:  $ApplyMeshOffset(T_j^{c^{adj}})$   $\triangleright c = Child, pvt = Pivot$ 
9:  $BodyOffsetMap[child] \leftarrow T_j^{c^{adj}}$ 

```

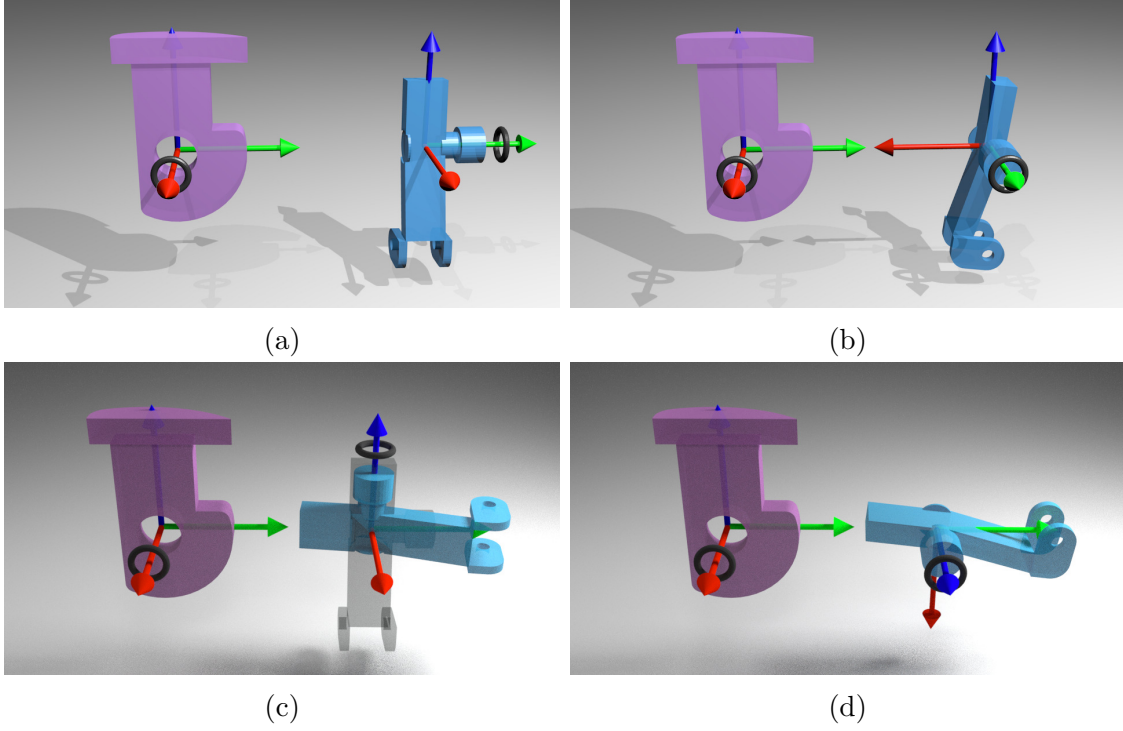


Figure 4.5: In the sub-figures, the purple and turquoise bodies represent the parent and child with the constraint axes marked with the black ring. In (b), the child body is rotated to form a constraint by aligning \vec{a}_p and \vec{a}_c . (c) shows the adjustment required in Blender such that the child body is rotated to adjust the constraint axis to default \vec{n}_z followed by (d) to align the constraint axes with parent's axis.

Figure 4.5 shows a parent (purple), and a child (turquoise) and the corresponding joint axes marked with the black rings for a rotational joint. To create the constraint, the child's axis is aligned with the parent using Algorithm 2 as shown in Figure 4.5 (b). As illustrated in the Figure, the child's constraint axis (\vec{n}_y) is different from the default axis for rotational constraint type (\vec{n}_z). Algorithm 4 is performed iteratively for each constraint before the final Algorithm 6 is performed. The goal here is to offset the body meshes such that the child pivots can be $\vec{0}$ and the child axis is set to the default constraint axis. While performing these mesh offset operations, the corresponding imparted offsets were kept track of so that they can be used later in Algorithm 6. These offsets are stored in a map ($f : body \rightarrow T_j^{c^{adj}}$) where the superscript $T_j^{c^{adj}}$ reiterates that the offset is applied to all bodies when considered

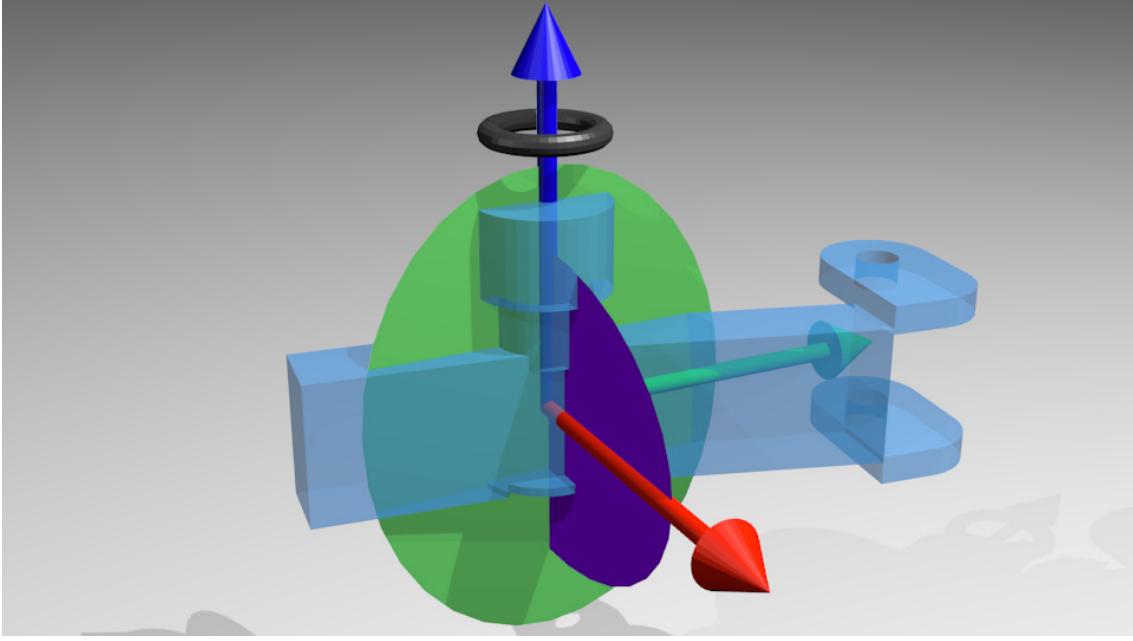


Figure 4.6: A visual representation of plane offset between the plane formed by shortest angle rotation between parent's and child's constraint axes (purple disk) and the rotation plane of correction axis (green disk).

as children, however, they are used in Algorithm 6 when treated as parents.

The nature of representing joint data using pivot/axis notation and the correction, thereby using Algorithm 4, can result in axis misalignment along the constraint axis. This misalignment occurs when the rotation due to offset correction occurs outside the plane of rotation between the parent's and child's body axes. The misalignment is best explained by Figure 4.6. To account for the imparted axis misalignment, Algorithm 5 is performed before Algorithm 6.

After adjusting for the child body offset and axis alignment, the final step is loading all constraints from the AMBF, and consequently, assigning the correct body poses, and eventually, parenting the bodies. This can be summed up with 5 transformations applied iteratively for each constraint to the respective bodies in Algorithm 6.

Algorithm 5 Axis Alignment

```

1:  $R_j^p = RotBetweenVectors(\vec{a}\vec{x}_j, \vec{a}\vec{x}_p)$ 
2:  $R_c^{p^{adj}} = R_j^p R_c^{j^{adj}}$ 
3:  $R_c^p = RotBetweenVectors(\vec{a}\vec{x}_c, \vec{a}\vec{x}_p)$ 
4:  $R_{off} = (R_c^{p^{adj}})^{-1} * R_c^p$ 
5:  $\vec{a}\vec{x}_{off}, \theta_{off} = ToAxisAngle(R_{off})$ 
6: if  $\theta_{off} > \epsilon$  then
7:    $R_{ao} = FromAxisAngle(\vec{a}\vec{x}_c, \theta_{off})$ 
8:    $BodyOffsetMap[child] \leftarrow R_{ao} T_j^{c^{adj}}$ 
9: else
10:   $BodyOffsetMap[child] \leftarrow I_{4 \times 4}$ 
11: end if

```

Algorithm 6 Pose Data from ADF

```

1: if  $JointType := Rotational$  then
2:    $\vec{a}\vec{x}_j = \vec{n}_z$ 
3: else  $JointType := Translational$ 
4:    $\vec{a}\vec{x}_j = \vec{n}_x$ 
5: end if
6:  $T_p^w \leftarrow$  Parent Body's Pose in World
7:  $T_{bo} \leftarrow BodyOffsetMap[parent]$ 
8:  $T_j^p = [I_{3 \times 3}, \vec{pvt}_p]$ 
9:  $T_{jo} = [R_{jo}, 0]$ 
10:  $T_c^j = [R_c^j, 0]$ 
11:  $T_c^w = T_p^w T_{bo} T_j^p T_{jo} T_c^j$ 

```

$\triangleright R_{jo} = FromAxisAngle(\vec{a}\vec{x}_j, \theta_{jo})$
 $\triangleright R_c^j = RotBetweenVectors(\vec{a}\vec{x}_c, \vec{a}\vec{x}_j)$

Support for Detached Joints

As stated in the previous sections, an important goal of the ADF is to support closed-loop mechanisms and parallel linkages for robots in an easy manner. While this is conveniently achieved using the front-end syntax of ADF itself, necessary means to achieve this have been provided using the graphical interface of Blender via Blender-to-AMBF add-on. Blender does not support multiple parents for an object, which is necessary for closed-loop mechanisms. To circumvent this, an empty frame is used with a specific prefix in its name. This empty frame is then used to define a constraint by defining a parent and child body. While parsing through the bodies and constraints in the Blender scene to generate an ADF file, the assigned naming prefix is leveraged to treat the empty frame as a place holder rather than an actual empty body. This allows the robust creation of densely connected bodies without having to manually touch up the ADF file.

4.4.3 Implementation of Multiple View-ports using Camera Data

One of the design requirements of the AMBF framework and the AMBF simulator is the ability to manipulate multi-manual tasks in the real-time dynamic simulation with haptic feedback. The design goal enables multiple users alongside AI to share a simulation via haptic/input devices. To this end, having multi-port frame buffers can assist the users in performing tasks within their respective view-ports. Additionally, the users should potentially possess the camera control for their view-port independent of the other users in the simulation. As a result, the prospective design of the ADF includes support for achieving multiple view-ports and binding input devices to each (described in Figure 4.1 for the camera tile). Figure 4.7 shows the

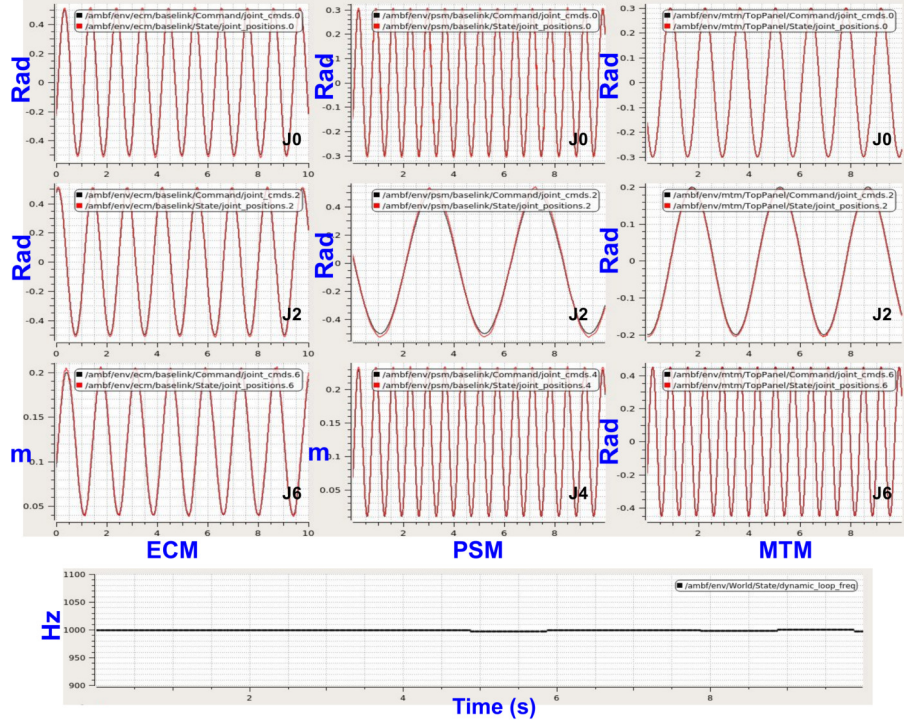


Figure 4.9: Each column shows the joint control of a different manipulator labeled underneath. The last row shows the dynamic update frequency of physics simulation. The ECM's and PSM's 3rd graph depicts a translational joint while all the joints of the MTM are rotational.

4.5 Results and Discussion

The same PC setup from Section 3.9 was used for the results in this section. The controller performance of multiple closed-loop robots is demonstrated in the simulation environment using ROS communication as IPC. The robots shown in Figure 4.8 are commanded at 1 kHz. The joints are controlled in position control mode and labeled in Figure 4.9. The inertial parameters of the PSM and MTM have been computed by Yan et.al [42] and can be utilized in the ADF files. As shown in Figure 4.9, the joint response at 1 kHz of communication frequency remains stable and robust.

The AMBF simulator was designed to reduce computational overheads and enable efficient loading and unloading of models. This also assists in the work-flow of

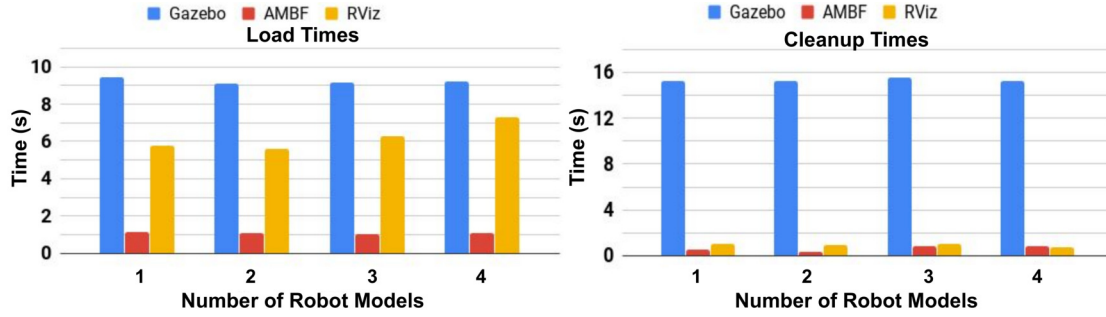


Figure 4.10: The loading vs unloading times for simulators with increasing number of complex robot models. The simulators are loaded using the bash terminal.

developing a multi-body representation using Blender-to-AMBF add-on and quickly loading it in the AMBF Simulator. The loading times of multiple robots in parallel with multiple closed-loop constraints are presented and compared with Gazebo and RViz using similar models and identical system load. As evident from Figure 4.10, not only does the AMBF simulator outperform Gazebo and RViz in terms of loading speed and controller performance, it also outperforms in the cleanup speed.

Another demonstration of a complex robot (WPI’s Neuro Robot [106]) and its controller performance is shown in Figure 4.11. The URDF description of the robot was developed at [107] and is converted using the URDF-to-ADF converter. The AMBF model is then loaded in Blender using the Blender-to-ADF add-on to create parallel linkages and then adding visual details and colors to the robot. Using the IPC controllers, various joints of the robot have been excited to follow a different sinusoidal frequency.

Various results for speed of communication for distributed controllers as well as the real-time performance of AMBF have been shown in this Chapter. However, the advantage of AMBF over other community-based simulators goes beyond just these performance metrics. AMBF can not only dynamically load robots and environments using a distributed definition that spans multiple description files but also cleanly remove segments of interconnected mechanisms in real-time. The incurred

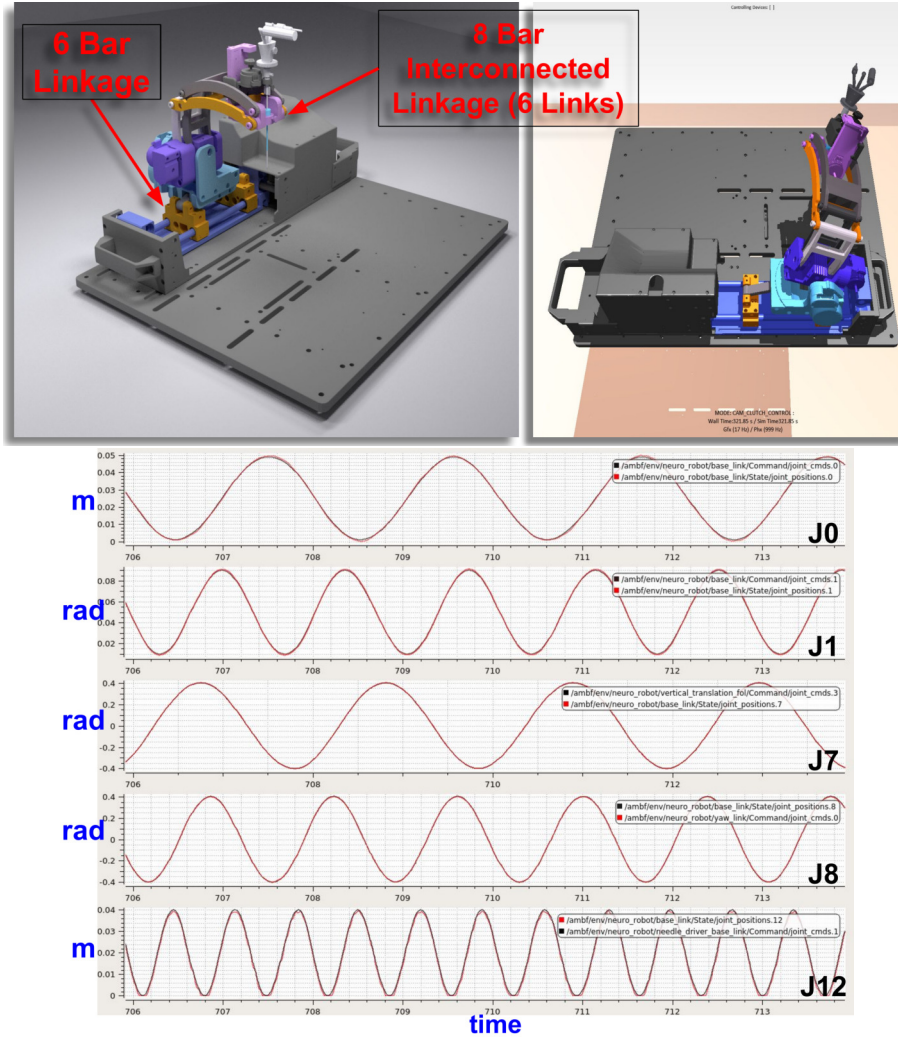


Figure 4.11: WPI's Neuro Surgery Robot Model using the Blender-to-AMBF add-on. The robot consists of a 6 bar linkage at the base and an inter-connected 8 bar linkage at the top. The robot is controlled using ROS topics at 1 kHz communication frequency.

changes to the body lineage (Section 4.2) are reflected in real-time both internally, as well as externally to the communication instances. An example of such dynamic removal is shown in Figure 4.12 where an interconnected body is removed dynamically from the running simulation. Such dynamic removal and addition are used in surgical robots that require interchangeable tools during a single procedure.

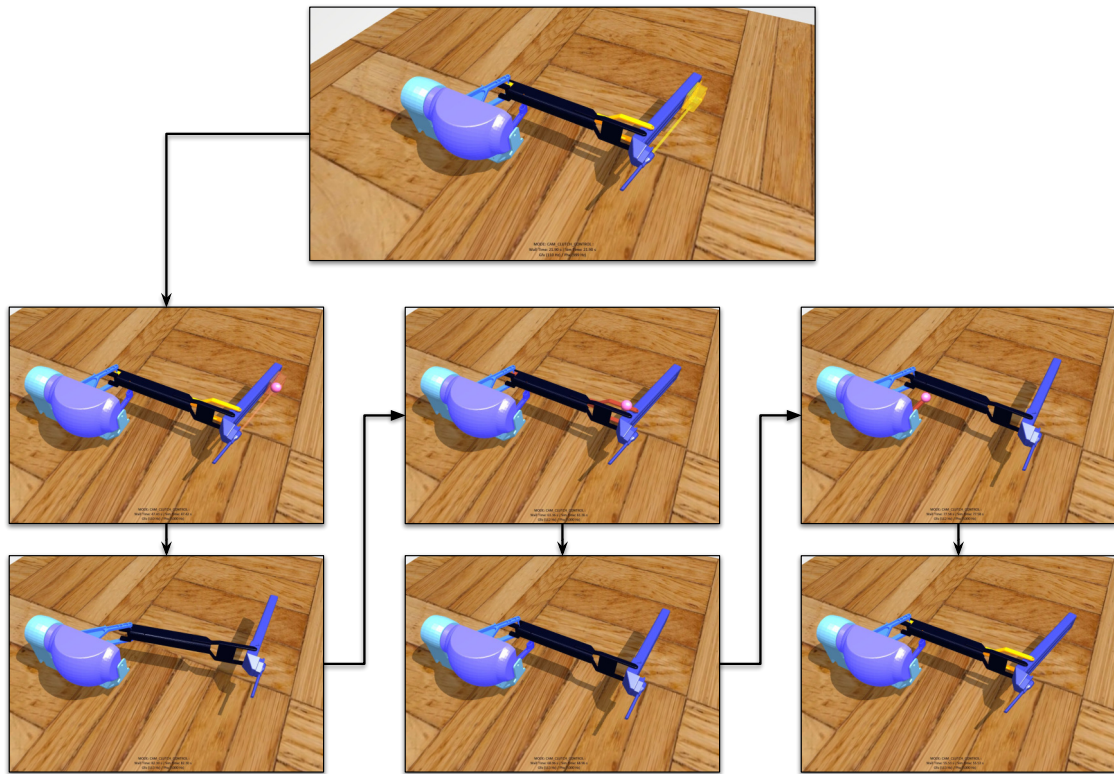


Figure 4.12: The dynamic selection and removal of links belonging to interconnected mechanism.

Chapter 5

Integration of Soft-Body Simulations

The prior chapters covered the design of the Asynchronous Framework for the inclusion of multiple IIDs, shared view-ports, a communication interface for both online and offline training and finally the associated real-time simulator with a novel front end specification format (ADF). This chapter focuses on the extension to both the ADF format as well as the AMBF simulator for allowing the development of soft-body simulations for training and control.

Section 5.3 introduces the challenges associated with the development of a general-purpose soft-body simulator. These challenges do not include the underlying complexity of solving the system of equations representing the soft-body, but the steps for easily specifying a soft-body. These challenges are addressed using a variety of tools and techniques which are presented in Section 5.5. The results and discussions for the implementation of the soft-body support in the AMBF are presented in Section 5.6 and 5.7.

5.1 Acknowledgement

The work presented in this chapter has been submitted for review as:

Munawar A, Srishankar N, Fischer G, “*An Open-Source Framework for Rapid Development of Interactive Soft-Body Simulations for Real-Time Training*” In Review for International Conference on Robotics and Automation (ICRA), 2020.

5.2 Introduction

The use of soft-body simulation has always been an area of interest for simulators targeting surgical robotics. A large proportion of research in this area has focused on solving the dynamics of tissue deformation, visual realism, and to some extent, the real-time simulations for specific tissues. On the other hand, surprisingly little work has been done, by the open-source community, towards the development of a generic framework for integrating custom soft-body simulations with real-time manipulation using input interfaces of research versions of surgical robots.

The design of ADF, among other things, allows the easy definition of robots and mechanisms designed using parallel linkages, a trait commonly employed in surgical robots, and even the specification of distributed controllers for each link and joint. To develop the Asynchronous Framework beyond just a rigid-body heterogeneous simulator, an extension to ADF that followed the original design principles covered in Section 4.3 was required.

5.3 Problem Formulation

Some of the widely focused aspects of soft-body simulations are the physical dynamics of deformation and visual realism. While these are certainly challenging aspects,

additional challenges need to be addressed to provide a generic framework. These challenges include:

1. Representation:

Representation refers to the description of the soft-body shape and dynamic parameters such as softness, stiffness, bending, and weight distribution. In addition to the geometric shape, the required number of controllable parameters for a simulated soft-body pose a challenge for description purposes. Simulators such as Gazebo [32] and V-REP [70] utilize physics computation libraries that support soft-body simulation, however, they do not support soft-body representations or their visualizations.

2. Visualization:

Visualization is more challenging as compared to representation. Unlike rigid bodies that do not require a constant update to the geometry of the mesh, soft-bodies require computationally expensive updates to the mesh at each simulation step before the shape can be pushed to rendering frame buffers. Thus the algorithm for updating the visual representation of the soft-body adds additional overhead to the simulation step. High-density meshes are preferred for visual realism, however, they are problematic for real-time soft-body simulations. For this reason, a pair of meshes can be specified: (1) a high-quality mesh for visualization and (2) a lower resolution mesh to represent the soft-body. The meshes can then be fused such that the lower resolution mesh can be used to update the vertices of the high-quality mesh.

3. Interaction and Manipulation: Interactions and manipulation of the soft-body are complex tasks that are usually specialized for specific soft-bodies.

Realistic grasping is a challenging problem in rigid body dynamics and holds even in soft-body dynamics.

4. Real-time dynamic update:

This challenge is similar to the one discussed for real-time dynamic update of rigid-bodies (Section 3.4.1), however, it can easily be argued, that achieving a real-time dynamic update for soft-body simulations is even more challenging than rigid body simulations.

5.4 Related Work

Framework level implementation of a combined real-time interactive simulator that includes both soft-body simulations and extensive rigid-body dynamics is rather limited. One possible reason is that the design choices that are usually undertaken to implement a robust rigid body framework, quite often, run contrary to the inclusion of a soft-body simulation, representation and manipulation. Regardless, one can look at other applications such as gaming for ascertaining the state of the art of soft-body applications. Maciel et al. [108] used NVIDIA’s PhysX¹ library to provide bi-manual interactivity in simulated surgical operations with implicit integration-based stepping (between 10 to 20 Hz). However, at that time, the PhysX source code was not freely available and only recently has the source code been made public (using the BSD license).

Danevičius et al. [109] worked on the gamification of a soft-body simulator that was based on the mass-particles model. To achieve real-time stepping, the graphics

¹<https://developer.nvidia.com/physx-sdk>

and physics computations were offloaded to a cloud-based system. However, this work was limited to simplistic soft objects and lacked extensive collision modeling. Tan et al. [110] worked on the simulation and control of muscle fibers using Finite-Element (FEA) methods for locomotion. The control was achieved using objective functions that controlled the length and surface characteristics of the muscle fibers. Mesit et. al [111], [112] modeled an interface for parametric soft-body simulation that employed pressure based constraints. This work used implicit integration (rather than explicit) to resolve the stiffness associated with the underlying system of equations. Finally, Müller introduced the Position-Based Dynamics method [60] [113] which extensively models complex soft-bodies and their interactions. As discussed in Section 2.5, this method is numerically less accurate as compared to other methods.

While most of the prior art dealt with the two factors (simulated dynamics and visual conformity) discussed in the introduction of this chapter, there is limited work towards the integration of the aforementioned advancements into a generalizeable framework for rapid development.

5.5 Methods

This ADF has been extended to incorporate an evolutionary interface for the specification, simulation, and manipulation of soft-bodies with full backward compatibility. The inclusion of the soft-body support to existing specification interfaces of ADF is shown in Figure 5.1 and it can be seen that it uses the basic principles (Section 4.3) used to describe rigid body dynamics.

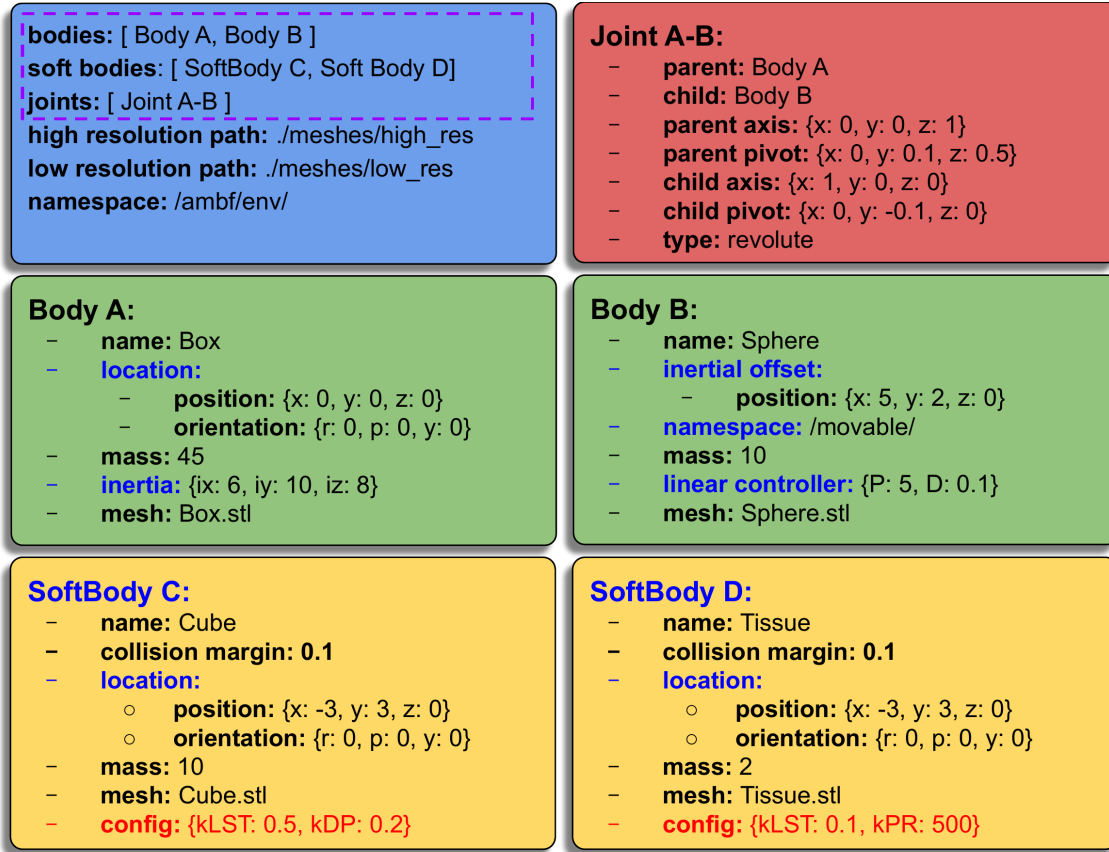


Figure 5.1: Anatomy of the ADF. The blue tile forms the header and consists of global parameters and header lists which are highlighted with the purple dotted border. The red tile represents a constraint, green represents rigid bodies and yellow represents soft-bodies. The tunable parameters for soft-body dynamics can be set using the config parameter highlighted in red. The defined parameters include $kLST$ = Linear Stiffness Coefficient, kDP = Node Damping Coefficient, kPR = Internal Pressure Coefficient.

5.5.1 Real-Time Simulation of Soft-Body Dynamics

Soft-bodies can be represented using a collection of nodes with inertial properties interconnected to their nearest neighbors. The interconnections between two nodes can be generalized with a constraint formed by a three-dimensional spring which models tension, torsion, and flexion. Additionally, each node is subject to the laws of dynamics and can collide with other objects in the environment. The combination of constraints due to motion, collision, and contact dynamics can be modeled using different methods which can be categorized into direct force computa-

tion, velocity-based methods (Sequential Impulse Constraints [114]), position-based methods (PBD) [60], and indirect representation as Linear Complementary Problems (LCPs) [115]. These methods have been discussed along with their differences in Chapter 2. The position of each node is updated at every step of the dynamics simulation based on the symplectic Euler method (although the implicit method is also used in some instances). Since the underlying problem of updating the position of each node is implemented numerically, the time-step dt between each update is an important factor in determining the accuracy of the solution. In real-time dynamic simulations where collision computation is a factor of the number of bodies in contact, it is impractical to fix the time-step dt to a preset value.

Mixed soft-body and rigid-body simulations with real-time dynamic updates pose challenges to implementation as stability and convergence is not guaranteed with a compound limit on the length of the time-step along with the number of sub-iterations. This problem is similar to the implicit variation of time step dt as mentioned in Section 3.4.1. The equation is presented again below:

$$dt < \delta t_i \times N \quad ; \quad N \in \mathbb{Z}^+ \quad \& \quad N \leq N_{max} \quad (5.1)$$

A similar methodology is applied for soft-body simulations while altering N_{max} from 10 to a lower value depending upon the complexity of the soft-body simulation at hand.

5.5.2 Representation of a Soft-Body

As discussed in Section 5.5.1, soft-bodies can be represented by inertial nodes that are interconnected to nearest neighbor nodes. Meshes used in computer simulations also employ a similar form of interconnection of vertices, although meshes usually

define only the surface. However, meshes for soft-body representation (defined by nodes) also comprise of an internal lattice forming the skeletal structure of the soft-body. An example is shown in Figure 5.2 showing two similar bodies with equivalent sub-divisions along the surface, however, one body (purple) has an internal skeletal structure while the other (pink) does not.

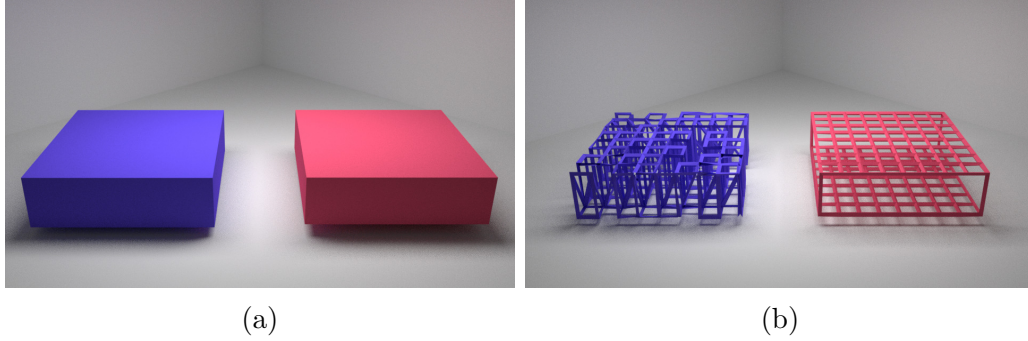


Figure 5.2: Two meshes with similar surface geometry but different internal structure defined using the OBJ mesh format.

Rather than developing a new specification for representing meshes with surface or internal 3D structure, existing standards were utilized. The following mesh formats have been used to define soft-bodies and are discussed in terms of their advantages and disadvantages for soft-body representation.

- **The StereoLithography (STL) Mesh:** STL is an older and widely used format. It supports the definition of triangle faces by specifying three vertices as floating-point numbers. A normal is specified for each triangle as well. Since each triangle defines its vertices explicitly, the shared vertices are repeated for each triangle.
- **The Autodesk (3DS) Mesh:** 3DS is a proprietary binary format that is widely used. Similar to STL, 3DS defines the mesh geometry as triangles containing a maximum of three vertices. 3DS can also define scene objects for animation software.

- **The Wavefront (OBJ) Mesh:** The OBJ format has more features compared to STL and 3DS representations and is a non-proprietary format. The biggest advantage for soft-body representation is that each face can comprise of more than 3 vertices. Similarly, edges without faces can be defined as polylines. All the vertices are specified as separate lists, and therefore, vertex repetition can be avoided although is not mandated in the format. OBJ can additionally store the normal and texture data per-vertex.

For representation, all three of these meshes have been supported. However, the Wavefront’s OBJ is the preferred mesh format since one can utilize the polylines feature to define non-faceted vertex interconnection. Additional mesh formats such as the VTK format² may be supported in the future. For visual realism, a pair consisting of a visual (high-density) mesh and a low-resolution collision mesh is used. The visual mesh can also store the texture information which is not used by the collision mesh. Additional properties for soft-body dynamics are specified in the same data-block as shown in Figure 5.1.

5.5.3 Visualization

As discussed in Section 5.5.2, a pair of visual and collision mesh is used to represent the soft-body. The collision mesh can be defined by using any of the three supported mesh formats. The array of vertices retrieved using these formats includes redundant vertices which pose a problem for the creation of the underlying soft-body nodes. This requires the elimination of the repeated vertices by unification. The brute force approach to counting repeated vertices is computationally exponential, and thus, not desirable. Instead, hashing techniques are used that turn the vertex unification into an almost linear problem. These techniques are usually modifications of the “vertex

²https://www.paraview.org/Wiki/ParaView/Data_formats

Algorithm 7 Vertex Triplet Generation

```
1:  $s_{block} = \text{Param. Block Size}$ 
2:  $n_{blocks} = n_{vtx} / s_{block}$   $\triangleright n_{vtx} = \text{No. of Vertices}$ 
3:  $res_{[x,y,z]} = n_{vtx} / bounds_{[x,y,z]}$ 
4:  $vtxChk[n_{vtx}] \leftarrow False$   $\triangleright vtxChk = \text{Mark Chkd Vtx}$ 
5:  $vtxTrp[n_{vtx}][3] \leftarrow -1$   $\triangleright vtxTrp = \text{Vtx Triplet}$ 
6:  $CHK[s_{block}][s_{block}][s_{block}] \leftarrow False$ 
7:  $IDX[s_{block}][s_{block}][s_{block}] \leftarrow -1$ 
8: for  $x_{block} = 0$  to  $n_{blocks}$  do
9:   for  $y_{block} = 0$  to  $n_{blocks}$  do
10:    for  $z_{block} = 0$  to  $n_{blocks}$  do
11:       $[x, y, z]_{low} = [x, y, z]_{block} + s_{block}$ 
12:       $[x, y, z]_{high} = [x, y, z]_{low} + s_{block}$ 
13:       $CHK[s_{block}][s_{block}][s_{block}] \leftarrow False$ 
14:       $IDX[s_{block}][s_{block}][s_{block}] \leftarrow -1$ 
15:      for  $i = 0$  to  $n_{vtx}$  do
16:        if  $vtxChk[i] == False$  then
17:           $p \leftarrow vtx.position$ 
18:           $key_{[x,y,z]} = res_{[x,y,z]} * (p - min_{[x,y,z]})$ 
19:          if  $key_{[x,y,z]} \in [[x, y, z]_{low}, [x, y, z]_{high}]$  then
20:             $key_{[x,y,z]} = key_{[x,y,z]} - bounds_{[x,y,z]}$ 
21:             $vtxChk[i] = True$ 
22:             $vtxTrp[i][0] = i$ 
23:            if  $CHK[key_x][key_y][key_z] == False$  then
24:               $CHK[key_x][key_y][key_z] = True$ 
25:               $IDX[key_x][key_y][key_z] = i$ 
26:               $vtxTrp[i][1] = i$ 
27:            else
28:               $vtxTrp[i][1] = IDX[key_x][key_y][key_z]$ 
29:            end if
30:          end if
31:        end if
32:      end for
33:    end for
34:  end for
35: end for
```

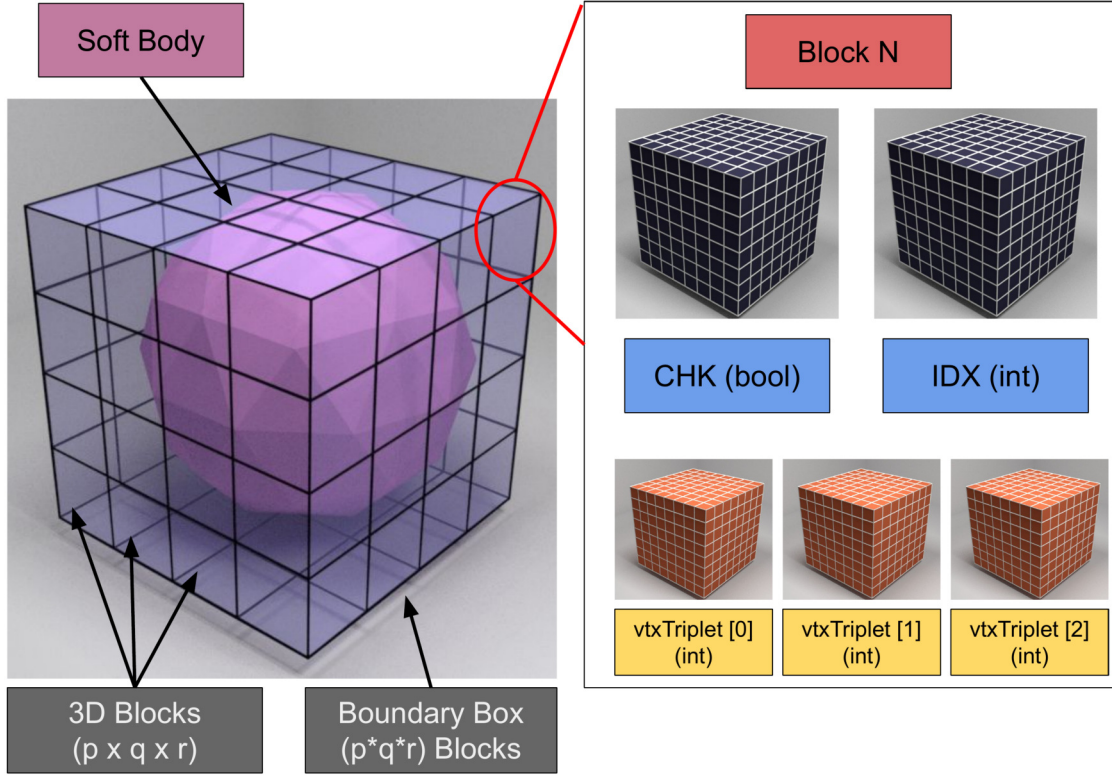


Figure 5.3: Reference image for Algorithm 7. The soft-body fits in the boundary box that is sub-divided into p , q and r blocks along x , y and z axes respectively. Each block is then parsed individually by creating 5 sub-blocks which are a CHK, an IDX and 3 Vertex Triplet sub-blocks.

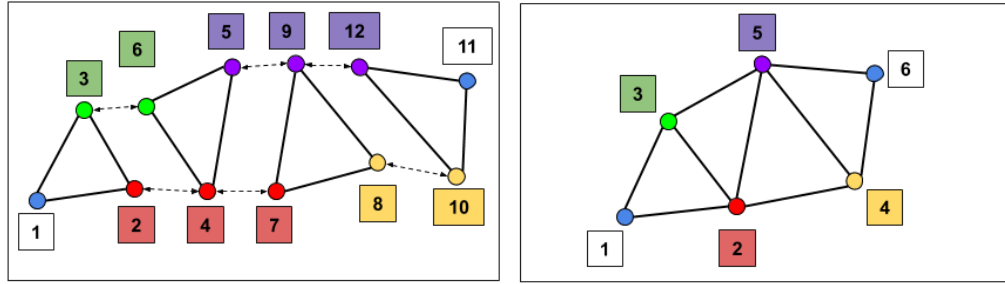


Figure 5.4: (a) The original vertex indices that do not account for repeated vertices. (b) The reduced vertex list with the duplicate vertices unified together into a new list.

welding” [116] approach in which the vertices forming a mesh are discretized into smaller bins (sub-blocks). The resulting welded mesh has a reduced vertex count. An extra step is required to modify the triangle indices forming the faces of the mesh to map (“rewire”) to the reduced set of vertices. This extra step is not directly related

to the vertex welding algorithm.

Vertex welding essentially gets rid of the repeated vertices which is necessary for a proper definition of an underlying soft-body but such a reduced mesh might not be desirable for a generic rendering application (which uses the original mesh with repeated vertices for visualization). Therefore, a different approach, which is in part based on vertex welding, is proposed which unifies the repeated vertices and stores them in a data structure. This algorithm also stores the relation between unified vertices and their original non-unified copies. The resulting data-structure is then used for rendering and solving the soft-body after each dynamic update-step.

The algorithm discussed above can be divided into two separate parts which include (1) the Vertex Triplet Generation (Algorithm 7) and (2) Generating Unique Triangle Indices (Algorithm 8). The first algorithm fills a data structure consisting of three arrays (3-dimensional sub-blocks). This data structure is called the **Vertex Triplets** and its three associated arrays are described as follows:

- **Original Vertex Indices** $vtxTriplet[0]$ The first array contains the indices to original vertices that form the mesh.
- **Unified Vertex Indices** $vtxTriplet[1]$ The second array contains the vertex indices referring to the first index at which the vertex occurred in the original vertex list.
- **New Vertex Indices** $vtxTriplet[2]$ Finally the third array contains indices from a newly formed array containing only the distinct vertices.

To complement Algorithm 7, Figure 5.3 visually illustrates the associated data structures mentioned in the algorithm. The **Vertex Triplets** for the mesh triangles shown in Figure 5.4 are presented in Table. 5.1. Afterward, Algorithm 8 is used

Table 5.1: Population of Vertex Triplets for the example in Figure 5.4

vtxTrp[0]	1	2	3	4	5	6	7	8	9	10	11	12
vtxTrp[1]	1	2	3	2	5	3	2	8	5	8	11	5
vtxTrp[2]	1	2	3	2	4	3	2	5	4	5	6	4

to compute rewired triangle indices corresponding to the newly sorted vertices and edges.

Algorithm 8 Generating Unique Triangle Indices

```

1:  $vtx_{count} = 0$ 
2:  $vtx_{tree} = [n_{vtx}][]$ 
3: for  $i = 0$  to  $n_{vtx}$  do
4:   if  $vtxTrp[i][1] == vtxTrp[i][0] \& vtxTrp[i][2] == -1$  then
5:      $vtxTrp[i][2] = vtx_{count}$ 
6:      $vtx_{tree}[i] \leftarrow vtx$ 
7:      $vtx_{count}++$ 
8:   else if  $vtxTrp[i][1] < vtxTrp[i][0]$  then
9:      $bIdx = vtxTrp[i][1]$ 
10:     $cIdx = vtxTrp[i][2]$ 
11:     $vtxTrp[i][2] = cIdx$ 
12:     $vtx_{tree}[cIdx] \leftarrow i$ 
13:   else if  $vtxTrp[i][1] > vtxTrp[i][0]$  then
14:      $bIdx = vtxTrp[i][1]$ 
15:      $vtxTrp[i][2] = vtxTrp[i][2]$ 
16:   end if
17: end for

```

5.5.4 Manipulation of Soft-Body

Manipulation of soft-bodies involves several intermediate preemptive steps. To develop a generic implementation to grasp any soft-body, sensors based on ray-tracing elements which are placed on the simulated graspers to detect proximity to collision objects were used. The ray-tracing algorithm is used in computer simulations to trace out the path of light rays as they repeatedly collide with objects in simulation and as a result, each ray computes the sensed points of contact with objects along its

path. Therefore, ray-tracing can be used to detect the nearest points between two surfaces as well. In a trivial ray-tracing implementation, the starting point of the rays originates at the light source, which is usually fixed. However, for the nearest point calculation in dynamic objects, the rays can be parented to a specific dynamic body. Therefore each proximity sensor has the attributes shown in Figure 5.5.

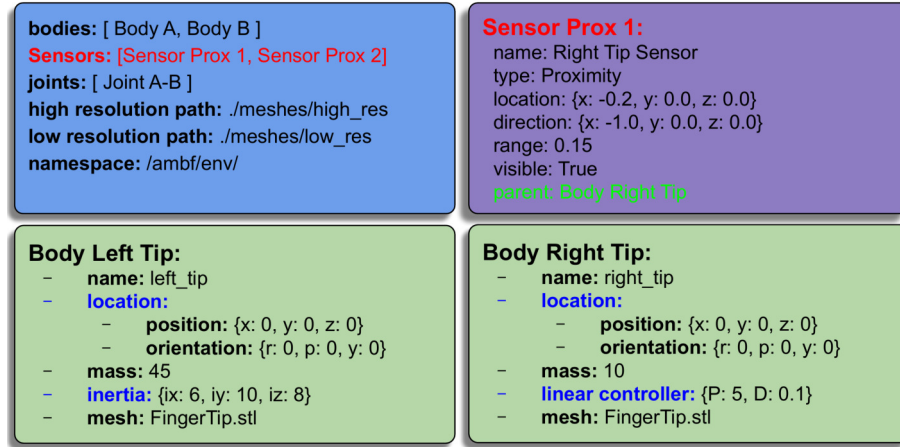


Figure 5.5: Proximity sensors can be defined using the same method as bodies, joints, and scene objects in Figure 5.1. The proximity sensor is parented to the desired body with the relative location offset, direction, and range.

Once a proximity sensor triggers and the grasping closure angle is less than a user-specifiable threshold (any angle of the grippers jaw), the contact face in the ray’s path is found. Then, the nodes forming the face are anchored using Algorithm 9. An anchor is a simulated constraint that attaches the vertices forming the nearest soft-body face to the parent body on which the sensor is mounted. To prevent jerk on newly anchored vertices, whenever grasping occurs, the offset between the parent body and the vertices is stored and then used as the desired offset while the anchor remains intact. This anchor produces a weak force according to Algorithm 10 that guides the connected vertices along the parent body.

Algorithm 9 Anchor Vertices to Parent

```
1:  $G := \text{Sensor's Parent}$ 
2:  $\text{Face} \in \text{Get Nearest Face to Contact Point}$ 
3:  $\text{Vertices} \in \text{Face}$ 
4:  $T_G^W \in G.\text{Transform}$ 
5: for  $v \in \text{Vertices}$  do
6:    $P_v^W \in v.\text{Pos}$ 
7:    $P_v^G = (T_G^W)^{-1} P_v^W$ 
8:    $a \leftarrow \text{Anchor}(v, P_v^G)$ 
9:    $G.\text{Anchors.append}(a)$ 
10: end for
```

Algorithm 10 Update Anchored Vertices

```
1: for  $G \in \text{Rigid Bodies}$  do
2:    $T_G^W \in G.\text{Transform}$ 
3:    $H_a, D_a := \text{Parametric Anchor Hardness and Damping}$ 
4:   for  $a \in G.\text{Anchors}$  do
5:      $P_v^{G'} \in a.\text{Offset}$ 
6:      $v \in a.\text{Vertex}$ 
7:      $P_v^W \in a.\text{Offset}$ 
8:      $P_v^G = (T_G^W)^{-1} P_v^W$ 
9:      $\delta P_n = P_v^{G'} - P_v^G$ 
10:     $F_a = H_a \delta P_n + D_a \frac{\delta P_n - \delta P_{n-1}}{dt}$ 
11:     $a.\text{ApplyForce}(F_a)$ 
12:   end for
13: end for
```

5.6 Results

As previously discussed in the introduction to this chapter, soft-body simulation for training simulators aimed for surgical robotics is a multi-fold problem, four of which have been identified in this chapter. The results of these four challenges are discussed in the following section.

Freely available software can be used for creating representative meshes for soft-bodies. In this chapter, Blender [98] was used simply due to the extensive prior use with AMBF [117]. To create a soft-body representation, either a shape primitive (as a mesh) or an externally supplied mesh can be used. These meshes can then be sub-divided, morphed, and cut using the existing tools provided in Blender. Figure 5.6 (a) and (b) shows the operations of converting a primitive mesh into a compound shape using simple Boolean operations. Similarly, Figure 5.7 shows the conversion of a simple cylindrical shape into a body resembling a textured slice of meat.

It is challenging to impart volumetric constraints to soft-body meshes. For this reason, many different approaches are used based on the context of simulation. These approaches include either constraint on the volume of a convex mesh [118], or a constraint of the form of a pressure [119]. Another popular approach is to model a skeletal mesh that forms the structure of the original visual mesh. Tetgen [120] is a useful library, which among other features, can be used to skeletalize a visual mesh. However, for this discussion, the internal structure of the mesh is explicitly computed by connecting the desired vertices using edges. It is important to note that connecting vertices inside the surface only requires the creation of edges and not faces. As discussed in Section 5.5.2, only the OBJ format can be used to store non-faceted connections. Figure 5.2 shows the original visual mesh with an interconnected lattice defined using generic tools in Blender.

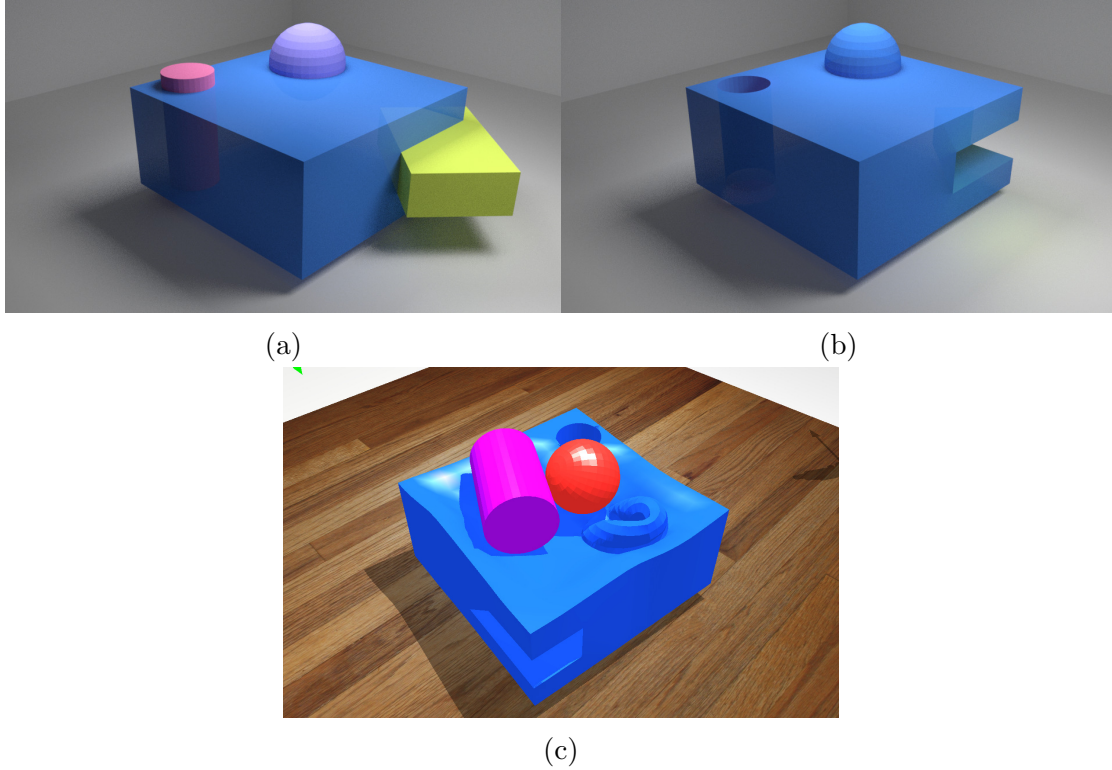


Figure 5.6: Similar to convex hulls used for rigid-body dynamics, a complex soft-body shape can be generated using a compound of simpler shapes. These simpler shapes can be used to perform Boolean operations of mesh subtraction or addition as shown from (a) \rightarrow (b). Finally, (c) shows the simulation and interaction of this mesh in AMBF.

The specification of soft-bodies using the front-end format described in Sections 5.5.1, 5.5.2 is parametric. Soft-body dynamic properties that define flexion, torsion, elongation etc. can be set in the config field displayed in red in Figure 5.1. This is the final step, after which these soft-bodies can be spawned alongside rigid-bodies and robots and can be manipulated with multiple IIDs including dVRK MTMs.

Figure 5.8 demonstrates the interaction of soft-bodies with SDEs that are controlled via dVRK MTMs. The methodology used for grasping is not modeled after natural interaction as it does not account for friction, and therefore, it does not allow controlled manipulation based on stick-slip friction around the grab points. However, grabbing soft-bodies using the methodology presented in Section 5.5.4 provides a simplistic, yet useful, approach to interactive manipulation. It should be noted

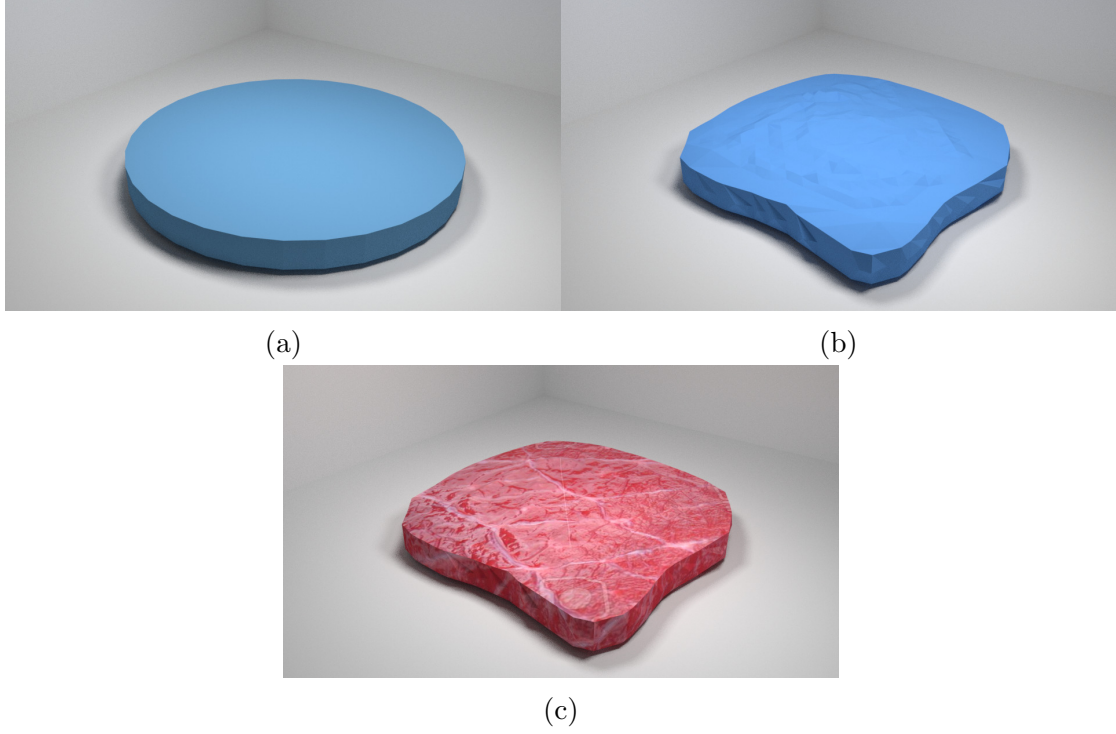


Figure 5.7: Sequential process of converting a cylindrical primitive to a mesh with coarse surface, creating edges inside for structural stability, and finally applying texture for visual realism.

that this methodology only works on faces of soft-bodies as covered in Algorithm 9. The grab force can be varied by changing the value of H_a & D_a in Algorithm 10. Figure 5.9 illustrates the Real-Time Factor (RTF) during the example of soft-cloth manipulation shown in Figure 5.8. The RTF is calculated by dividing the simulation clock time by the real world clock time and a value of 1 indicates that the simulation is real-time.

5.7 Discussion

While this extension to the Asynchronous Framework can potentially provide a convenient research platform for the research community working towards surgical robotics, several additional features need to be added. First, the soft-body implementation is limited to homogeneous materials, while surgical tissues are both

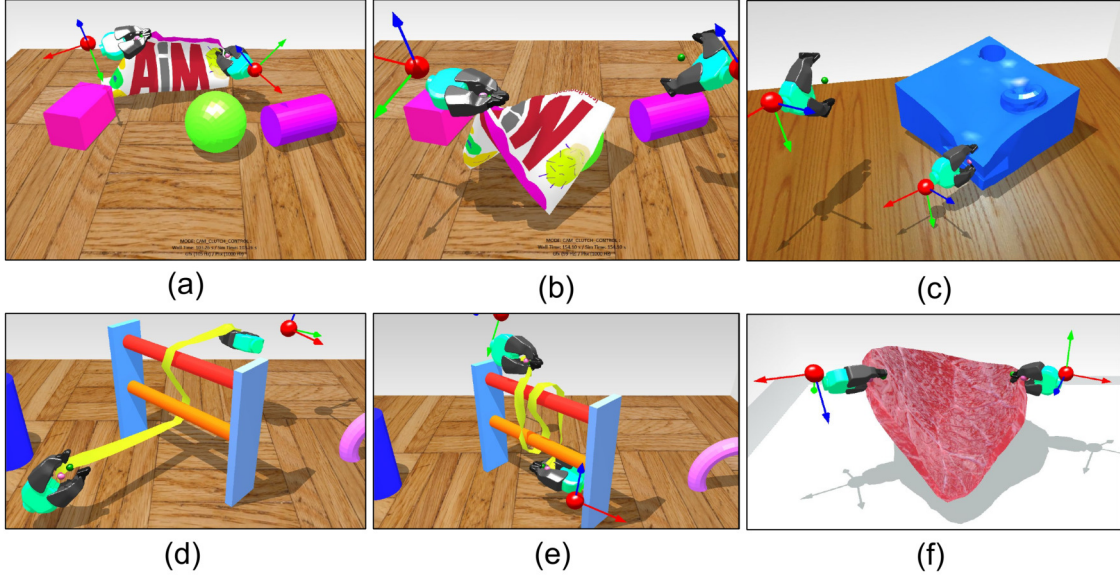


Figure 5.8: Examples of Soft-body manipulation using the dVRK MTMs

non-homogeneous and visco-elastic. This might be possible, in the future, with the inclusion of additional data per vertex, in the mesh format. Secondly, the soft-bodies are usually anchored to fixed points. This is currently implemented by using the index of required nodes and setting their mass to be zero. In physics simulations, the mass of 0 has a special meaning as it essentially treats the object as immovable (or having infinite mass). A better approach is to use either visual guides or rough positional coordinates for setting the node masses to zero.

Other features that are necessary for a simulator targeting soft-body simulators for surgical robotics is the support for cutting and stitching. In this regard, an extension of AMBF's sensor interface is required. Similar to proximity sensors (Figure 5.5), cutting sensors can potentially be mounted to a rigid-body and can be used to sub-divide and dissect the connecting links nearest to the contact point. Stitching is a more challenging problem to address using generic methods and requires further exploration.

Lastly, there is a need for developing soft-body environments that can replicate

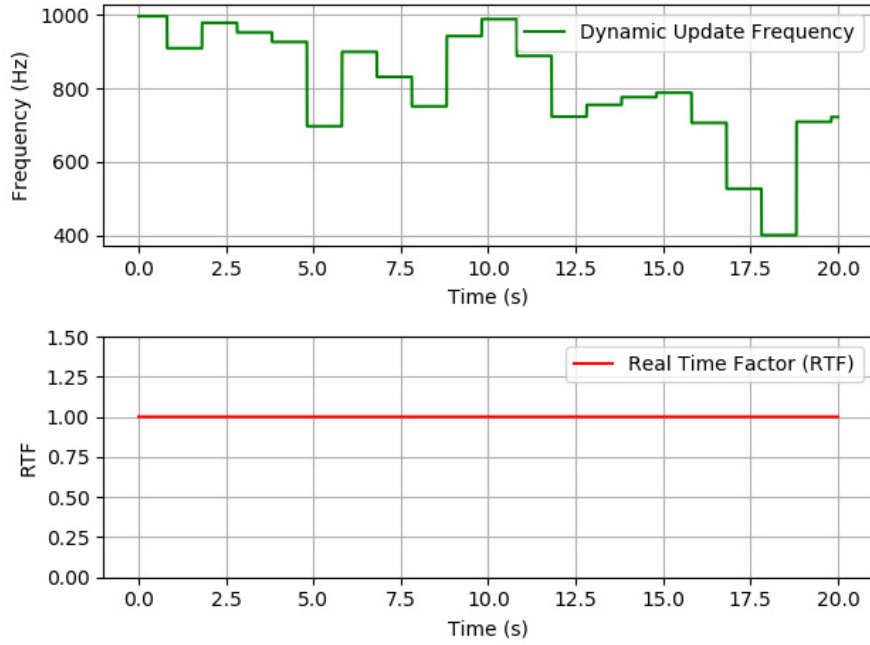


Figure 5.9: Real Time Factor for tasks shown in Figure 5.8 (a), (b).

actual surgical sub-tasks while using our proposed framework. Although this task mostly entails the specialty required to design such environments rather than the validity of the proposed framework, it is imperative in demonstrating the actual use-case of the AMBF soft-body extension.

Chapter 6

Grasping in Simulation

Having discussed the various aspects of the Asynchronous Framework, all the way from soft-body simulations to the inclusion of IIDs for multi-user collaboration, this chapter discusses a fundamental aspect of user interaction to manipulate objects in a simulated environment. In most cases, interactive manipulation by a user-controlled SDE is initiated via grasping, which is the focus of this chapter. The challenges to grasping using only the friction modeled in physics simulations is presented in Section 6.4. These challenges are overcome by the inclusion of penalty-based contact penetration elements called Resistive sensors (Section 6.5.1). These sensors can be contoured to the desired shape and parametrized using the ADF specification. This is discussed in Section 6.5.4. Finally, the results and discussions are presented in Section 6.6 and 6.7.

6.1 Acknowledgement

Most of the work presented in this chapter has been submitted for review as:

Munawar A, Srishankar N, Fichera L, Fischer G, “*Multi-Manual Grasping and Interaction in Real-Time Dynamic Simulations using a Penalty Based Approach*”,

6.2 Introduction

Interactive computer simulations play an important role in robotic teleoperation by enabling human operators to practice and gain experience with a system before operating the actual physical hardware. In the context of the Asynchronous Framework, users can interact with the dynamic objects (such as simulated soft-tissues, rigid-body puzzles, peg and hole tasks, etc.) using the SDEs controlled (teleoperated) via the IIDs. These interactive simulators can enhance the skills of the user for performing coordinated hand motions for complex tasks.

The ability to grasp and manipulate physical dynamic objects is fundamental to such interaction which includes the dynamic deformation of skin tissue in the vicinity of contact points as shown in Figure 6.1. To be perceived realistic, interactive simulators need to compute the real-time physics of the interaction between the robot and its environment. This encompasses, among other things, the interplay between the forces created by the user through his/her actions, the forces created by the objects present in the simulated environment, and the resulting changes in the environment.

While most interactive simulators excel at rendering realistic physics, grasping is implemented using simplified techniques that do not mimic natural contact dynamics as shown in Figure 6.1. As an example, one widely used technique for simulated grasping [24], [121] deactivates the dynamic properties of the grasped object and treats it as a kinematic body affixed to the grasper. This approach has obvious shortcomings as it does not scale well with multi-manual manipulation. Moreover, an important aspect of natural grasping is the ability to allow controlled “slip and

slide” of the grasped object as shown in Figure 6.2. This is useful in training applications, modeled after realistic tasks, that require multi-handed interactions but is challenging to implement.

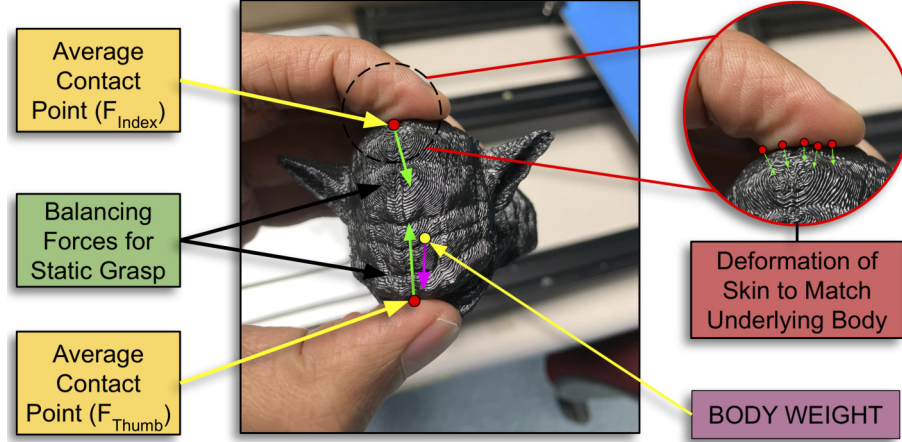


Figure 6.1: Natural method of grasping for fully-static or quasi-static dynamics. The skin surface in the vicinity of contact points deforms according to the underlying shape of object, thereby providing better surface friction.

Based on the associated challenges to modeling contact dynamics, two different classes of simulators are currently used in research. One class specializes in high-speed rigid-body dynamics, employed mostly for training and entertainment purposes while the other class of simulators use time-consuming Finite Element (FEA) based methods for offline computation of surface deformation and contact response, which is not suitable for real-time dynamics. Although this study does not intend to replicate the techniques from the latter class of simulators into the former, it is a step towards separating interaction/contact dynamics for grasping purposes in the context of interactive simulators into a formalized problem. This study utilizes a form of penalty based parametric sensors [122], [123] that can be mounted on any grasper to emulate adequate friction for easy grasping. The proposed approach is implemented on a variety of different simulated graspers and friction surfaces. These graspers are used to perform complex tasks that involve manipulation of various dy-

namic objects, physical interaction with surgical robots, and controlling deformable bodies (modeled using finite primitives) using two hands. These approaches are generic and can be modeled on any existing physics library.

In this chapter, the grasping approach is demonstrated on both rigid bodies and deformable bodies. It is important to establish the difference between the use of the term “deformable body” from the term “soft-body” within the context of this chapter. Soft-bodies, as described in Chapter 5, are represented by a single mesh, comprising of vertices connected to form faces. The vertices can interact with other objects in the environment and result in the deformation of the corresponding faces. Deformable bodies, on the other hand, are represented by a finite group of rigid-body meshes (nodes), connected via constraints. Unlike soft-bodies, the deformable bodies have no faces between the inter-connected nodes.

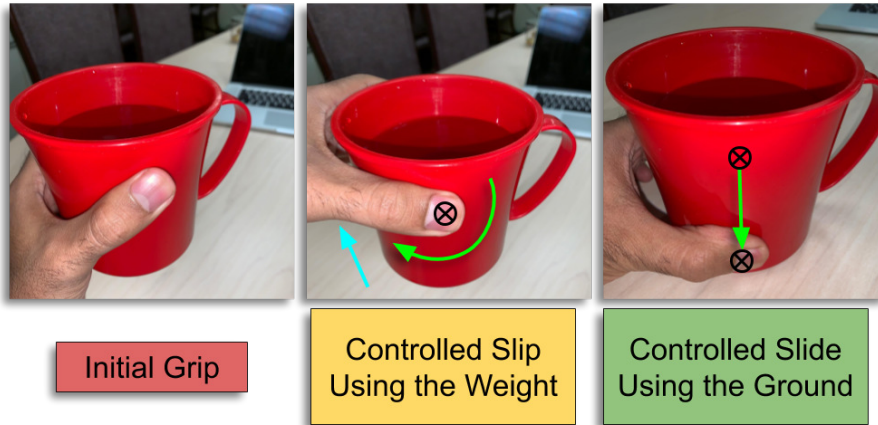


Figure 6.2: Natural Manipulation using either controlled slip, controlled slide or both. The controlled slip and slide is usually assisted by either the weight of the grasped body, using a second hand or leveraging the collision with other objects in the environment.

6.3 Related Work

A summary of notable work addressing the problem of grasping using contact dynamics in simulation is described in this section. GraspIt! [124] by Miller et al.

was a prominent simulator that contained multiple robotic hands, contact body dynamics, and a basic grasp planner, but lacked an API and a modular architecture which limited its functionality. A proposed improvement by Léon et al., called OpenGRASP [125] simulates contact sensors for grasping and attempts to create a realistic simulation of grasping rigid objects using soft contacts.

Moisio et al. [126] used the OpenGRASP toolkit to improve an existing model of simulated tactile sensors that used a soft contact approach without modeling stick-slip. The improvements were made by using a parametric contact force model for surface forces, holding torques and stick-slip as well as generating the sensors using a geometry patch. The resulting simulated framework was compared with real-world robot grasping. However, the authors stated that the computational efficiency of the tactile sensor model can be improved by using a non-brute force collision detection solver. Ciocarlie et al. [127] worked on a general analytical method to model fingertip gripping with friction by analyzing friction constraints on non-planar contacts of elastic materials, formulating a linear complementary problem (LCP), and removing any assumptions about the objects geometries. Goldfeder et al. [128] implemented a grasp planner in GraspIt! that was generalizeable to various object/hand geometries/kinematics by decomposing and representing an object as a tree of super-quadratics, hence defining a smaller search space of potentially successful grasps as those which have good grasps on sub-components of the object in the decomposition tree.

Hawkes et al. [129] proposed an alternative to gripping an object purely using the normal force. Their work used gecko-inspired controllable fibrillar adhesives that utilize tangential shear forces mimicking the curvature of an object. Such a gripper could grasp convex objects that are relatively large and featureless as well as delicately grab objects without squeezing. Todorov et. al. model the contact

dynamics for grasping as an implicit complimentary problem instead of a linear complimentary problem [130]. The implicit complementarity formulation sets contact impulses and contact velocities as functions that satisfy the complementarity constraints automatically and optimizes for a set of unconstrained variables which in turn reduces the number of unknowns by a factor of three. They also created a simulation which incorporated multi-joint dynamics with grasping [131].

Malvezzi et al. [132] developed a lightweight Matlab toolbox called SynGrasp. In addition to providing an easy way to load hand models, it allows grasping performance analysis and grasps quality measures such as minimizing contact forces for a given grasp based on a specified cost function. Finally, Spiers et al. [133] worked on building a mechanical gripper equivalent of a biological finger by having a low-friction surface for an object sliding as well as a high-friction surface for firm gripping.

Existing applications for contact-based grasping are usually restricted to rigid bodies that are simplified as collision primitives or convex hulls. Interactive training, on the other hand, may involve multiple non-convex and multi-jointed puzzles in a real-time simulation that requires multi-manual and within-hand manipulation. Moreover, existing applications model the graspers to have uniform surface friction properties. As discussed in [133], however, mechanical equivalents of a biological finger can have surface regions with different friction coefficients. Therefore a generalizable approach for interactive training is presented which uses penalty based contact sensors that complement the underlying collision constraints provided by the physics solver [34].

6.4 Problem Formulation

The challenges associated with realistic grasping in simulation can be investigated in conjunction with the limitations introduced by A) calculating the collision between dynamic bodies, B) the geometrical representation of simulated rigid bodies and C) the implementation of dynamics in physics libraries. These are discussed as follows:

6.4.1 Limitations Associated with Rigid-Body Collisions

Collision techniques for simulated bodies differ greatly from real-world collisions which impacts the computational methodology of contact forces. The state-of-the-art algorithms used for rigid-body collision can be classified into implicit and explicit techniques [56]. Implicit collision techniques are employed for primitive shapes while non-primitive shapes (convex and concave meshes consisting of faces and vertices) are solved for using explicit collision techniques. Implicit techniques model the collision shapes using analytical expressions that represent the shape of the underlying geometric primitives. As an example, consider a simulation involving spherical bodies which are modeled using the corresponding analytical function, representing the geometry. The collision computation only requires that the center $P_{B_i}^w$ of each **Body B** maintain its radius r_{B_i} from all the other bodies. This constraint can be expressed simply as $\|P_{B_1}^w - P_{B_2}^w\| \geq (r_{B_1} + r_{B_2})$. On the other hand, explicit collision shapes require instantaneous (at each update step) calculation for all the colliding faces to generate a resulting force acting on the body. The Gilbert-Johnson-Keerthi (GJK) [134] algorithm is used quite often in modern physics and collision detection libraries for computing collision between explicit (non-primitive) shapes. Due to the nature of physics computation using numerical methods (implicit and explicit Euler methods), the dynamic bodies are processed at discrete time-steps $[n, n_1, n_2, \dots, n_d]$.

As a result, simulated rigid bodies (for instance) might penetrate each other such that the constraint $\|P_{B_1}^w - P_{B_2}^w\| \geq (r_{B_1} + r_{B_2})$ is violated. The resulting error $D_p = r_{B_1} + r_{B_2} - \|P_{B_1}^w - P_{B_2}^w\|$, called the penetration depth, is used to calculate a resulting correction force which serves to “repel” the penetrating bodies to re-satisfy the constraint.

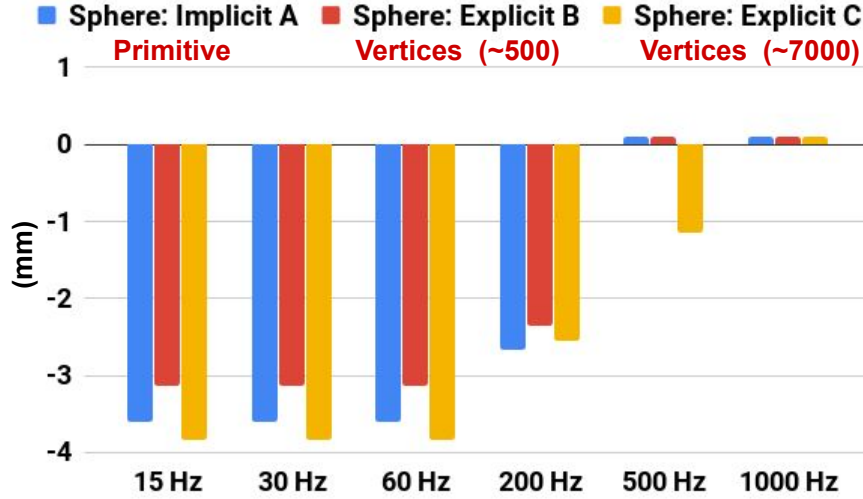


Figure 6.3: The penetration depth D_p of three collision shapes (Spheres) with mass = 50 Kg, radius 0.5 m, non static ground plane position at 0 m and drop height = 2 m. The penetration depth was recorded as the difference between the maximum fall distance and the resting position after stabilization.

6.4.2 Limitations Associated with Geometric Representation of Rigid Bodies

Another important difference between simulated and physical rigid bodies is that simulated rigid bodies are simplified to have infinite surface stiffness. As a result, the surface in the vicinity of a contact point for a simulated rigid-body does not deform. Moreover, individual faces of shapes and meshes, representing simulated bodies, are locally smooth whereas the contact surfaces of physical bodies are rough

at a microscopic level as illustrated in Figure 6.1. This roughness plays a major role in both the static and sliding friction response.

An interesting observation from the collision algorithms for rigid-body dynamics is that they render an inherent softness in the form of the penetration depth D_p that counteracts the infinite stiffness of rigid bodies. At a glance, this penetration depth can be leveraged to mimic the softness inherent to real-world bodies. However, this penetration depth varies for the simulated body based on a few factors, which include 1) the update rate of the physics simulation and 2) the complexity of the body’s geometry. As demonstrated in Figure 6.3, an implicit and two explicit spherical shapes (of a different number of mesh faces) of identical scale are dropped onto a static plane at various physics update frequencies. The difference in geometry and physics update frequency alters the penetration depth of each object. This varying behavior of the resulting penetration depth D_p limits the use of implicit softness rendered by collision algorithms for contact dynamics purposes.

6.4.3 Dynamics Calculation in Physics Libraries

The computation of the normal force \vec{F}_N is used for the derivation of both the static and sliding friction. The analytical approach to modeling the static friction force, depicted in Figure 6.4 is trivial for specifically optimized examples, modeled using non-stiff differential equations. On the other hand, for general rigid body dynamics, as in our case, iterative techniques are preferred where the collisions are modeled as instantaneous inequality constraints. Examples of these methods include both velocity (such as Sequential Impulse [135] [61]) and position based methods (PBD) [60]). These methods counter the penetration by applying corrective impulses (proportional to penetration) or position correction respectively. The application of corrective impulse instead of direct position rectification makes velocity based methods

more suitable for simulating friction. However, even for SI solvers, the dependence of the normal force on the penetration depth makes the friction constraint non-linear and not strictly convergent. Moreover, it is difficult to compute the correct contact area for non-primitive shapes without some simplifications. Thus especially for real-time simulations, where the accuracy of the solver has to be comprised to some extent, the rigidity associated with collision shapes makes even an acceptable friction model using velocity based method insufficient for rendering adequate stick-slip friction.

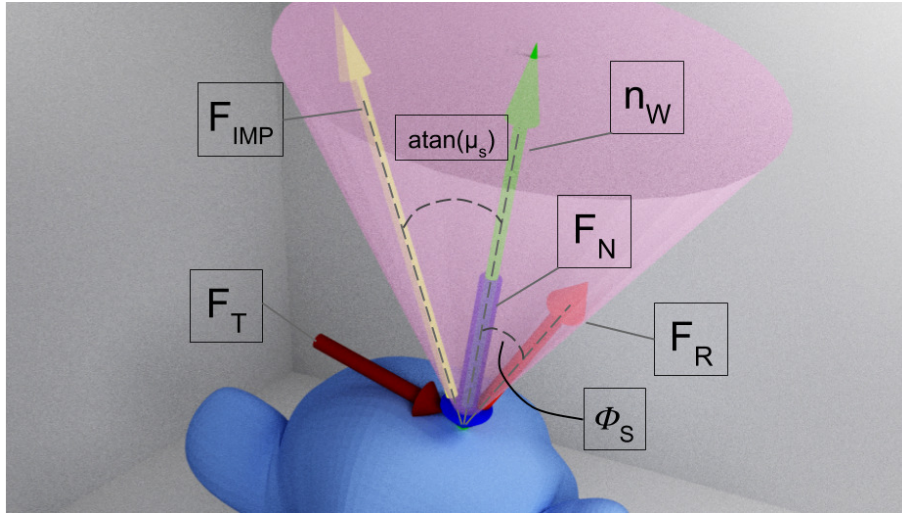


Figure 6.4: Visualization of the friction cone to model the natural friction response. The coefficient \vec{n}_W is the contact normal in the world, \vec{F}_R is the resultant force which is expressed as ($\vec{F}_R = \mu * \vec{F}_N$), \vec{F}_{IMP} is the impending friction and ϕ_s is the static friction ratio.

Penalty based contact modeling approaches [122], which are forms of Force Based methods, are usually avoided in popular physics solvers due to their instability. These methods are also difficult for achieving impenetrability as a high enough velocity of a colliding body may result in passing through, referred to as “tunneling” [136]. However, in our case, it is their exact penetrability that is leveraged to simplify grasping and emulate stick-slip friction as discussed in the next section.

6.5 Methods

The following section provides details about the approach used to model sensor-based friction for grasping and manipulation. The section uses several coefficients which are listed in Table 6.1 for clarity.

Table 6.1: Symbols used in this Manuscript.

Symbol	Description
\vec{F}_S	Static Friction Force
\vec{F}_V	Sliding Friction Force
\vec{F}_N	Normal Contact Force
\vec{e}_T	Tangential Contact Error
\vec{e}_V	Tangential Velocity Error
\vec{e}_N	Normalized Penetration Depth
μ_S	Static Friction Coefficient
μ_V	Sliding Friction Coefficient
K_N	Normal Contact Stiffness
K_D	Normal Contact Damping
$V_{[A,B]}^C$	General Notation to Express V_A^C and V_B^C , which can be read as V_A and V_B expressed in C
σ_a	Contact Area for Static Friction specified as a radius

6.5.1 Resistive Sensors for Preemptive Contact Computation

Ray Shooting (Ray Tracing) [137], [138] is an approach that is widely used in computer graphics for rendering realistic scenes resulting from tracing the path of light rays that repeatedly collide with the objects in the scene. Since this technique essentially computes the intersecting point of an object along the path, it can be used for a variety of other applications (For example Section 5.5.4). For abstraction, an assembly of a limited horizon individual Ray and the associated parametric data can be called a Resistive sensor. Figure 6.5 visually demonstrates the impor-

tant parametric data associated with the Resistive sensor. Each Resistive sensor is self-contained for computation purposes which allow for managing groups of sensors in parallel. This is important as Ray-Tracing is a computationally demanding algorithm especially for a large number of Rays.

6.5.2 Anatomy of a Resistive Sensor

A Resistive sensor is shown in Figure 6.5 which visually illustrates the coefficients mentioned in Table 6.1. The sensor can be defined w.r.t. a start $\vec{P}_{rayStart}$ and an endpoint \vec{P}_{rayEnd} and is “triggered” when it intersects with another object. In addition to the starting point, the sensor is parameterized by a range and an offset (or a depth) from the surface of its parent body. To specify multiple sensors, a separate triangular mesh can be used as discussed in Algorithm 12.

Due to the challenges associated with implementing the static friction cone, as discussed in section 6.4, one alternative method is to use a fully deterministic analytical approach. This approach is shown in Figure 6.6. A torque $\tau_{[GA,GB]}$ is being applied to both fingers by an external controller. To solve for the contact normal force at points P_{AC} and P_{BC} , the knowledge of the torque $\tau_{[GA,GB]}$ at J_G , and the lengths \vec{P}_{GA} and \vec{P}_{GB} is required. This example can easily be expanded to non-quasi-static problems, where the gripper is accelerating while holding **Body Z** (Figure 6.6). In such cases, the Inertial and Coriolis components need to be considered in addition to the contact dynamics and gravitational components. Furthermore, additional bodies may interact with the gripper’s finger and as a result, increase the complexity of contact force computation at P_{AC} and P_{BC} .

Secondly, the vector of Tangential force \vec{F}_T is required to compute the Resultant force \vec{F}_R . This tangential force is balanced by the static friction force \vec{F}_S under the following condition $\vec{F}_T \leq \vec{F}_S$ and until this condition holds, there is no dis-

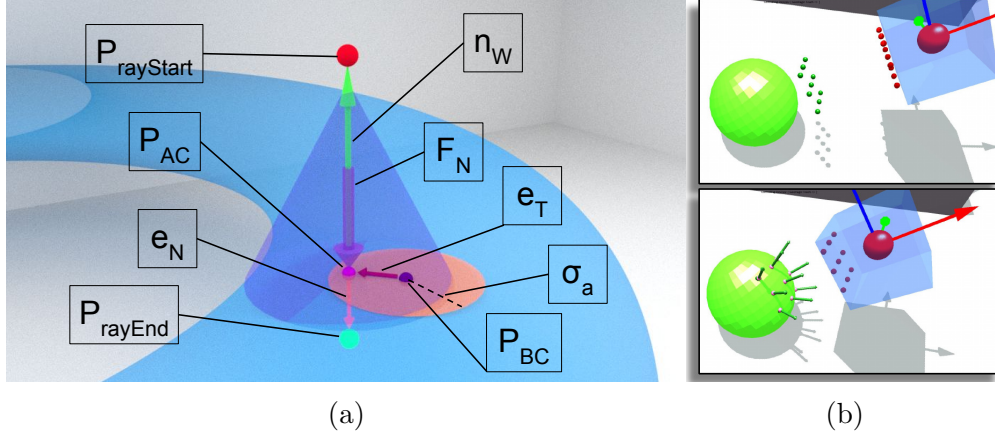


Figure 6.5: (a) A visual representation of an individual Resistive Sensor during contact with an external body B (the blue torus). The coefficients are defined in Table. 6.1. (b) Visualization of Resistive sensors mounted on a body (blue cube) before and after penetration into another body (green sphere).

placement along the tangential direction. However, due to the nature of physics computation using explicit Euler methods, it is not trivial to compute the tangential force before a displacement has already taken place. Taking this limitation into consideration, a different methodology is presented to compute the interaction dynamics which utilizes σ_a and K_N . A displacement along the tangential direction results in error $\vec{e}_T = (T_A^W \vec{Q}_C^A - T_B^W \vec{Q}_C^B)_{proj_T}$, where $proj_T$ is the projection of the contact error in the tangential plane to the direction of the sensor and is then used to compute \vec{F}_S as:

$$\vec{F}_S = \mu_S * \vec{e}_T * \|\vec{F}_N\| \quad (6.1)$$

Equation 6.1 does not rely on the limit $\vec{F}_T \leq \vec{F}_S$ but instead on the limiting error \vec{e}_T . Once a Resistive sensor triggers as a result of contact with another body, the sensed point P_C^W is recorded. This sensed point is then used to calculate \vec{Q}_C^A

and \vec{Q}_C^B as:

$$\vec{Q}_C^{[A,B]} = \begin{cases} (T_{[A,B]}^W)^{-1} \vec{P}_C^W, & \text{if } \|\vec{e}_T\| \leq \sigma_a, \\ \text{Recompute,} & \text{otherwise.} \end{cases} \quad (6.2)$$

Where \vec{Q}_C^A is the position of the sensed point \vec{P}_C^W in Body A and \vec{Q}_C^B is the sensed point's position in Body B. If the limiting condition $\|\vec{e}_T\| \leq \sigma_a$ is exceeded, the new value of \vec{P}_C^W is used to compute \vec{Q}_C^A and \vec{Q}_C^B based on Algorithm. 11.

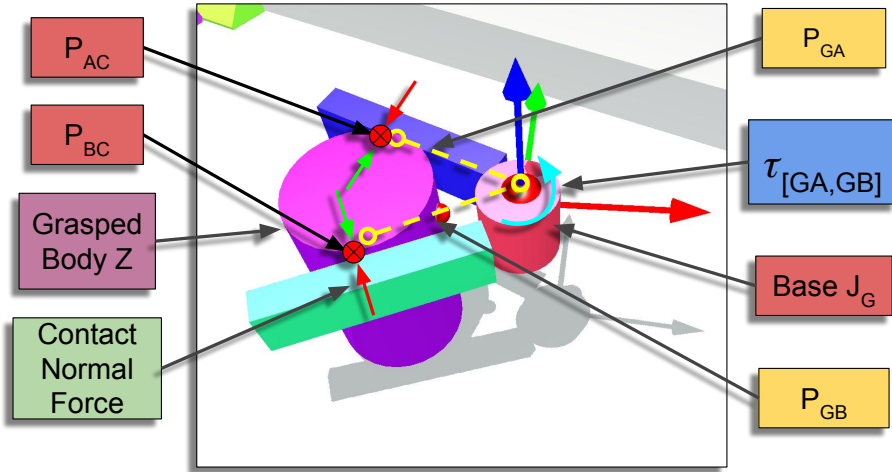


Figure 6.6: Grasping an object using a simple two finger gripper. The normal force at the contact points is required to compute the normal force \vec{F}_N for static friction computation.

Algorithm 11 Store Contact Points in Body Frames

```

1: if Sensor Triggered == True then
2:    $\vec{P}_C^W = \text{Get Sensed Point}()$ 
3:    $\vec{e} = T_A^W \vec{Q}_C^A - T_B^W \vec{Q}_C^B$ 
4:   if  $\vec{e}_{projT} > \sigma_a$  then
5:      $\vec{P}_C^{[A,B]} = (T_{[A,B]}^W)^{-1} \vec{P}_C^W$ 
6:      $\vec{Q}_C^{[A,B]} = \vec{P}_C^{[A,B]}$ 
7:   end if
8: end if

```

Next, the tangential error is calculated by:

$$\vec{e}_T = (T_A^W \vec{Q}_C^A - T_B^W \vec{Q}_C^B)_{proj T} \quad (6.3)$$

Similarly, \vec{F}_N , which is the normal contact force is calculated as:

$$\vec{F}_N = \vec{K}_N \vec{e}_N + \vec{K}_D \delta \vec{e}_N / dt \quad (6.4)$$

This force is based on the penetration depth of the sensed point w.r.t. the sensor limits. This depth is normalized using Equation 6.5 and then projected along the sensor's direction in the World frame.

$$\vec{e}_N = \left(\frac{\vec{P}_C^W - T_A^W \vec{P}_{rayStart}^A}{T_A^W (\vec{P}_{rayStart}^A - \vec{P}_{rayEnd}^A)} \right)_{proj \vec{N}_A} \quad (6.5)$$

The terms $\vec{P}_{rayStart}^A$ and \vec{P}_{rayEnd}^A are the start and endpoints of the Resistive sensor in the Body A frame, and $\delta \vec{e}_N / dt$ is to provide damping to the contact normal force.

Finally, the sliding friction force is computed from the differential velocities of contact points of Body A and Body B.

$$\vec{F}_V = \mu_V * \vec{e}_V \quad (6.6)$$

In physics computation libraries, kinematics and dynamics are usually expressed in the World frame, thus if the velocity of bodies A and B are $(\vec{V}_A^W, \vec{\omega}_A^W)$ and $(\vec{V}_B^W, \vec{\omega}_B^W)$ in the world frame respectively, then:

$$(\vec{v}_{C[A,B]}) = (R_{[A,B]}^W)^{-1} \vec{V}_{[A,B]}^W + (R_{[A,B]}^W)^{-1} \vec{\omega}_{[A,B]}^W \times T_W^{[A,B]} P_C^W \quad (6.7)$$

In Equation 6.7, the rotation $(R_{[A,B]}^W)^{-1}$ is not collected outside as the cross product $(R_{[A,B]}^W)^{-1} \vec{\omega}_{[A,B]}^W \times T_W^{[A,B]} P_C^W$ is not commutative. These velocities are converted

back to the World frame using Equation 6.8. The $\vec{v}_{C[A,B]}^W$ notation emphasizes that the velocity of sensed point (C) relative to Body A, and Body B, is expressed in World frame:

$$\vec{v}_{C[A,B]}^W = R_{C[A,B]}^W \vec{v}_{C[A,B]} \quad (6.8)$$

Which leads to the sliding error:

$$\vec{e}_V = (\vec{v}_{CA}^W - \vec{v}_{CB}^W)_{projT} \quad (6.9)$$

The three components of force are summed together according to Equation:

$$\vec{F}_{total} = \vec{F}_N + \vec{F}_V + \vec{F}_S \quad (6.10)$$

This force can now be used to compute the resulting moment on Body A and Body B by converting it back to Body Frames.

$$\vec{\omega}_{[A,B]}^W = R_{[A,B]}^W ((R_{[A,B]}^W)^{-1} \vec{F}_{total} \times (T_{[A,B]}^W)^{-1} \vec{P}_C^W) \quad (6.11)$$

The final force and moment are then applied to both Body A and Body B as action/reaction wrenches.

6.5.3 Visualization of Contact Forces

The static force computation described in Section 6.5.2 can be used model stick-slip friction. Interestingly, the magnitude of this static friction force can be visualized as an inverse cone shown in Figure 6.5 (a). This static friction force depends on the penetration depth based on Equation 6.5. Figure 6.5 (b) shows the deformation of the contact surface formed by Resistive sensors as it penetrates another body. The

normal force introduces “softness” to contact dynamics and improves the friction response, which can be leveraged to loosely mimic natural manipulation using soft contacts without the explicit use of soft-body simulations. However, a normal force might not be desirable in certain applications. Such cases can be implemented by modification of Equation 6.1 as follows:

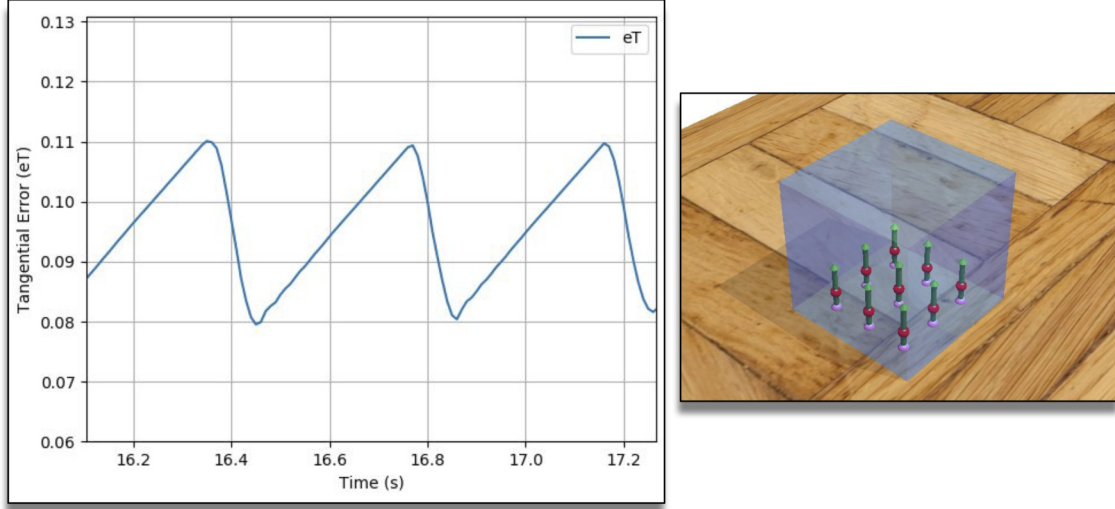


Figure 6.7: Static friction response of body-mounted with Resistive sensors (transparent blue box) sliding along the plane underneath (shown with the wooden texture) subject to a constant force applied along the direction of the ground plane. The response is calculated as the tangential error (i.e. the difference between the commanded and current position of the blue box).

$$\vec{F}_S = \mu_S * \vec{e}_T * \|\vec{e}^N\| \quad (6.12)$$

Where the depth is the normalized fraction of the sensor’s penetration depth. The underlying equations used to represent the behavior of Resistive sensors are based on the combination of both penalty based methods for contact computation and the classical Coulomb friction model. However, these equations are slightly different in their application. This difference results from the fact that instead of two bodies colliding with each other, it is the Resistive sensors mounted on a Body A which penetrate Body B. Thus, there is no common normal of collision at this

instance, instead, it is the direction of the sensor that is used for the computation of the normal force \vec{F}_N (Eq. 6.4) as well as the static and sliding friction force \vec{F}_S (Eq. 6.1) and \vec{F}_V (Eq. 6.6). The damping K_D assists in the stability of the normal force by reducing “jitter” that is generally associated with penalty based methods.

6.5.4 Automating Sensor Placement

An obvious question arises about the placement of Resistive sensors along the body. For very simple meshes, containing a few faces, it is possible to place the sensors individually, however, it is impractical to place an array of Resistive sensors on simulated dynamic bodies represented by complex meshes. After all, the density and curvature of the Resistive surface play an important role in contact dynamics. Two approaches are proposed for sensor placement which are, sensor placement (1) based on the visual mesh of the object and (2) based on a separate parametric mesh specified alongside the visual mesh. These meshes are called the “source meshes” and are used according to Algorithm. 12. The resulting sensor placement is shown in Figure 6.8. The primitive patches are wrapped around the visual mesh using Algorithm. 13.

Based on the vertex and triangle data of the source mesh, the surface is covered using the trivial Algorithm. 12. This algorithm can be expanded to cover edges and vertices. Using a separate mesh to define the surface of placement has many advantages over using the visual mesh of the object. For gripping tasks, the resistive surface forming the grasp closure is of more interest. Thus a mesh covering only the specific surfaces may be used. Furthermore, the parametric mesh can be defined to smooth out sharp corners of the visual / collision mesh.

Real-world rigid bodies may contain surfaces that are represented by different friction coefficients. This effect is hard to replicate by rigid-body dynamics as a

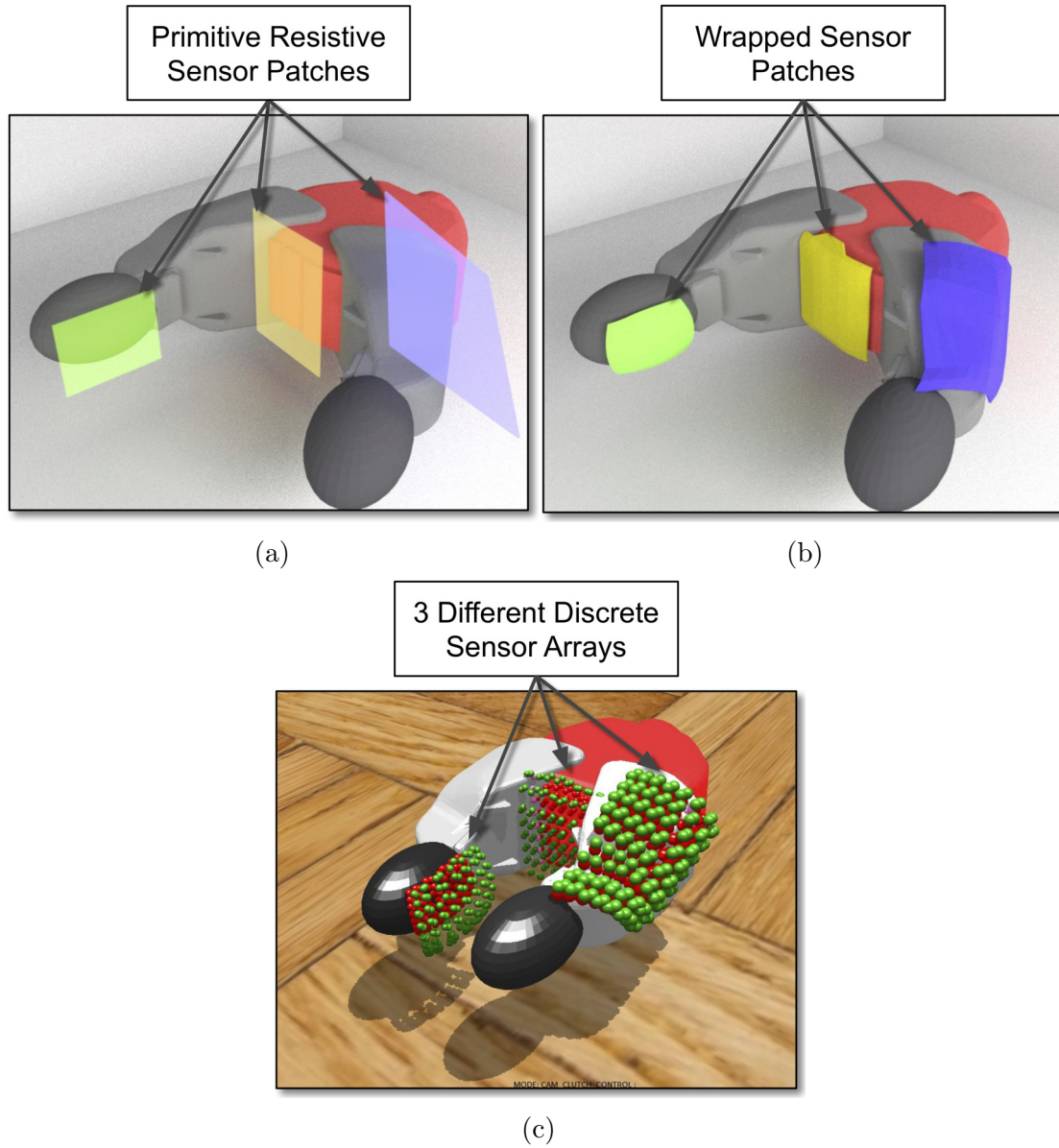


Figure 6.8: (a) Primitive Patches for Resistive Sensor Placement. (b) The primitive shapes can also be "skin-wrapped" to match the contours of the underlying complex shape. Figure (c) Sensor placement on the simulated gripper with red spheres representing the $P_{rayStart}$ and the green spheres representing P_{rayEnd} .

friction coefficient is an attribute of the entire body. By placing Resistive sensors with separately defined parametric meshes, the different surfaces of the body can be "coated" with sensors with different coefficients of friction, surface stiffness and even range of the Resistive sensors.

Algorithm 12 Populate Sensors Along Body Surface

```
1:  $D := \text{Param. Depth}, R := \text{Param. Range}, M := \text{Mesh}$ 
2:  $Triangles \leftarrow M$ 
3: for  $T \in Triangles$  do
4:    $\vec{vtx}_0, \vec{vtx}_1, \vec{vtx}_2 \in T$ 
5:    $\vec{edge}_0 = \vec{vtx}_1 - \vec{vtx}_0, \vec{edge}_1 = \vec{vtx}_2 - \vec{vtx}_1$ 
6:    $\vec{midpoint} = \vec{vtx}_0 + \vec{vtx}_1 + \vec{vtx}_2$ 
7:    $\vec{n}_f = \vec{edge}_0 \times \vec{edge}_1 / \|\vec{edge}_0 \times \vec{edge}_1\|$ 
8:    $\vec{P}_{rayStart} = \vec{midpoint} - \vec{n}_f D$ 
9:    $\vec{P}_{rayEnd} = \vec{P}_{rayStart} + \vec{n}_f R$ 
10: end for
```

Freely available software can be used to generate and modify meshes. Blender was used in this case for creating skinned meshes. Afterward, their subdivisions were created to generate Resistive surfaces.

Algorithm 13 Wrapping Primitive Around Visual Shape

```
1:  $S \leftarrow \text{Visual Shape}, M \leftarrow \text{Primitive Mesh}$ 
2:  $p_{off} := \text{Param. Offset}$ 
3: for  $v \in M.Vertices$  do
4:    $p_v := v.position, n_v := v.normal$ 
5:    $p_c := \text{Contact Point on } S \text{ of Ray along } n_v$ 
6:   if Contact Occurred then
7:      $f_c := \text{Nearest Face of } S \text{ to } p_c$ 
8:      $n_c := f_c.normal$ 
9:     if  $\text{dot}(n_c, n_v) < 0$  and  $p_{off} \leq (p_c - p_v) \cdot n_v$  then
10:       $l_v = p_c - \frac{(p_c - p_v) \cdot n_v}{\|n_v\|^2} * n_v$ 
11:       $p_v = l_v$ 
12:     else
13:       Discard  $v$ 
14:     end if
15:   end for
16: end for
17: Recompute Normals for  $M$ 
```

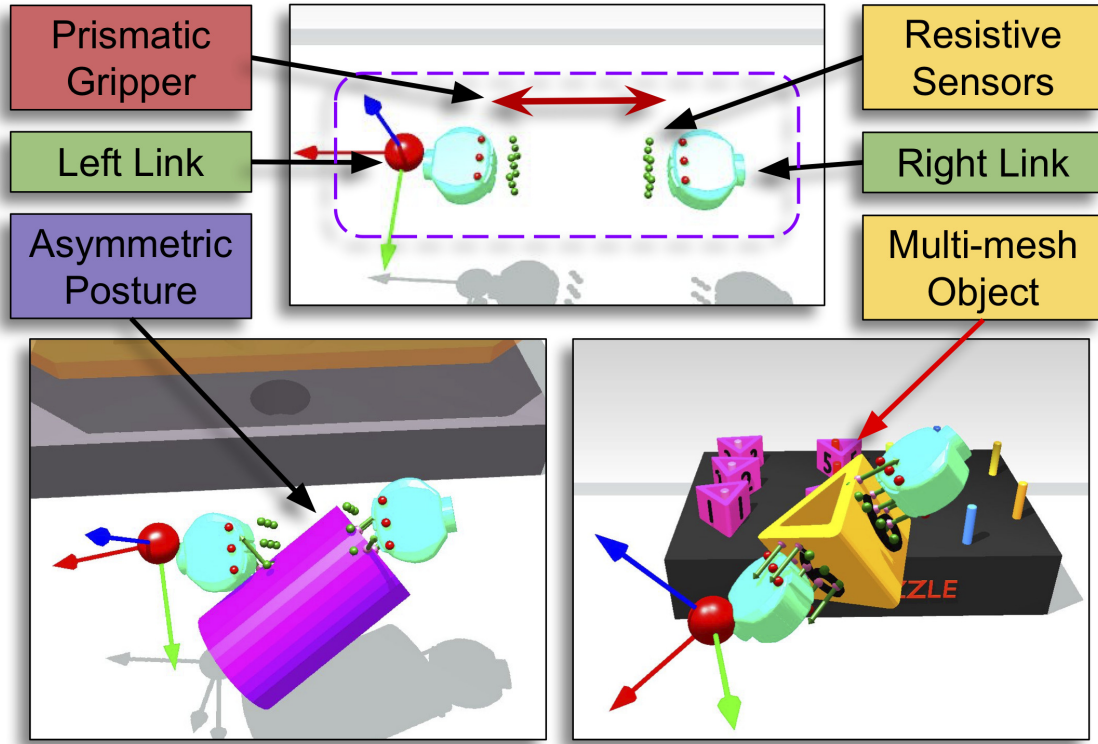


Figure 6.9: A simple prismatic gripper with two links, mounted with an array of Resistive sensors at ends facing each other. The bottom two figures show the grasping of a magenta cylindrical object with an asymmetric posture and the grasping of a yellow object that is composed of multiple collision meshes.

6.6 Results

The dynamic environments shown in this section have been designed specifically for the demonstration of manipulation using the proposed approach. The SDEs have been controlled using various IIDs including Geomagic Touch, Razer Hydras and the dVRK MTMs. First, the stability of the Resistive sensors is demonstrated by placing a sensorized box on an inclined plane and then recording its position over time. The response is shown in Figure 6.10.

Natural manipulation involves grasping complex objects at asymmetric postures which is demonstrated in Figure 6.9. Such interactions show potential scenarios that are not only applicable to interactive simulators for surgical training but also

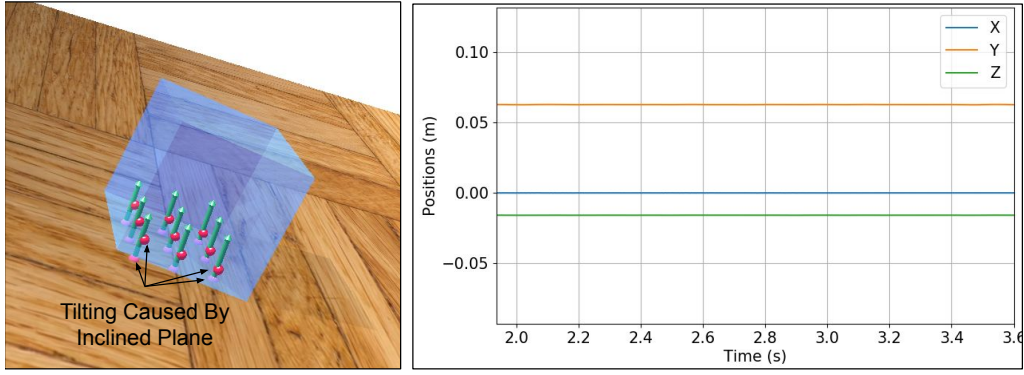


Figure 6.10: Stability analysis on an inclined plane. $m = 0.5Kg$, $K_s = 5000$, $\sigma_a = 0.001$, $K_n = 1$, $K_D = 50$ and $\mu_v = 0.1$.

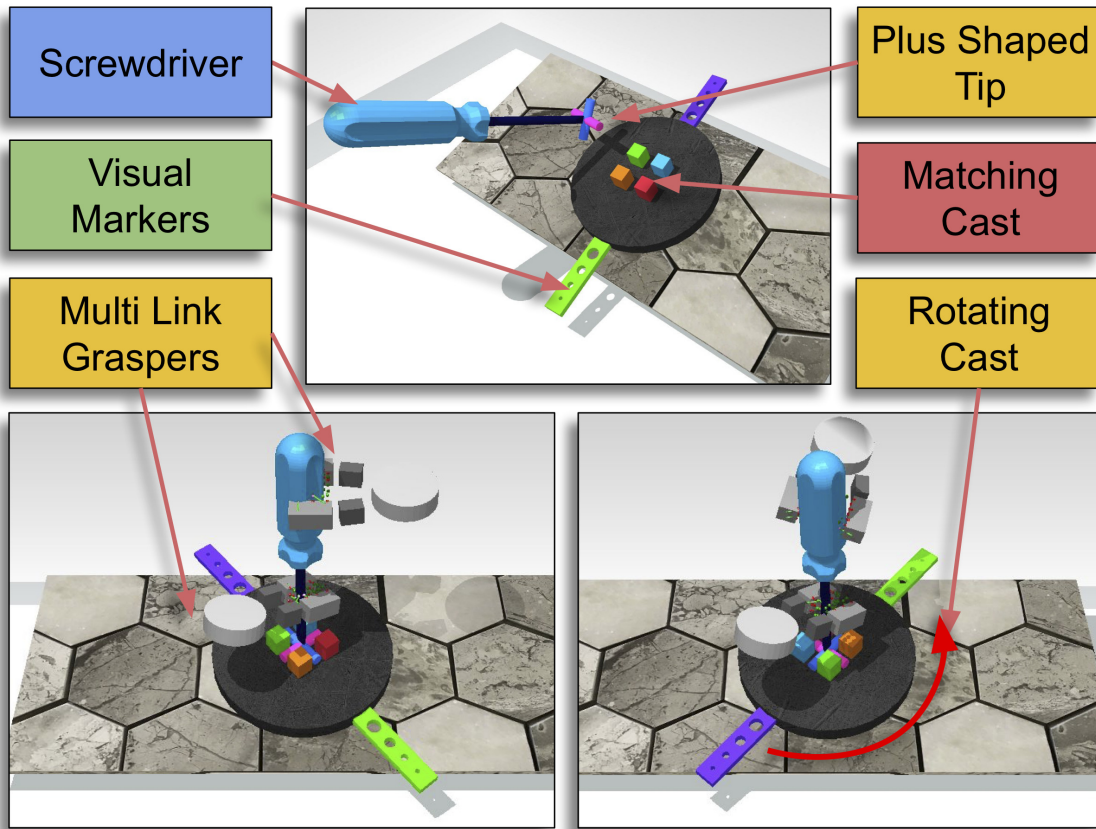


Figure 6.11: Bi-manual manipulation of a screwdriver to rotate the cast assembly underneath

entertainment and gaming simulators. The procedure of contact dynamics using Resistive sensors takes away the factor of varying geometrical shapes from grasp mechanics.

Table 6.2: Parametric Data for Specified Tasks

Task	Obj. Mass (Kg)	μ_s	μ_v	K_N	K_D	$\sigma_a(m)$
Obj. Grasp	0.4-0.8	1000	0.5	1.0	5	0.001
Screwdriver	0.6	5000	0.8	0.8	0.1	0.001
Thread	0.002/Prim.	1000	0.3	0.05	5	0.005

Figure 6.11 shows a more challenging scenario that mimics a two-handed screwdriver operation. The screwdriver is first grasped and then inserted into the cast (matching the tip shape of the screwdriver) via bi-manual manipulation (controlled via haptic input interface device). After insertion into the cast, the minor hand softens the grip, thereby reducing static friction according to Equation 6.1, and the dominant hand rotates the tool to rotate the cast underneath. The tasks can be carried out repeatedly by tightening the non-dominant hand and softening the dominant hand and re-orienting to a comfortable pose for rotating the screwdriver. Using constraint-based grasping, such a scenario would require pre-planning at the time of picking such that the user holds the objects to accommodate switching between the hands as the dominant hand would require the release of the grasped object by the non-dominant hand for rotation. Lastly, using kinematic simplification for grasping by making the tool static and affixed to one simulated end-effector would make the initial positioning in the cast impossible.

Similarly, multiple connected objects can be grasped and manipulated. Figure 6.12 illustrates a task involving the manipulation of a deformable thread around the puzzle. The parametric values used for these three examples are presented in Table. 6.2. The response of the AMBF simulator during the two-handed screwdriver operation is shown in Figure 6.13. As illustrated, the dynamic frequency of the simulation varies throughout but the Real-Time Factor (RTF) stays constant.

The stiffness achieved through the inclusion of Equation 6.4 introduces a soft

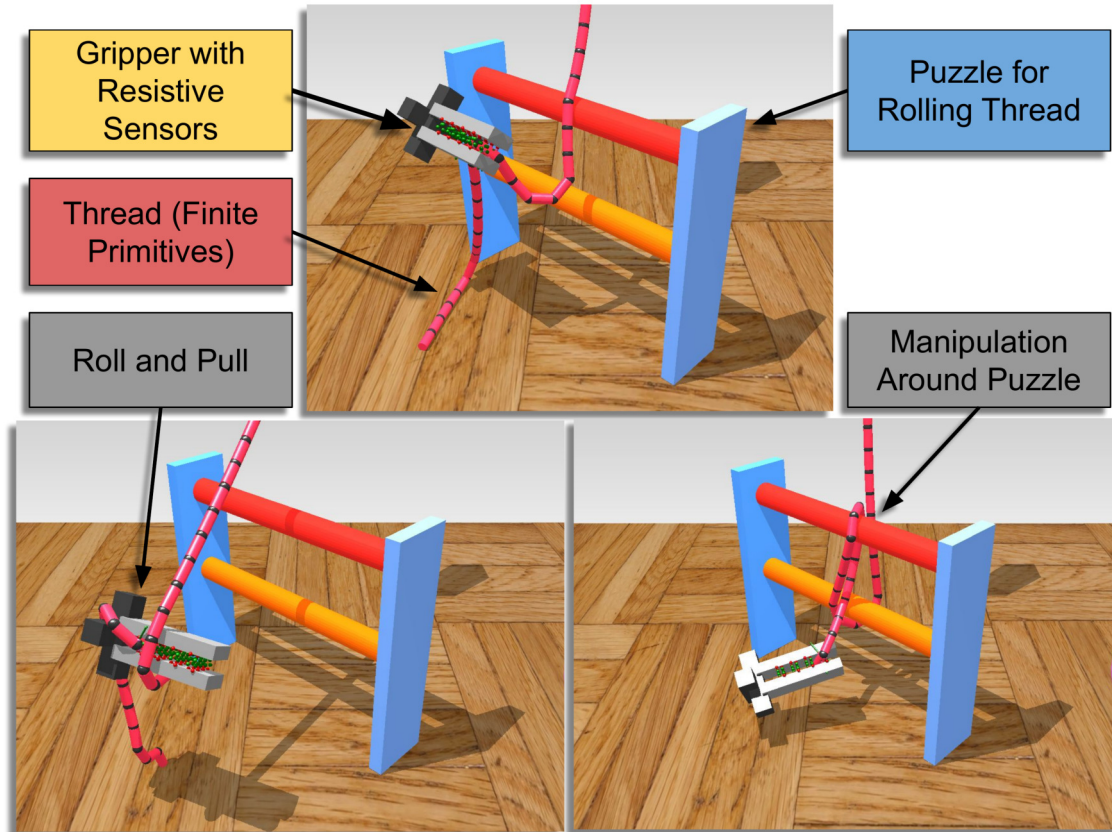


Figure 6.12: Manipulation of a deformable thread around the gripper jaws and around a puzzle.

feel to grasping. This is useful as it shows that objects do not have to be grasped symmetrically and also extends the grasp to allow manipulation without the use of soft-body dynamics. For visualization purposes, a rendered skinned mesh can be used for the surface comprising the Resistive sensors. The vertices of this skinned mesh can be anchored to the sensed point of each Resistive sensor.

6.7 Discussion

This chapter presented a parametric approach to tackle the problem of grasping and manipulation in simulation using Resistive sensors. While the initial results are promising, the proposed approach has a few limitations. One major limitation

is the number of parameters required to define the friction response of individual Resistive sensors. These parameters vary based on the scope of simulation and require tuning using a combination of empirical and analytical methods. The problem is compounded by the unbounded nature of friction coefficients as high values can render the grasp unstable and lower values result in insufficient grip forces.

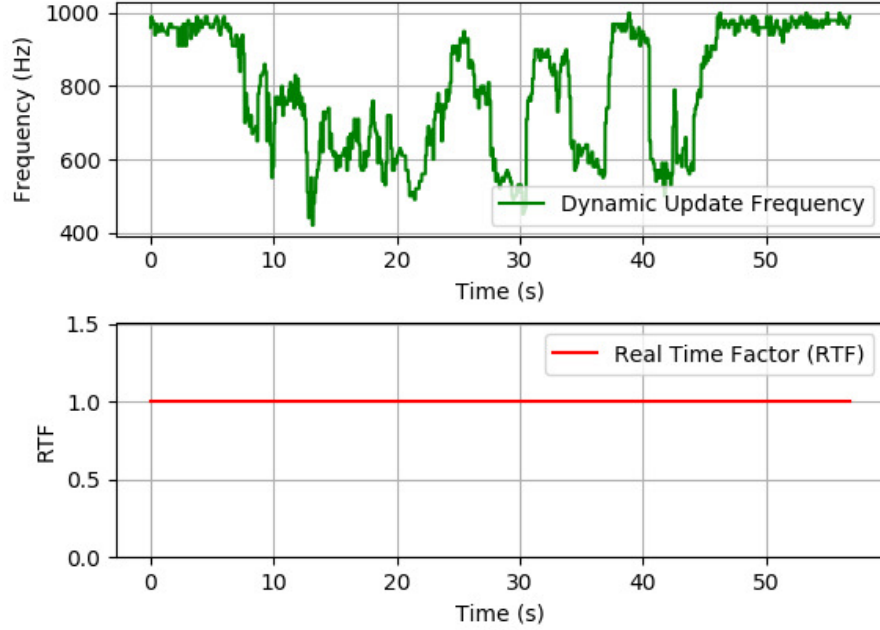


Figure 6.13: The dynamic update frequency of real-time simulation for the two-handed screw driver task and the real-time factor.

Each Resistive sensor requires the calculation of ray-tracing which is an expensive operation. For a large array of Resistive sensors, the required computation time will adversely affect the speed of the dynamics solver for real-time physics. The advent of hardware specialized for Ray Tracing (<https://developer.nvidia.com/rtx>) can potentially be leveraged to compute the response from Resistive sensors in parallel. The encapsulation of relevant data for the Resistive sensors means that each sensor can be computed independently at each time-step of the physics simulation. Rather than relying on dedicated GPU hardware, the computation can also be performed

in parallel by using multi-threading and batching a group of sensors together.

Chapter 7

Applications and Use-Cases

The AMBF simulator has already been used by several researchers across North America and Europe for various applications despite its nascent release. These applications include cooperative learning and training for surgical task automation, dynamic simulation of lower limb exoskeletons and impedance control of simulated dVRK manipulators to name a few.

To demonstrate the utility of AMBF for actual user-studies, two applications are presented in this chapter. These applications are 1) Supervised semi-autonomous control with Bayesian optimization and 2) Analysis of collateral control on surgical training tasks. It is important to point out that while these applications show the feature set and ease of use of AMBF, both of the user-studies are valid candidates for evaluation actual surgical training tasks. In that respect, while the first user study is more or less possible with other community-based software, one would still need to modify the core parts of these simulators to achieve hard real-time physics updates and establish the command and communication pipeline that AMBF inherently supports. More details are discussed in Section 7.1. The second user-study is an application that relies on the XVII Frame representation, discussed in Section 3.5,

which allows four primary modes of collaborative control. The details of this study are presented in Section 7.2.

7.1 Supervised Semi-Autonomous Control with Bayesian Optimization

Junhong and Dan-Dan from Imperial College London were the first to employ the AMBF simulator for a two part user-study. The work has been submitted for review as:

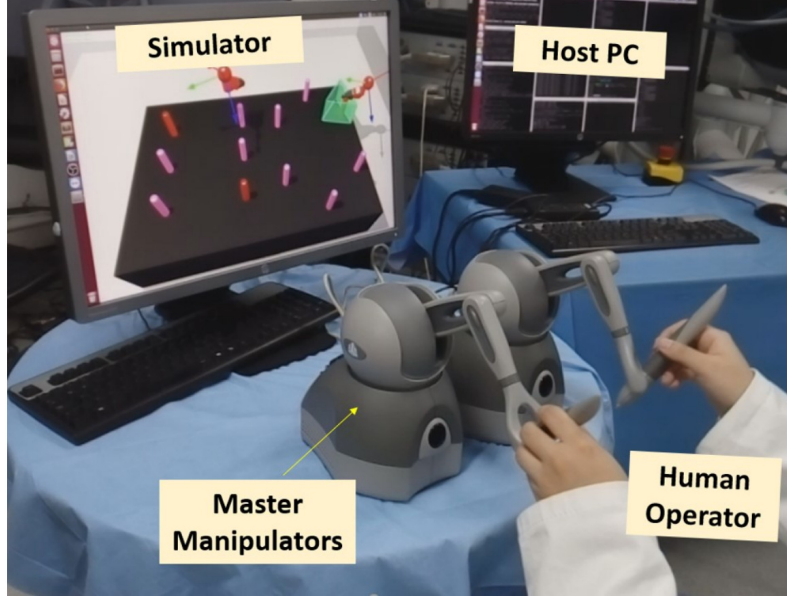
Junhong C, Zhang D, Munawar A, Fischer G S, Yang G Z, “*Supervised Semi-Autonomous Control for Surgical Robot Based on Bayesian Optimization*”, In Review for International Conference on Robotics and Automation (ICRA), 2020.

The results and figures used in this chapter have been taken from the submission listed above. As part of the user-study, the test subjects were asked to perform the peg transfer task with and without supervised assistance. Figure 7.1 illustrates the overview of the task. The supervised semi-autonomous control setup can be summed up using the flowchart in Figure 7.2. The task performance of the test subjects was evaluated based on the following parameters:

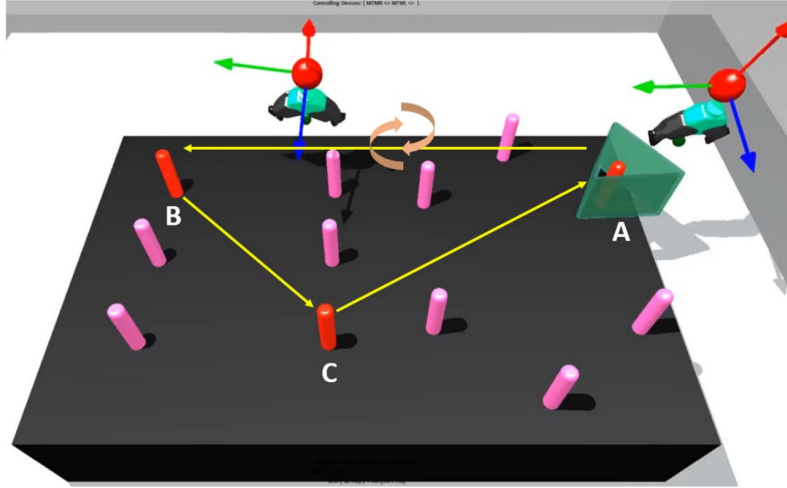
1. The total path length traversed by the IIDs (P).
2. The frequency of clutching (T).
3. The task-completion time (C).
4. The average velocity of the SDEs (A).

Furthermore, the supervised assistance was calibrated with Bayesian optimization. The qualitative analysis of the Bayesian optimization was performed by com-

paring it with the supervised task performance without the optimization.



(a)



(b)

Figure 7.1: (a) The study setup involves the human-subjects looking at the screen where the interactable simulation is being displayed. (b) The goal of the exercise is to pick the peg located at A using the right SDE, handing it over to the left SDE and placing it at B. Then picking back the peg at B and placing it at C using the left SDE. Finally, switching hands to use the right SDE to pick and place the peg back at A. [3].

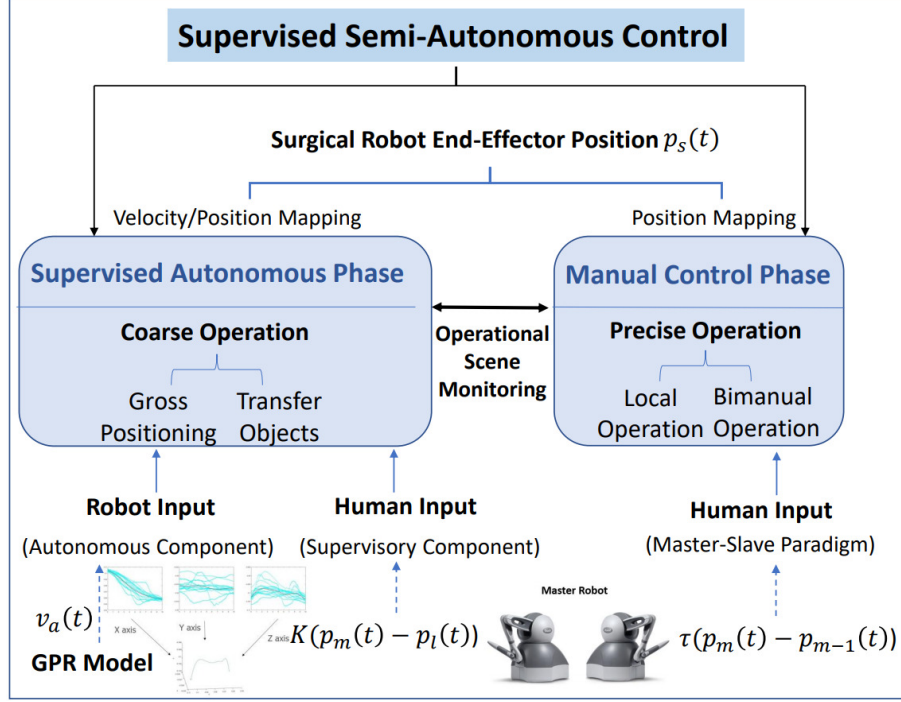


Figure 7.2: The supervised semi-autonomous control scheme for assisting the human subjects in surgical task performance [3].

7.1.1 Design of Training Puzzle

The puzzle pieces shown in Figure 7.1(a) are triangular in shape with holes (cavities) inside. This helps in placing these pieces on cylindrical pegs. However, the presence of this cavity renders the collision shape non-convex. As discussed in Chapter 3, a collision mesh representing a non-convex shape can be unstable at lower frequencies of the dynamic update loop. Since the underlying puzzle was meant for user training, it was desirable to have a consistent simulation without the collision objects suddenly jumping and altering the study data. To cater to this scenario, the ADF was extended to define a compound of primitive shapes to represent a single collision shape. This feature was integrated at a framework level in the ADF without breaking backward compatibility. Using this feature, one can define arbitrarily complex collision shapes, both convex and concave, using any number of simpler primitives.

The entire simulated puzzle was designed using the ADF Blender Add-on [105]. The takeaway here is that this puzzle involves a large number of collision objects and relative transforms that one has to deal with for accurate initial placement. Using the provided ADF Blender Add-on, one can rapidly generate such complex puzzles and tweak them with ease in a matter of seconds as compared to every other solution that exists out there. Similarly, one can disable parts of the puzzle from loading just as easily using the header list feature discussed in Section 4.3.1.

In terms of inputs to the simulation, the study investigators requested the use of Geomagic Touch haptic devices as they offer a smaller form factor. Although these devices are natively supported in the AMBF Simulator using the CHAI-3D device driver interfaces, additional button events were required for switching different modes of control during the study. Instead of temporarily tweaking the device driver code for the specific applications, the highly customizable interface of AF Arm Plugin was leveraged. For this purpose, a light-weight Python application [139] was developed that expanded some of the capabilities of the AF Arm Plugin without having to modify the plugin itself.

7.1.2 Study Results

12 human subjects participated in the study which was conducted by Junhong Cheng and Dan-Dan Zhang at the Hamlyn Center in London. The user-studies were divided into two parts. The first study was a comparison between the performance of a task performed manually versus being performed with semi-autonomous supervision. The second study was a comparison between the performance of the supervised semi-autonomous control task with and without Bayesian optimization.

The result of the first part of the user-study is demonstrated in Table 7.1. Supervised semi-autonomous control resulted in reduced clutching frequency (C) and

Table 7.1: Comparison between the task performance of manual and semi-autonomous supervisory control

Params	Manual	Semi-Autonomous	Significance (p-value)
$M(m)$	4.70 ± 1.43	2.37 ± 1.50	0.0000
$T(sec)$	100.35 ± 48.48	111.30 ± 45.0	0.0037
$A(mm/s)$	10.826 ± 10.11	10.03 ± 2.94	0.9337
C	15.8 ± 6.2	8.0 ± 6.40	0.0012

shorted path traversal (P) compared to fully manual control whereas the completion time (T) and average SDE velocity (A) did not change significantly. Figure 7.3 presents the results of the application of Bayesian optimization to supervisory control. Table 7.2 shows the numerical values of the study parameters with the corresponding statistical significance. As observed, the application of Bayesian optimization resulted in lower values of task completion time (T) and the average path traversed by IIDs (P).

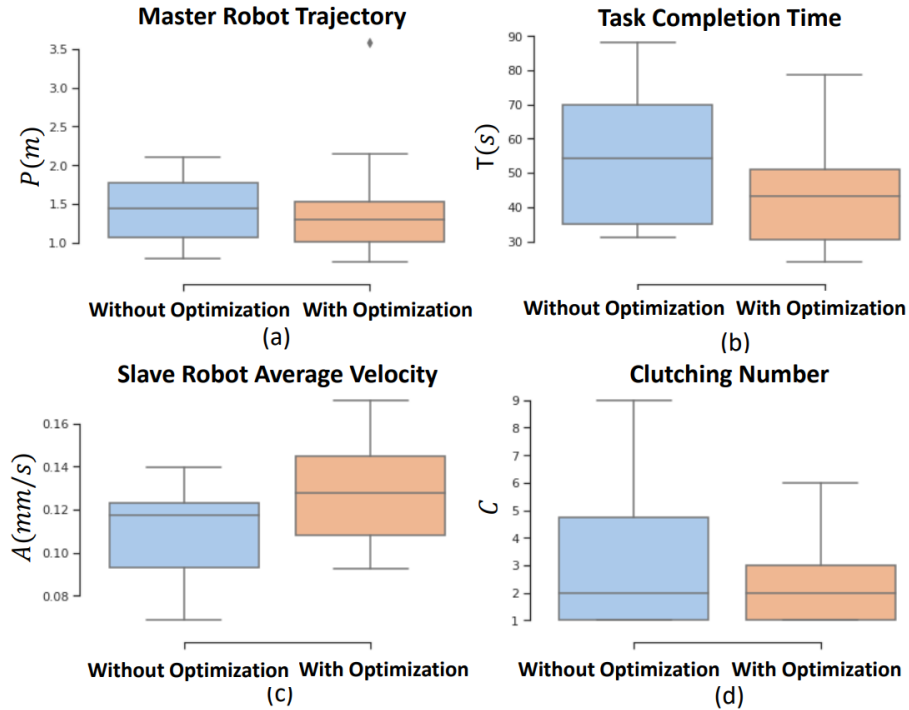


Figure 7.3: The box plots show the results of semi-autonomous control assistance with and without optimization through Bayesian learning. [3].

Table 7.2: Comparison between the task performance of supervised semi-autonomous control with and without Bayesian optimization.

Params	With Optimization	Without Optimization	Significance (p)
$M(m)$	1.48 ± 0.51	1.44 ± 0.66	0.2145
$T(sec)$	55.47 ± 20.64	43.34 ± 15.03	0.0038
$A(mm/s)$	11.01 ± 2.12	12.77 ± 2.48	0.0280
C	3.3 ± 2.5	2.4 ± 1.4	0.0018

This user-study showed the potential of the AMBF simulator for surgical training (and general-purpose training) tasks. The inclusion of various control modes and the use of the AF Arm plugin interface for replacing Geomagic Touch drivers were possible due to the distributed nature of AMBF. Finally, the data collection was done using the easy to use interfaces exposed by the AMBFs communication pipeline.

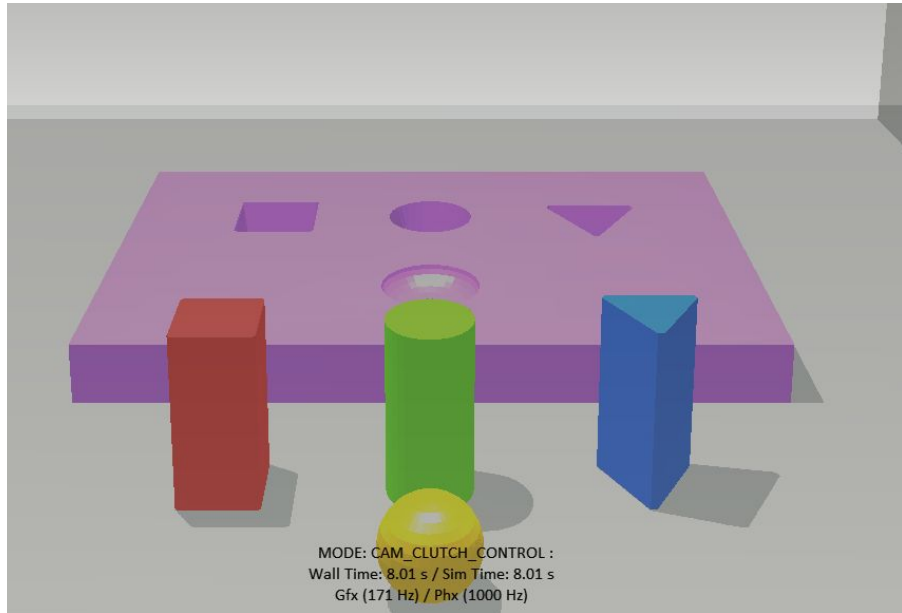


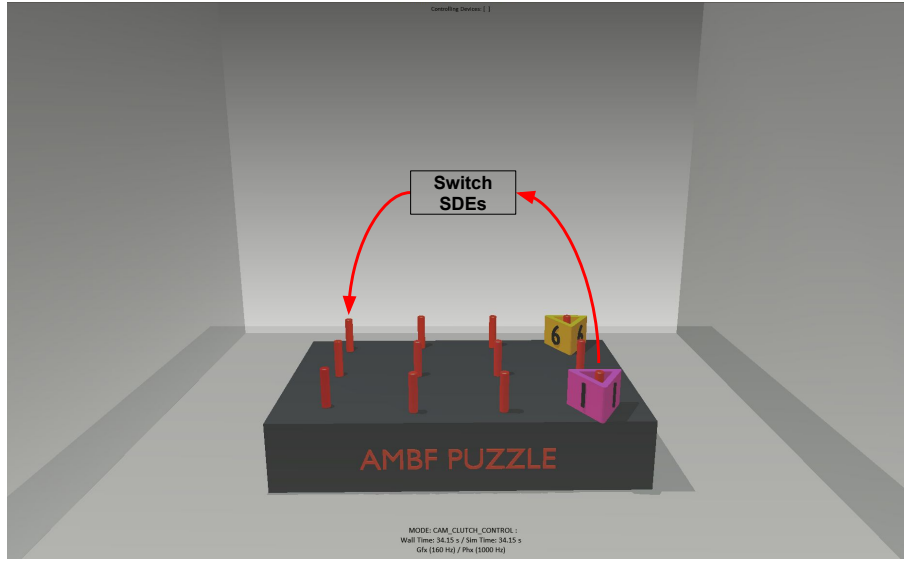
Figure 7.4: The training environment for getting the study subjects on a similar footing for the user-study. The goal of the environment is to pick and place the puzzle pieces (red, green, blue and yellow bodies) in the purple cast with matching intrusions. The subjects learned the use of clutching control and grasping to perform the task.

7.2 Analysis of Collaborative Control for Surgical Training Tasks

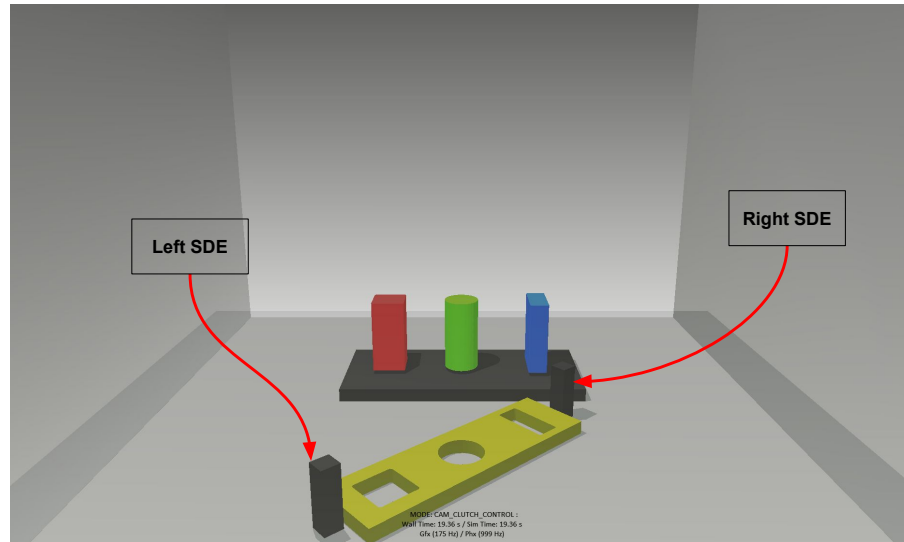
Different from the previous application, where the study showed the potential of the AMBF for learning applied to semi-autonomous task performance, this user-study did not involve any form of learning or AI. Instead, the purpose of this study was to analyze the performance of multi-manual tasks with and without collaborative assistance and to identify if assistive collaboration could improve the task performance, and to what extent. Haptic IIDs were used for this user-study. Prior to the recorded study, the test subjects were trained to get accustomed to the interface of IIDs and the general manipulation in the simulation using a test environment. This environment is shown in Figure 7.4. Even after the successful placement of the pieces in the cast, each subject was allowed as many trials and time as they requested to get accustomed to the control interface and the simulated environment.

After the training, the subjects were asked to perform 2 tasks of varying difficulty, 3 times each. Each task repetition can be summed up as:

1. Individual Task with Manual Control Mode: The first time, the task was performed individually by the test subjects with haptic feedback enabled. This can be called as the single or manual control mode.
2. Collaborative Task with SISO Control Mode: The second time, the same task was accomplished with assistance from another user (the study investigator). The second user can be referred to as the collaborator. Both the study subject and the collaborator could feel the corresponding forces (not moments) from each other as well as from the collision with the environment. The collaborator used Novint Falcons as IIDs which do not have orientation control, thus the



(a)



(b)

Figure 7.5: (a) Similar to the first user-study, the test subject is required to pick the pegs labeled 1 and 6 using the right SDE, then switching mid-air to the left SDE and placing the blocks in the opposite corners. The pegs can be transferred in any order. (b) This was a more involved task requiring simultaneous control input from both SDEs. The goal was to pick the yellow puzzle from the two handles (dark gray in color) and place it on the base with matching extrusions.

study subject was only assisted in position control.

3. Collaborative Task with SINO Control Mode: The third time, the same task was performed in collaboration but without any force feedback.

The two tasks can be seen in Figure 7.5. To make the study fair, the subjects were allowed to perform each of these two tasks as many times as they requested before the study.

7.2.1 Setting up the Study

Conducting this study using any other open-source simulator would not have been an easy task as apart from challenges to simulation, one would have to write specific controllers for collaborative control that deal with the associated transformations in addition to performing real-time haptic updates. AMBF, on the other hand, has been designed and developed since its inception to allow performing such types of user studies without having to write a single line of code in the core framework. The design and integration of the XVII frame representation, although complex at the onset, allows AMBF to be used as a truly heterogeneous simulator. To make two IIDs pair to a single SDE, the following fields are used which are specified in the “Input Device Specification” configuration file. These fields have been discussed in Section 3.5 but are presented here again.

- **Simulated MultiBody:** This is an optional field and defines the name of a multi-body in the AMBF representation format. Any body/robot/gripper can be defined using this field and loaded as a proxy for the IID.
- **Root Link:** This is an optional field as well and specifies the base link that is to be paired with the IID. For instance, a usual gripper, consisting of a palm and left and right fingers can be specified as the “simulated multibody” in the field above, and the root link can be specified as the name of the palm link. Afterward, the IID would be able to control all the lower joints from the palm link by normalizing all the joint limits.

Specifying the fields of “Simulated MultiBody” and “Root Link” can result in multiple different configurations that are elaborated in the flowchart shown in Figure 3.8. For the specific user-study, in the “Input Device Specification” configuration file, the dVRK MTM’s data blocks specify the “Simulated MultiBody” and the “Root Link”. The Novint Falcons do not specify the “Simulated MultiBody” but only the “Root Link” which is a link belonging to the SDEs that are loaded with the MTMs. This allows both the MTMs and Novint Falcons to bind to the corresponding SDEs.

Having achieved the pairing between the IIDs and SDEs, the control modes such as SISO and SINO can be achieved by tweaking the gains in the corresponding datablocks in the “Input Device Specification” configuration file. These fields are the linear and angular Cartesian and haptic controller gains. Both these gains are unique between as IID and an SDE. The haptic controller gains scale the error between the SDE and the IID that serves as the control input for the IID. Similarly, the Cartesian controller gains scale the error between the SDE and the IID to be fed as the control input for the SDE. That being said, since multiple IIDs can share a single SDE, there would be that many instances of Cartesian controllers associated with that SDE. Moreover, there is always going to be an additional Cartesian controller that listens to the *afObjectComm* instance of the SDE. There indigenous and parallel controllers make full use of the XVII Frame representation and the distributed nature of AMBF to allow asynchronous control.

The test subjects were required to use the dVRK MTMs as IIDs. In the actual da Vinci MTMs, the Gimbals have redundancy that allows low effort and seamless orientation control. The dVRK systems do not have this implemented yet. Thus to achieve a similar setup, a simple null-space controller was devised. This controller relies on the kinematic constraint illustrated in Figure 7.6.

The angle produced by the constraint is retrieved using the following set of

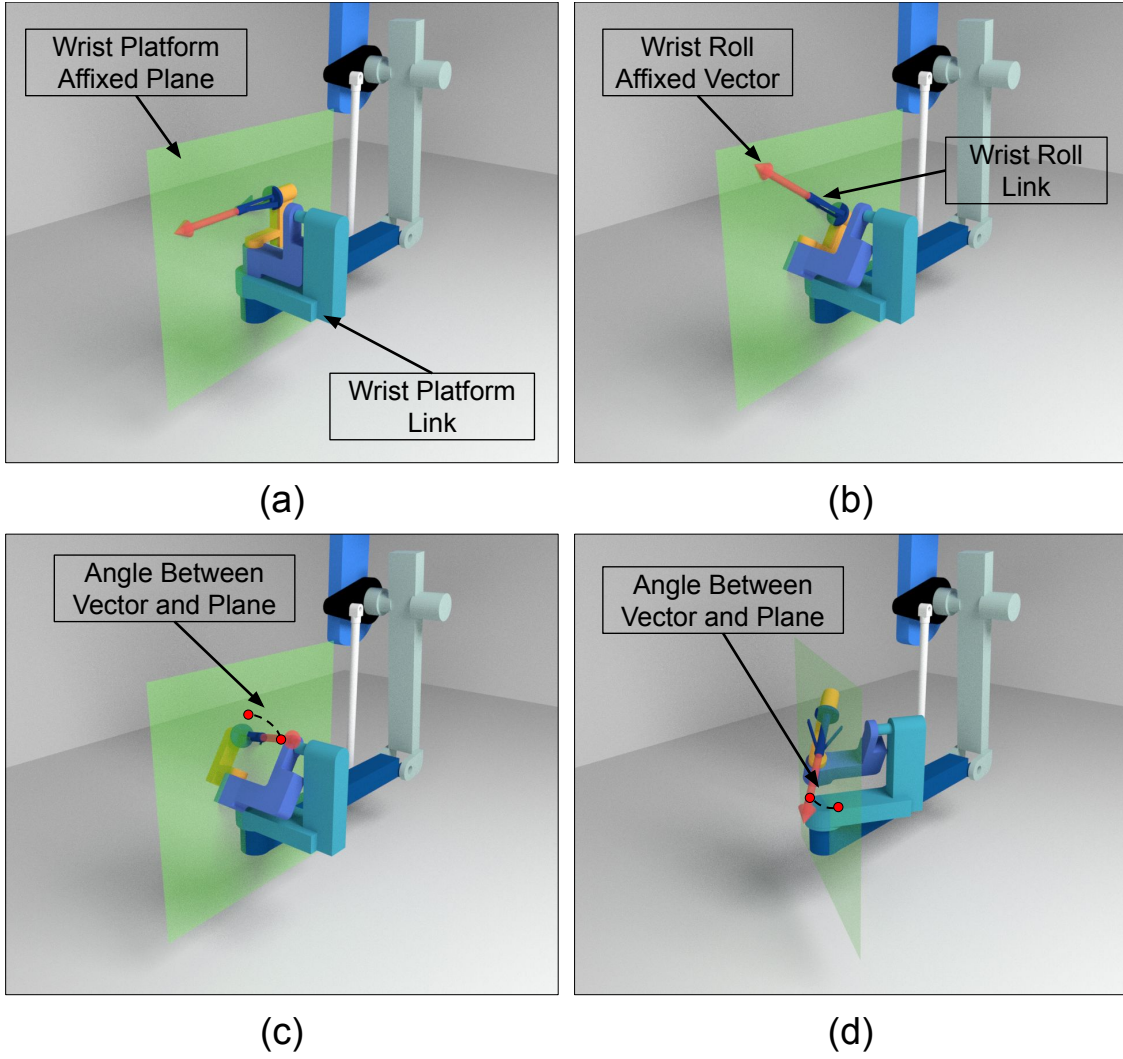


Figure 7.6: The MTM's wrist platform link is actuated to provide null space control by affixing a virtual plane (translucent green plane) to the platform link. Secondly, a virtual unit vector (red arrow) is affixed to the tip roll link. The angle between the red arrow and the green plane is used in a PD control law to rotate the wrist platform link.

equations.

$$R_{l4}^w = I_{3 \times 3}; \quad R_{l5}^{l4} = R_y(-\pi/2); \quad R_{l6}^{l5} = R_y(\pi/2); \quad R_{l7}^{l6} = R_x(\pi/2) \quad (7.1a)$$



Figure 7.7: The GUI for recording the user-study data.

$$R_4 = R_z(\theta_{j4}); \quad R_5 = R_z(\theta_{j5}); \quad R_6 = R_z(\theta_{j6}); \quad R_7 = R_z(\theta_{j7}) \quad (7.1b)$$

$$R_7^4 = R_{l4}^w R_{j4} R_{l5}^{l4} R_{j5} R_{l6}^{l5} R_{j6} R_{l7}^{l6} R_{j7} \quad (7.1c)$$

$$\vec{v}_7^4 = R_7^4 \vec{n}_z; \quad \vec{v}_{4_x}^w = R_4^w \vec{n}_x \quad (7.1d)$$

$$e_7^4 = (\pi/2) - \cos^{-1}(\vec{v}_7^4 \cdot \vec{v}_{4x}^{vw}) \quad (7.1e)$$

$$\tau_{j4} = K_p e_7^4 - K_d \dot{\theta}_{j4} \quad (7.1f)$$

Where the terms K_p and K_d are linear and damping gains. The terms R_{lx}^y are the fixed offset rotation matrices between link x and link y . Finally, the terms R_x are the variable rotation matrices that are parametrized by the corresponding angle of joint x . This basic implementation was appended to the output of the Cartesian space controller.

The data exposed by AMBFs communication pipeline contains ample information for classifying the performance of user-studies. However, to capture this data conveniently for multiple sub-studies and users, a lightweight Python application [140] was developed. A complementary GUI that is shown in Figure 7.7 helped in data collection using a minimal interface. The user name is specified at the top field and the corresponding study type is selected by the radio buttons. Afterward, pressing the “Start Record” button prepends the subjects’ ID and study type with the date and starts recording the data as ROS Bags [141]. The topics to be recorded are provided as a Python list rather than a single string which allows code reusability for other user studies. To sync the data, the Python code has a recordable built-in timer that initiates when the record button is pressed and terminates when the “Stop Record” button is pressed. This helps in classifying the length of the study in case the message clocks do not sync up.

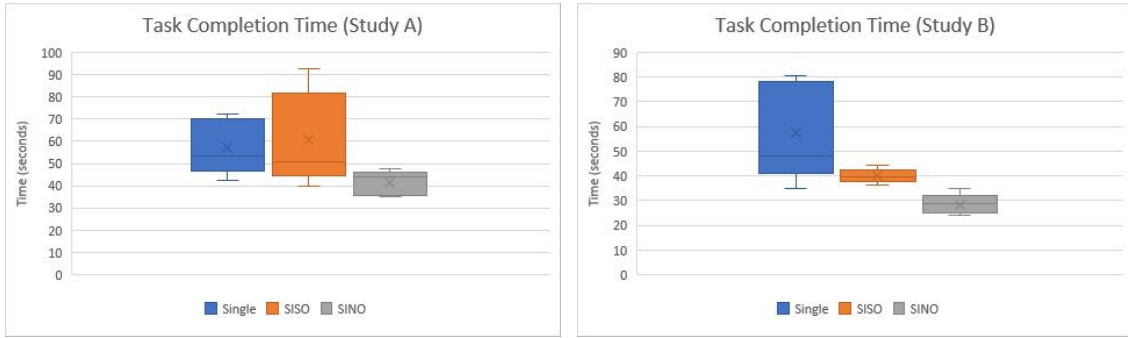


Figure 7.8: Task completion time between the two studies A and B and three control modes each.

7.2.2 Study Protocol

The subjects were seated in front of the dVRK console shown in Figure 7.9. The foot-pedal tray had the two control pedals. One foot-pedal allowed the re-positioning of the simulated camera (Camera Clutch) while the other pedal (Control Clutch) to engage/disengage the control of simulated graspers for re-positioning the MTMs. The subjects were trained to get used to the interface of the SDEs by performing the example task shown in Figure 7.4.

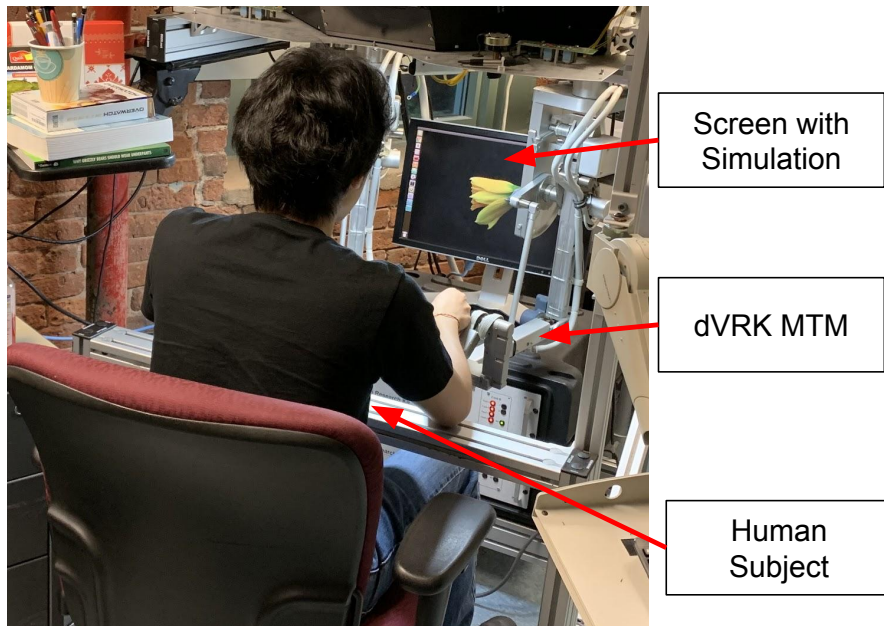


Figure 7.9: A example of a human subject performing the collaborative user-study.

The task performance of each task was measured by the following metrics:

- Time required for task completion.
- The average path traversed by the IIDs.
- The position clutching frequency.
- Subject's over-all comfort, measured using the Reduced NASA TLX protocol [142].

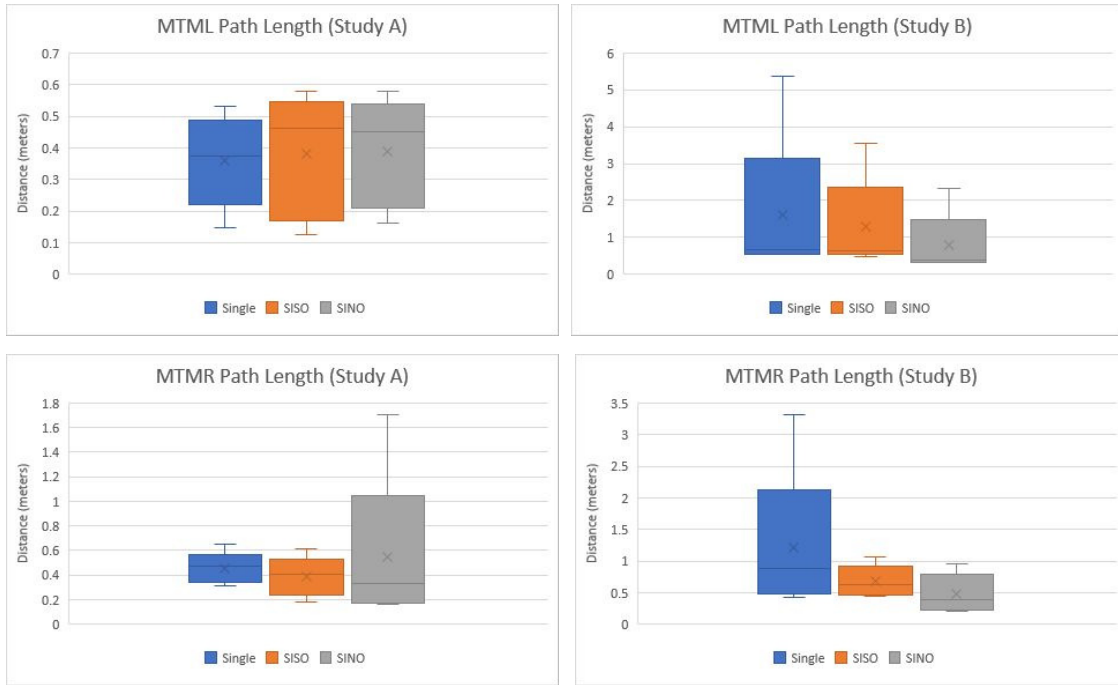


Figure 7.10: Length of the path traversed by IIDs between the two studies A and B and three control modes each.

7.2.3 Study Results

A total of 5 human subjects participated in the user-study. All of the study participants were right-handed. Two of the study participants had prior experience with teleoperating the da Vinci but this experience was limited to less than a couple of

hours of operation. The remaining three participants had never teleoperated the dVRK or the da Vinci before.

The completion times of the user-study are shown in Figure 7.8. Quite evidently, the SINO collaborative control modes have the least times for performing the tasks followed by the SISO collaborative control mode. Based on this result alone, there is evidence that collaborative control helped to speed up the required tasks. Figure 7.10 shows the average path traversal of MTMR and MTML for all the three control modes of each study task. Since the average path traversal fluctuated between different arms, the results lack any statistical significance and hence no conclusion can be drawn.

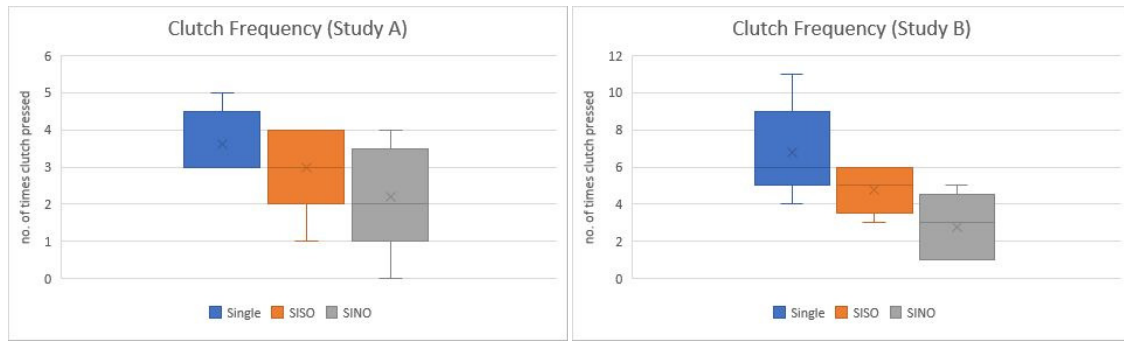


Figure 7.11: Clutching frequency between the two studies A and B and three control modes each.

Figure 7.11 shows the clutching frequency between the user studies. Interestingly, the average frequency drops between the three control modes for each study. This points to the subjects feeling more comfortable with the addition of collaborative control.

While the above three results are quantitative, the results demonstrated in Figure 7.12 show the real value of collaborative control from the perspective of the user comfort. It can be seen that the subjects felt more confident about the task with the addition of collaborative input. Surprisingly or unsurprisingly, the subjects felt more comfortable with the collaboration involving force feedback yet most confident

without force feedback.

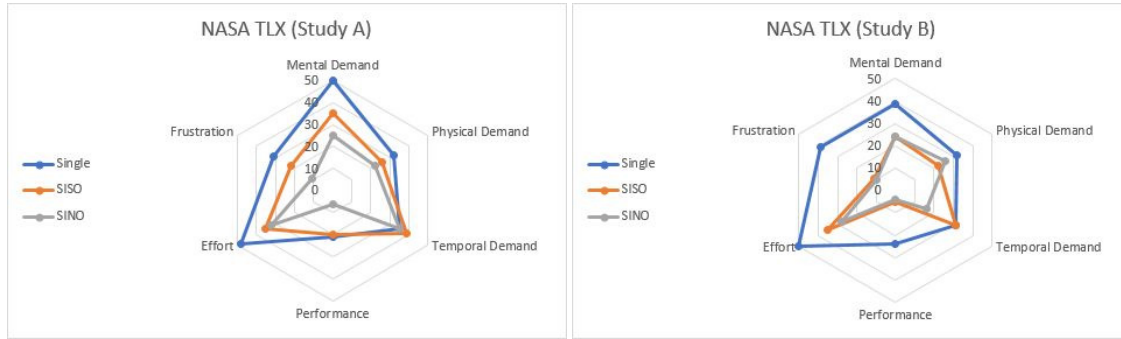


Figure 7.12: Average of Reduced NASA TLX questionnaire between the two studies A and B and three control modes each

7.3 Conclusion

Setting up user-studies for surgical training and evaluation is a time-consuming task. A significant amount of the time is spent in the placement and calibration of recording instruments as well as the actual training tasks. In many instances, the effort required does not reflect in either the results or the efficacy of the user study. Moreover, the nature of these studies makes it difficult to replicate across different research institutions. Two such user-studies were presented in the chapter. Both these user-studies can be carried out without the inclusion of AMBF, however, the specific applications demonstrate the ease of setting up AMBF for different kinds of user-studies. AMBF is a step in the direction of simplifying the process of setting up repeatable user-studies across research institutions.

Chapter 8

Conclusion and Future Work

This chapter reviews the contributions of this dissertation, discusses the lessons learned and then presents the future work related to the development of the Asynchronous Framework.

8.1 Review of Contributions

The following conceptual contributions have been presented in this dissertation.

1. A framework for simulating a real-time dynamic simulation alongside high-speed input interface devices.
2. An extensive frame representation for multi-user, multi-camera and collaborative input mapping.
3. A distributed communication pipeline for speed varying communication from the Asynchronous Framework.
4. Inclusion of action based sensors for simplifying common manipulation tasks.
5. A novel specification format for specifying complex robots and environments.

These conceptual contributions have been converted into implementations for the Asynchronous Framework and presented as different chapters in this dissertation.

8.2 Lessons Learned

The motivation that led to the design and development of the Asynchronous Framework was drawn from years of experience with other community-based software and simulators. The ever-changing API of these software, which might not have affected their core user-base, diminished the already scant support for complex surgical robots and heterogeneous input interfaces. It was not until 2016 when the first implementations which eventually led to the development of the Asynchronous Framework took place. As the adage goes:

“The journey of a thousand miles begins with one step [Laozi a.k.a. Lao Tzu]”

While this may seem cliched in general, and rightly so, and out of context for software development which usually starts from a broader set of specifications towards a narrower implementation, almost the opposite is true for the Asynchronous Framework. Initially, the implementations were carried out using the CHAI-3D [143] examples for static kinematic objects. Soon, these examples were expanded to include a pair of dynamic bodies (a donut and a peg) which were solved using Bullet Physics [34]. These simulated bodies were hardcoded into the software and were interacted via rudimentary dual-link, single joint and hard-coded SDEs. At that point, these implementations were simply called Extended CHAI-3D. The expansion beyond Extended CHAI-3D and towards Asynchronous Framework started in mid-2017 with the inclusion of multiple IIDs simultaneously that could achieve a real-time dynamic simulation. Fast forward to today, the Asynchronous Framework along with its simulation component, the AMBF, allows the flexible specification

and simulation of complex robots, environments and soft-bodies, all interactable with real-time IIDs with haptic feedback.

Although the simulation component of the Asynchronous Framework, competes with existing open-source simulators, it inherits the best design practices employed by many of the same software. For example, ADF is a result of not only the limitations of URDF and SDF, but also the good design practices employed by both. Similarly, several sub-components of the AF draw inspiration from the dVRK software [20].

The early development of a framework that was also being used in parallel by other researchers presented an interesting scenario. Many use-cases originated which had to be incorporated in the design. The complexity of the core framework required more work for the inclusion of these features as compared to other sequential simulators. However, it was the continuous evolution of the AF based on user feedback which made many unique features of AMBF possible. The Python client is a great example of this.

The more experienced the users of a new framework are, the more useful their feedback and feature-requests are. However, there is also a possibility of “experience bias” which goes against some of the core design philosophies. For example, since many of the AMBF users also use ROS, RViz and Gazebo, a request is often made to model ADF along some of the URDF’s and SDF’s frame specifications. While it is possible to do so, it goes against the future vision of AMBF. However, how can software have a future if it does not have sufficient user-base in the first place? This is a question that arises often concerning AMBF.

There has not been an easy answer yet to this question. However, diverse upfront examples that showcase advanced features of the software without having to delve into the design is a good attractor. Then, the users feel more comfortable to take

the time in learning the implementation details if they want more from the software. A prime example of this is CHAI-3D [143] which has ample examples for nascent users. Bullet physics [144] is another example. However, there is a key difference between the examples shipped with CHAI-3D as compared to Bullet. CHAI-3D's examples are meticulously designed with a lot of code repetition. While this might not seem like a good software engineering practice, it makes it much easier to learn CHAI-3D as compared to Bullet Physics. AMBF has only a few examples but these examples cover many advanced features.

8.3 Future Work

The Asynchronous Framework has achieved several milestones set during its initial development, however, more work needs to be done in many different areas. The sections below briefly discuss the types of future work that needs to be incorporated into AF and AMBF.

8.3.1 Object Specific Communication Payloads

At the moment, all the simulation components in AMBF use a single pair of communication payloads called *afObjectCmd* and *afObjectState*. While these payloads are generic can cover some kinematic aspects of all the simulation components, they are not ideal for scene objects. For example, visual entities such as lights and cameras do not use most of the fields of *afObject* message payloads. The appropriate fields for controlling cameras and lights are much different from controlling a rigid body. A camera is usually controlled by setting the view direction, the target view (position where to look at) and field of view, etc. Likewise, light objects are better controlled by specifying parameters such as shine direction and spot exponents.

Where these communication instances fall short are for newer entities that have been added to AMBF. These include various types of sensors and soft-bodies. Sensors need a newer design of communication payloads that follow the same underlying design principles as the original afObject payloads did.

8.3.2 Extension of the Python Client

One of the most attractive features of AMBF is the complementary Python client which makes the process of controlling simulated bodies much easier as compared to any other simulator. This is primarily due to the design of communication payloads. As discussed in the previous point, a newer communication payload design would require a revamp of the Python client itself. The proposed idea is that the client will itself disseminate objects into corresponding class instances based on the data types so the end-user gets the exact seamless experience as they get from the current Python client.

8.3.3 Inclusion of More Sensors

The most important feature of any simulator is the ability to emulate environmental data using simulated sensors. AMBF inherently relays kinematic and dynamic data of robots, joints and free bodies using extensive communication payloads and thus does not require simulated sensors for these specific purposes. However, with the evolution of AMBF, various kinds of sensors have been added and there is room for much more. Sensors for depth mapping, range finding and inertial updates, although applied in one form or another in AMBF, are not yet available for general purpose use.

8.3.4 Support for Application Program Interface

There is a need to develop a plugin-based API where users can expand AMBF's capabilities without modifying the core framework. A primary candidate that can leverage an extensible API is the inclusion of various kinds of sensors.

8.3.5 Improved Softbody Simulation Framework

The inclusion of easy to use soft-body simulation is a distinguishing feature of AMBF. However, there is tremendous room for improvement in the specification, simulation and interaction of soft-bodies. At the moment, the elastic parameters are set as a whole for a single soft-body. Realistic soft-bodies and tissues are rarely homogenous. Furthermore, soft-body simulations are not as valuable if the underlying soft-bodies cannot be altered via manipulation. These alterations include cutting, stitching, sewing, etc. All these can still be achieved programmatically as applications in AMBF, but the point here is that some of these tasks can be simplified and specified as action-based sensors. The example of Grasping using Resistive sensors in an example of such task simplification.

8.3.6 Swappable Middleware

At the moment, ROS is used as the sole middleware for AMBF. This has several advantages, however, this is certainly not ideal. The future work for AMBF includes the abstraction of the middleware as well as the mode of communication to allow faster update speeds depending upon one's needs.

8.3.7 Incorporating CRTK Specification

The Collaborative Robotics Toolkit (CRTK) is a tremendous specification for interfacing with robots at various levels of control. This design of this toolkit happened at the same time as the development of AMBF. Thus, some of the earlier specifications were incorporated in various components of the AMBF, however, the inclusion of all the CRTK standards is left for the future.

8.4 Conclusion

The future of robot-assisted surgery may involve multiple surgeons coordinating to perform a single procedure and using collaborative control to train lesser experienced surgeons. The Asynchronous Framework is a small step in this direction as it lets users get trained via more experienced operators with haptic feedback in a shared environment with multiple contextual viewports. Other than its application for surgical robotics, the AMBF is a versatile simulator for general-purpose robots and rich environments. Several examples of its usage by the community have been presented in this manuscript. It is hoped that the development of AMBF will continue for many years to come both by the author and the community in general. AMBF is available publically at <https://github.com/WPI-AIM/ambf>. The readers of this manuscript are encouraged to try it out.

Bibliography

- [1] S. Staff, “Laparoscopic surgery.” <https://www.aboutkidshealth.ca/Article>, 2009.
- [2] Á. Takács, D. Nagy, I. Rudas, and T. Haidegger, “Origins of surgical robotics: From space to the operating room,” Acta Polytechnica Hungarica, vol. 13, pp. 13–30, 01 2016.
- [3] C. Junhong, D.-D. Zhang, A. Munawar, G. S. Fischer, and G. Z. Yan, “Supervised Semi-Autonomous Control for Surgical Robot Based on Bayesian Optimization,” In Review for International Conference on Robotics and Automation (ICRA), 2020.
- [4] P. P. Rao, “Robotic surgery: new robots and finally some real competition!,” World journal of urology, vol. 36, no. 4, pp. 537–541, 2018.
- [5] M. Salman, T. Bell, J. Martin, K. Bhuvra, R. Grim, and V. Ahuja, “Use, cost, complications, and mortality of robotic versus nonrobotic general surgery procedures based on a nationwide database,” The American surgeon, vol. 79, no. 6, pp. 553–560, 2013.
- [6] V. Patel, M. Chammas, and S. Shah, “Robotic assisted laparoscopic radical prostatectomy: a review of the current state of affairs,” International journal of clinical practice, vol. 61, no. 2, pp. 309–314, 2007.

- [7] J. Ruurda, T. J. Van Vroonhoven, and I. Broeders, “Robot-assisted surgical systems: a new era in laparoscopic surgery,” Annals of the Royal College of Surgeons of England, vol. 84, no. 4, p. 223, 2002.
- [8] B. Davies, R. Hibberd, M. Coptcoat, and J. Wickham, “A surgeon robot prostatectomy-a laboratory evaluation,” Journal of medical engineering & technology, vol. 13, no. 6, pp. 273–277, 1989.
- [9] M. Gagner, E. Begin, R. Hurteau, and A. Pomp, “Robotic interactive laparoscopic cholecystectomy,” The Lancet, vol. 343, no. 8897, pp. 596–597, 1994.
- [10] S. Bann, M. Khan, J. Hernandez, Y. Munz, K. Moorthy, V. Datta, T. Rockall, and A. Darzi, “Robotics in surgery,” Journal of the American College of Surgeons, vol. 196, no. 5, pp. 784–795, 2003.
- [11] J. A. Singh, “Epidemiology of knee and hip arthroplasty: a systematic review,” The open orthopaedics journal, vol. 5, p. 80, 2011.
- [12] T. Surgical, “History of ROBODOC.” <https://thinksurgical.com/company/history>, 2019.
- [13] “ZEUS Robotic Surgical System.” <http://allaboutroboticsurgery.com/zeusrobot.html>.
- [14] R. M. Satava, “Robotic surgery: from past to future—a personal journey,” Surgical Clinics of North America, vol. 83, no. 6, pp. 1491–1500, 2003.
- [15] R. M. Satava, “Surgical robotics: the early chronicles: a personal historical perspective,” Surgical Laparoscopy Endoscopy & Percutaneous Techniques, vol. 12, no. 1, pp. 6–16, 2002.

- [16] R. A. Beasley, “Medical robots: current systems and research directions,” Journal of Robotics, vol. 2012, 2012.
- [17] R. Schmitz, F. Willeke, M. Ibrahim Darwich, A. Surgeon, S. M. Kloeckner-Lang, H. Saelzer, J. Labenz, D. Klinkum, D.-P. Borkenstein, A. Physician, et al., “Robotic-Assisted Nissen Fundoplication with the Senhance® Surgical System: Technical Aspects and Early Results,” PubMed, 2019.
- [18] C. Newmarker, “Medtronic finally unveils its new robot-assisted surgery system.” <https://www.massdevice.com/medtronic-finally-unveils-its-new-robot-assisted-surgery-system/>, Sep 2019.
- [19] S. Crowe, “CMR Surgical raises \$240M for Versius surgical robot.” <https://www.therobotreport.com/cmr-surgical-243-million-versius-surgical-robot/>, Sep 2019.
- [20] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, and S. P. DiMaio, “An Open-Source Research Kit for the da Vinci Surgical System,” in IEEE Intl. Conf. on Robotics and Auto. (ICRA), (Hong Kong, China), pp. 6434–6439, 2014.
- [21] Z. Chen, A. Deguet, R. Taylor, S. DiMaio, G. Fischer, and P. Kazanzides, “An open-source hardware and software platform for telesurgical robotics research,” in Proceedings of the MICCAI Workshop on Systems and Architecture for Computer Assisted Interventions, Nagoya, Japan, vol. 2226, 2013.
- [22] D. Hershberger, D. Gossow, and J. Faust, “RViz, 3D visualization tool for ROS,” URL: <http://wiki.ros.org/rviz>, 2016.

- [23] A. Munawar, “Implementation of a Surgical Robot Dynamical Simulation and Motion Planning Framework,” Master’s thesis, Worcester Polytechnic Institute, 2015.
- [24] S. Chitta, I. Sucan, and S. Cousins, “Moveit![ros topics],” IEEE Robotics & Automation Magazine, vol. 19, no. 1, pp. 18–19, 2012.
- [25] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72–82, 2012.
- [26] A. Munawar, “A Matlab Toolkit for controlling the dVRK.” <https://github.com/WPI-AIM/dvrk-matlab-ros>, 2015.
- [27] Mathworks, “Using ROS Bridge to Establish Communication between ROS and ROS 2.” <https://www.mathworks.com/help/ros/ug/communicate-between-ros-and-ros-2-networks-using-ros-bridge.html>, 2014.
- [28] MATLAB, version (R2018a). Natick, Massachusetts: The MathWorks Inc., 2018.
- [29] A. Ruszkowski, Z. F. Quek, A. Okamura, and S. Salcudean, “Simulink to C++ interface for controller development on the da Vinci[®] Research Kit (dVRK),” ICRA Workshop on Shared Frameworks for Medical Robotics Research, May 2015.
- [30] Y.-H. Su, A. Munawar, A. Deguet, A. Lewis, K. Lindgren, Y. Li, R. H. Taylor, G. S. Fischer, B. Hannaford, and P. Kazanzides, “Collaborative Robotics Toolkit (CRTK): Open Software Framework for Surgical Robotics Research,” In Review, IRC 2020 : The Fourth IEEE International Conference on Robotic Computing, 2020.

- [31] P. Kazanzides, B. Hannaford, G. Fischer, R. H. Taylor, A. Deguet, A. Munawar, Y.-H. Su, A. Lewis, K. Lindgren, and Y. Li, “Collaborative Robotics Toolkit (CRTK).” <https://github.com/collaborative-robotics/collaborative-robotics.github.io>, 2019.
- [32] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), vol. 3, pp. 2149–2154, IEEE, 2004.
- [33] R. Smith et al., “Open dynamics engine,” Manual, 2005.
- [34] E. Coumans, “Bullet physics engine,” Open Source Software: <http://bulletphysics.org>, vol. 1, no. 3, p. 84, 2010.
- [35] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, “DART: Dynamic Animation and Robotics Toolkit.,” J. Open Source Software, vol. 3, no. 22, p. 500, 2018.
- [36] M. A. Sherman, A. Seth, and S. L. Delp, “Simbody: multibody dynamics for biomedical research,” Procedia Iutam, vol. 2, pp. 241–261, 2011.
- [37] X. S. Part, “2: XML Datatypes,” 2001.
- [38] T. P. Goodman, “A least-squares method for computing balance corrections,” Journal of Engineering for Industry, vol. 86, no. 3, pp. 273–277, 1964.
- [39] R. Gondokaryono, A. Agrawal, A. Munawar, C. Nycz, and G. Fischer, “An Approach to Modeling Closed-Loop Kinematic Chain Mechanisms, applied to Simulations of the da Vinci Surgical System,” Special Issue on Platforms for

Robotics Research - Acta Polytechnica Hungarica, vol. 16, pp. 29–48, Nov 2018.

- [40] A. Agrawal and R. Gondokaryono, “Accurate URDF and SDF models of Intuitive Surgical’s daVinci Research Kit (dVRK).” <https://github.com/charlespwd/project-title>, 2018.
- [41] G. R. Wang. Yan, “Dynamic model identification of the DVRK.” https://github.com/WPI-AIM/dvrk_dynamics_identification, 2019.
- [42] Y. Wang, R. Gondokaryono, A. Munawar, and G. S. Fischer, “A Convex Optimization-based Dynamic Model Identification Package for the da Vinci Research Kit,” IEEE Robotics and Automation Letters, pp. 1–3, 2019.
- [43] R. S. Hartenberg and J. Denavit, “A kinematic notation for lower pair mechanisms based on matrices,” Journal of applied mechanics, vol. 77, no. 2, pp. 215–221, 1955.
- [44] T. Foote, “tf: The transform library,” in 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp. 1–6, IEEE, 2013.
- [45] A. Shademan, R. Decker, J. Opfermann, S. Leonard, A. Krieger, and P. C. W. Kim, “Supervised autonomous robotic soft tissue surgery,” Science Translational Medicine, vol. 8, pp. 337ra64–337ra64, 05 2016.
- [46] K. Bumm, J. Wurm, J. Rachinger, T. Dannenmann, C. Bohr, R. Fahlbusch, H. Iro, and C. Nimsky, “An Automated Robotic Approach with Redundant Navigation for Minimal Invasive Extended Transsphenoidal Skull Base Surgery,” Minimally invasive neurosurgery : MIN, vol. 48, pp. 159–64, 07 2005.

- [47] S. Sen, A. Garg, D. V. Gealy, S. McKinley, Y. Jen, and K. Goldberg, “Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization,” in Robotics and Automation (ICRA), 2016 IEEE International Conference on, pp. 4178–4185, IEEE, 2016.
- [48] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg, “Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms,” in Robotics and Automation (ICRA), 2015 IEEE International Conference on, pp. 1202–1209, IEEE, 2015.
- [49] A. Krupa, J. Gangloff, M. de Mathelin, C. Doignon, G. Morel, L. Soler, J. Leroy, and J. Marescaux, “Autonomous retrieval and positioning of surgical instruments in robotized laparoscopic surgery using visual servoing and laser pointers,” in Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on, vol. 4, pp. 3769–3774, IEEE, 2002.
- [50] R. Bauernschmitt, E. U. Schirmbeck, A. Knoll, H. Mayer, I. Nagy, N. Wessel, S. Wildhirt, and R. Lange, “Towards robotic heart surgery: Introduction of autonomous procedures into an experimental surgical telemanipulator system,” The International Journal of Medical Robotics and Computer Assisted Surgery, vol. 1, no. 3, pp. 74–79, 2005.
- [51] K. Shamaei, Y. Che, A. Murali, S. Sen, S. Patil, K. Goldberg, and A. M. Okamura, “A paced shared-control teleoperated architecture for supervised automation of multilateral surgical tasks,” in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pp. 1434–1439, IEEE, 2015.

- [52] T. D. Nagy and T. Haidegger, “An Open-Source Framework for Surgical Sub-task Automation,” Robotics and Automation (ICRA) Workshops, 2018 IEEE International Conference on, 2018.
- [53] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni, “Sofa-an open source framework for medical simulation,” in MMVR 15-Medicine Meets Virtual Reality, vol. 125, pp. 13–18, IOP Press, 2007.
- [54] A. Boeing and T. Bräunl, “Evaluation of real-time physics simulation systems,” in Graphite, vol. 7, pp. 281–288, 2007.
- [55] D. Baraff, “Analytical methods for dynamic simulation of non-penetrating rigid bodies,” in ACM SIGGRAPH Computer Graphics, vol. 23, pp. 223–232, ACM, 1989.
- [56] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, “Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations,” Applied Numerical Mathematics, vol. 25, no. 2-3, pp. 151–167, 1997.
- [57] L. F. Shampine and C. W. Gear, “A user’s view of solving stiff ordinary differential equations,” SIAM review, vol. 21, no. 1, pp. 1–17, 1979.
- [58] P. Volino and N. Magnenat-Thalmann, “Comparing efficiency of integration methods for cloth simulation,” in Proceedings. Computer Graphics International 2001, pp. 265–272, IEEE, 2001.
- [59] D. Donnelly and E. Rogers, “Symplectic integrators: An introduction,” American Journal of Physics, vol. 73, no. 10, pp. 938–945, 2005.

- [60] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, “Position based dynamics,” Journal of Visual Communication and Image Representation, vol. 18, no. 2, pp. 109–118, 2007.
- [61] B. V. Mirtich, Impulse-based Dynamic Simulation of Rigid Body Systems. PhD thesis, University of California at Berkley, 1996.
- [62] G. E. Forsythe and P. Henrici, “The cyclic Jacobi method for computing the principal values of a complex matrix,” Transactions of the American Mathematical Society, vol. 94, no. 1, pp. 1–23, 1960.
- [63] M. Usui, H. Niki, and T. Kohno, “Adaptive Gauss-Seidel method for linear systems,” International Journal of Computer Mathematics, vol. 51, no. 1-2, pp. 119–125, 1994.
- [64] D. Baraff, “Linear-time dynamics using Lagrange multipliers,” in SIGGRAPH, vol. 96, pp. 137–146, Citeseer, 1996.
- [65] M. W. Spong and M. Vidyasagar, Robot dynamics and control. John Wiley & Sons, 2008.
- [66] R. Featherstone, “The calculation of robot dynamics using articulated-body inertias,” The International Journal of Robotics Research, vol. 2, no. 1, pp. 13–30, 1983.
- [67] A. Jain and G. Rodriguez, “Diagonalized Lagrangian robot dynamics,” IEEE Transactions on Robotics and Automation, vol. 11, no. 4, pp. 571–584, 1995.
- [68] M. L. Felis, “RBDL: an efficient rigid-body dynamics library using recursive algorithms,” Autonomous Robots, vol. 41, no. 2, pp. 495–511, 2017.

- [69] M. Kranzfelder, C. Staub, A. Fiolka, A. Schneider, S. Gillen, D. Wilhelm, H. Friess, A. Knoll, and H. Feussner, “Toward increased autonomy in the surgical or: needs, requests, and expectations,” Surgical Endoscopy, vol. 27, pp. 1681–1688, May 2013.
- [70] E. Rohmer, S. P. N. Singh, and M. Freese, “V-REP: A versatile and scalable robot simulation framework,” in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1321–1326, Nov 2013.
- [71] A. Deguet, R. Kumar, R. Taylor, and P. Kazanzides, “The cisst libraries for computer assisted intervention systems,” in MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions, Midas Journal, vol. 71, 2008.
- [72] F. Conti and et al., “The CHAI libraries,” in Proceedings of Eurohaptics 2003, (Dublin, Ireland), pp. 496–500, 2003.
- [73] C. Berglund, “GLFW-An OpenGL Framework,” 2006.
- [74] B. Community, “The Boost C++ Libraries.” <https://www.boost.org/>.
- [75] J. Beder, “YAML parser and emitter in C++.” <https://github.com/jbeder/yaml-cpp>.
- [76] K. Simonov, “Canonical source repository for PyYAML.” <https://github.com/yaml/pyyaml>.
- [77] C. et al., “Keras.” <https://github.com/keras-team/keras>, 2015.
- [78] M. Plappert, “Keras-RL.” <https://github.com/matthiasplappert/keras-rl>, 2016.
- [79] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” 2016.

- [80] D. Shreiner and The Khronos OpenGL ARB Working Group, OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1. Addison-Wesley Professional, 7th ed., 2009.
- [81] B. Mirtich, “Rigid body contact: Collision detection to force computation,” in Workshop on Contact Analysis and Simulation, IEEE Intl. Conference on Robotics and Automation, 1998.
- [82] A. C. Thompson and A. C. Thompson, Minkowski geometry. Cambridge University Press, 1996.
- [83] J. E. Colgate and J. M. Brown, “Factors affecting the z-width of a haptic display,” in Proceedings of the 1994 IEEE International Conference on Robotics and Automation, pp. 3205–3210, IEEE, 1994.
- [84] L. Dalessandro, D. Dice, M. Scott, N. Shavit, and M. Spear, “Transactional mutex locks,” in European Conference on Parallel Processing, pp. 2–13, Springer, 2010.
- [85] A. Munawar and G. Fischer, “Towards a haptic feedback framework for multi-DOF robotic laparoscopic surgery platforms,” in Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, pp. 1113–1118, IEEE, 2016.
- [86] A. Munawar, “A repository for davinci research kit at wpi.” <https://github.com/WPI-AIM/wpi-dvrk-ros>, 2013.
- [87] Z. Chen, A. Deguet, R. H. Taylor, and P. Kazanzides, “Software architecture of the da Vinci Research Kit,” in 2017 First IEEE International Conference on Robotic Computing (IRC), pp. 180–187, IEEE, 2017.

- [88] H. Lin, C. Vincent Hui, Y. Wang, A. Deguet, P. Kazanzides, and K. W. S. Au, “A Reliable Gravity Compensation Control Strategy for dVRK Robotic Arms With Nonlinear Disturbance Forces,” IEEE Robotics and Automation Letters, vol. 4, pp. 3892–3899, Oct 2019.
- [89] A. Gokhale and D. C. Schmidt, “Measuring the performance of communication middleware on high-speed networks,” in ACM SIGCOMM Computer Communication Review, vol. 26, pp. 306–317, ACM, 1996.
- [90] P. Hintjens, ZeroMQ: messaging for many applications. ” O’Reilly Media, Inc.”, 2013.
- [91] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in ICRA workshop on open source software, vol. 3, p. 5, Kobe, Japan, 2009.
- [92] A. B. Dirk Thomas, Dorian Scholz, “ROS RQT.” <https://github.com/ros-visualization/rqt,>.
- [93] D. Faconti, “The timeseries visualization tool that you deserve.” <https://github.com/facontidavide/PlotJuggler>.
- [94] D. Thomas, T. Field, J. Leibs, and J. Bowman, “Rosbag-ROS Wiki.,” URL: <http://wiki.ros.org/rosbag>, 2014.
- [95] R. B. R. Tully Foote, “ROS Nodelet.” <http://wiki.ros.org/nodelet>, 2017.
- [96] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, “Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo,” arXiv preprint arXiv:1608.05742, 2016.

- [97] T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchallick, “Solving semi-Markov decision problems using average reward reinforcement learning,” Management Science, vol. 45, no. 4, pp. 560–574, 1999.
- [98] R. Hess, Blender Foundations: The Essential Guide to Learning Blender 2.6. Focal Press, 2010.
- [99] S. Cousins, “Welcome to ros topics [ros topics],” IEEE Robotics & Automation Magazine, vol. 17, no. 1, pp. 13–14, 2010.
- [100] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033, Oct 2012.
- [101] G. Sincarsin, G. D’Eleuterio, and P. Hughes, “Dynamics of an elastic multi-body chain: part Dmodelling of joints,” Dynamics and Stability of Systems, vol. 8, no. 2, pp. 127–146, 1993.
- [102] W. R. Saunders, D. G. Cole, and H. H. Robertshaw, “Impact of piezoelectric sensoriactuators on active structural acoustic control,” in Smart Structures and Materials 1993: Smart Structures and Intelligent Systems, vol. 1917, pp. 578–586, International Society for Optics and Photonics, 1993.
- [103] A. Munawar, “A versatile Universal Robot Description Format (URDF) to Asynchronous Multi-Body Framework (AMBF) Format Converter.” https://github.com/WPI-AIM/urdf_2_ambf, 2019.
- [104] S. Brawner, “SolidWorks to URDF Exporter.” https://github.com/ros/solidworks_urdf_exporter.

- [105] A. Munawar, “A Graphical add-on for Blender for Creating and Loading AMBF Models.” https://github.com/WPI-AIM/ambf_addon, 2019.
- [106] C. J. Nycz, R. Gondokaryono, P. Carvalho, N. Patel, M. Wartenberg, J. G. Pilitsis, and G. S. Fischer, “Mechanical validation of an MRI compatible stereotactic neurosurgery robot in preparation for pre-clinical trials,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1677–1684, Sep. 2017.
- [107] C. Bove, “Constrained Motion Planning System for MRI-Guided, Needle-Based, Robotic Interventions,” Master’s thesis, Worcester Polytechnic Institute, 2018.
- [108] A. Maciel, T. Halic, Z. Lu, L. P. Nedel, and S. De, “Using the PhysX engine for physics-based virtual surgery with force feedback,” The International Journal of Medical Robotics and Computer Assisted Surgery, vol. 5, no. 3, p. 341353, 2009.
- [109] E. Daneviius, R. Maskelinas, R. Damaeviius, D. Poap, and M. Woniak, “A Soft Body Physics Simulator with Computational Offloading to the Cloud,” Information, vol. 9, p. 318, Nov 2018.
- [110] J. Tan, G. Turk, and C. K. Liu, “Soft body locomotion,” ACM Transactions on Graphics, vol. 31, p. 111, Jan 2012.
- [111] J. Mesit and R. K. Guha, “Simulation of Soft Bodies with Pressure Force and the Implicit Method,” First Asia International Conference on Modelling and Simulation (AMS07), 2007.

- [112] J. Mesit, MODELING AND SIMULATION OF SOFT BODIES. PhD thesis, Department of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida, 2010.
- [113] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, and M. Macklin, “A survey on position-based simulation methods in computer graphics,” in Computer graphics forum, vol. 33, pp. 228–251, Wiley Online Library, 2014.
- [114] B. Mirtich and J. Canny, “Impulse-based simulation of rigid bodies,” in Proceedings of the 1995 symposium on Interactive 3D graphics, pp. 181–ff, ACM, 1995.
- [115] R. Tonge, L. Zhang, and D. Sequeira, “Method and program solving LCPs for rigid body dynamics,” July 18 2006. US Patent 7,079,145.
- [116] L. Velho, “A dynamic adaptive mesh library based on stellar operators,” Journal of Graphics Tools, vol. 9, no. 2, pp. 21–47, 2004.
- [117] A. Munawar, “The Asynchronous Multi-Body Framework.” <https://github.com/WPI-AIM/ambf>,, 2019.
- [118] Y. Duan, W. Huang, H. Chang, W. Chen, J. Zhou, S. K. Teo, Y. Su, C. K. Chui, and S. Chang, “Volume preserved mass-spring model with novel constraints for soft tissue deformation,” IEEE journal of biomedical and health informatics, vol. 20, no. 1, pp. 268–280, 2014.
- [119] M. Matyka and M. Ollila, “Pressure model of soft body simulation,” in The Annual SIGRAD Conference. Special Theme-Real-Time Simulations. Conference Proceedings from SIGRAD2003, pp. 29–33, Linköping University Electronic Press, 2003.

- [120] H. Si, “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator,” ACM Trans. Math. Softw., vol. 41, pp. 11:1–11:36, Feb. 2015.
- [121] G. A. Fontanelli, M. Selvaggio, M. Ferro, F. Ficuciello, M. Vendittelli, and B. Siciliano, “A v-rep simulator for the da Vinci research kit robotic platform,” in 2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob), pp. 1056–1061, IEEE, 2018.
- [122] K. Yamane and Y. Nakamura, “Stable penalty-based model of frictional contacts,” in Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pp. 1904–1909, IEEE, 2006.
- [123] D. C. Ruspini and O. Khatib, “Collision/contact models for dynamic simulation and haptic interaction,” in Robotics Research, pp. 185–194, Springer, 2000.
- [124] A. Miller, P. Allen, V. Santos, and F. ValeroCuevas, “From robotic hands to human hands: a visualization and simulation engine for grasping research,” Industrial Robot: An International Journal, vol. 32, no. 1, p. 5563, 2005.
- [125] B. Len, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moio, J. Bohg, J. Kuffner, and et al., “OpenGRASP: A Toolkit for Robot Grasping Simulation,” Simulation, Modeling, and Programming for Autonomous Robots Lecture Notes in Computer Science, p. 109120, 2010.
- [126] S. Moio, B. Len, P. Korkealaakso, and A. Morales, “Model of tactile sensors using soft contacts and its application in robot grasping simulation,” Robotics and Autonomous Systems, vol. 61, no. 1, p. 112, 2013.
- [127] M. Ciocarlie, C. Lackner, and P. Allen, “Soft Finger Model with Adaptive Contact Geometry for Grasping and Manipulation Tasks,” Second Joint

EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC07), 2007.

- [128] C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelosof, “Grasp Planning via Decomposition Trees,” Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007.
- [129] E. W. Hawkes, D. L. Christensen, A. K. Han, H. Jiang, and M. R. Cutkosky, “Grasping without squeezing: Shear adhesion gripper with fibrillar thin film,” 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015.
- [130] E. Todorov, “Implicit nonlinear complementarity: A new approach to contact dynamics,” 2010 IEEE International Conference on Robotics and Automation, 2010.
- [131] E. Todorov, “Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo,” 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014.
- [132] M. Malvezzi, G. Gioioso, G. Salvietti, D. Prattichizzo, and A. Bicchi, “Syn-Grasp: A MATLAB toolbox for grasp analysis of human and robotic hands,” 2013 IEEE International Conference on Robotics and Automation, 2013.
- [133] A. J. Spiers, B. Calli, and A. M. Dollar, “Variable-Friction Finger Surfaces to Enable Within-Hand Manipulation via Gripping and Sliding,” IEEE Robotics and Automation Letters, vol. 3, no. 4, p. 41164123, 2018.
- [134] M. Sagardia, T. Stouraitis, and J. L. e Silva, “A new fast and robust collision detection and force computation algorithm applied to the physics engine bullet: Method, integration, and evaluation,” in Prof. of the Conf. and Exhibition

- of the European Association of Virtual and Augmented Reality (EuroVR), pp. 65–76, 2014.
- [135] B. Chang and J. E. Colgate, “Real-time impulse-based simulation of rigid body systems for haptic display,” in Proc. Symp. on Interactive 3D Graphics, pp. 200–209, 1997.
- [136] R. Bridson, R. Fedkiw, and J. Anderson, “Robust treatment of collisions, contact and friction for cloth animation,” in ACM Transactions on Graphics (ToG), vol. 21, pp. 594–603, ACM, 2002.
- [137] R. L. Cook, T. Porter, and L. Carpenter, “Distributed ray tracing,” in ACM SIGGRAPH computer graphics, vol. 18, pp. 137–145, ACM, 1984.
- [138] Y. M. Govaerts and M. M. Verstraete, “Raytran: A Monte Carlo ray-tracing model to compute light scattering in three-dimensional heterogeneous media,” IEEE Transactions on geoscience and remote sensing, vol. 36, no. 2, pp. 493–505, 1998.
- [139] A. Munawar, “Interact with AMBF Simulator using ROS Based Geomagic Devices.” https://github.com/adnanmunawar/geomagic_utils, August 2019.
- [140] A. Munawar, “AMBF User Study GUI.” https://github.com/WPI-AIM/ambf/blob/user-study/ambf_ros_modules/ambf_comm/scripts/tests/user_study.py, 2019.
- [141] T. Field, J. Leibs, and J. Bowman, “ROS Bag.” <http://wiki.ros.org/rosbag>.

- [142] S. G. Hart and L. E. Staveland, “Development of nasa-tlx (task load index): Results of empirical and theoretical research,” in Advances in psychology, vol. 52, pp. 139–183, Elsevier, 1988.
- [143] F. Conti, F. Barbagli, D. Morris, and C. Sewell, “Chai 3d: An open-source library for the rapid development of haptic scenes,” IEEE World Haptics, pp. 21–29, 2005.
- [144] E. Coumans, “Bullet Physics Simulation,” in ACM SIGGRAPH 2015 Courses, SIGGRAPH ’15, (New York, NY, USA), ACM, 2015.