



HOKKAIDO UNIVERSITY

Title	Control and Identification Framework for Modularized Robot Platforms
Author(s)	Sobehart, Lionel Matias
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第13205号
Issue Date	2018-03-22
DOI	https://doi.org/10.14943/doctoral.k13205
Doc URL	https://hdl.handle.net/2115/69988
Type	doctoral thesis
File Information	Lionel_Matias_Sobehart.pdf



Control and Identification Framework for Modularized Robot Platforms



Lionel M. Sobehart

Graduate School of Engineering
Hokkaido University

This dissertation is submitted for the degree of
Doctor of Philosophy

Abstract

Recent advances in compact computers and 3D scanning allow high fidelity simulations to be calculated directly on the robot, in real time, to assist in the operation of the robot itself. These simulations can be used to provide soft-sensing of the robot state and environment without the need of additional hardware sensors. In particular, this concept can be applied to modularized robots where sensory information is limited. Modularization of the robot hardware and software minimizes manufacturing and maintenance costs typically associated with highly articulated robots, especially if the number of sensors per module are reduced to the minimum number necessary for control of the servomotors in the modules. This thesis will address aspects of the design, control, and simulation of a minimal sensing modularized robot with optimization aided soft-sensing.

Quadruped robots, one type of modularizable robots, have long been studied for their potential use in exploration and payload transfer in dangerous situations on uneven terrain where wheeled options are not a viable option. Due to the high power required to achieve legged motion with payloads, many prior designs have used compact hydraulics and intricate joints to improve performance. The proposed platform instead uses digitally controlled servo-motors with self-contained sensing and control are used as the base components of actuation modules. When connected by a universal frame design joining individual modules, overall costs associated with the manufacture and maintenance of the robot can be minimized while allowing for relatively high overall performance. A simplified robot comprising of a single actuator module and its connecting frames was designed to calibrate simulations and validate the state estimation methods.

The hardware is controlled at a low level by means of a custom high-performance control framework that includes a highly-integrated simulation component and abstracted control. This framework was developed to work in conjunction with the generalized nature of modularized robots and allow for simplified respecification of core components without significant changes to higher level control schemes. The framework offloads responsibility for joint dynamics in simulation from the simulation architecture to derivatives of a core actuator module class through the use of language level callbacks. Communications devices are likewise abstracted to allow for respecification. In order to ensure optimal performance across varying communication protocol types, all communications are separated into unidirectional and bidirectional signal types, which are interleaved asynchronously from the robot command update rate according to the communications device specification. Communications optimizations can be implemented by overloading certain functions within the framework to allow for master-slave relationships that can minimize latency in the hardware used. Interleaved with these components is a high fidelity, faster than real-time simulation of the robot state within a known environment.

Additional components of the framework were implemented to assist the use of simulation and experimental data for robot state estimation. Multiple non-linear and meta-heuristic optimization methods were developed to improve automation of system identification and develop a basis for alternative soft-sensing methods. A multithread optimized data logger and comparison tool was

developed to improve the performance of parallel simulations used within system identification and optimization based sensing, with reduced overhead compared to a message passing approach that is typically seen in robotic applications. Finally, a synchronized system stepper and portable timing method were designed to operate robot platforms both in real-time and during online and offline optimization.

Simulation of actuators is improved by modeling the internal controller and dynamics of actuator modules using known and estimated module state information, allowing an abstraction of the actuator specifics from the core simulation and control code. Although electro-mechanical motor models have been used to simulate systems directly when the module components are known, this requires full knowledge of the system and small steps during discrete time simulation to prevent instabilities in the simulation and control. A rigid body constraint based model for the simulation of internally controlled articulated robot modules is proposed which can reduce simulation instabilities at larger time-steps without requiring simulation specific non-physical damping or closed form dynamic solutions. This method uses the properties of rigid body constraints to clamp system dynamics when low system inertia or high control gains would otherwise result in physically impossible performance. The accuracy of this method is comparable to traditional models, but an order of magnitude faster in practice due to larger time-steps, improving the overall performance of simulations significantly.

System state information for feedback control and pose planning in robotic systems often requires multiple torque sensors that can increase the overall system complexity and cost. Using available module internal sensor data and rigid body simulation based observers, the proposed methods can minimize the number of hardware sensors required to estimate unknowns in the robot state. In the case that the state unknowns significantly impact the system dynamics, traditional joint torque observer methods may show degraded performance due to un-modeled nonlinearities. A real-time soft sensing methodology using optimization techniques and comprehensive simulation models is proposed to improve estimation of robot state using minimal sensors. This method uses parallel simulations to test a wide variety of possible values for system unknowns using existing sensory data for comparison.

Table of Contents

ABSTRACT	II
TABLE OF CONTENTS	IV
TABLE OF FIGURES	VII
NOMENCLATURE	XI
1 INTRODUCTION	1
1.1 MOTIVATIONS	1
1.2 CONTRIBUTIONS	2
1.3 OUTLINE	2
2 STATE OF THE ART	4
2.1 QUADRUPEL ROBOTICS ELECTROMECHANICAL HARDWARE DESIGN	4
2.1.1 <i>Sony Aibo</i>	4
2.1.2 <i>Boston Dynamics Quadruped Line</i>	5
2.1.3 <i>HyQ</i>	8
2.2 ROBOT POSE	9
2.2.1 <i>Kinematic control and inverse-kinematics</i>	9
2.2.2 <i>Scripted Gaits</i>	10
2.3 SIMULATION AND ESTIMATION IN ROBOTICS	12
2.3.1 <i>Pure Rigid Body Method</i>	14
2.3.2 <i>Soft Body Method</i>	15
2.3.3 <i>Particle Fluid Method</i>	16
2.3.4 <i>Hybrid Methods</i>	17
3 HARDWARE AND COMMUNICATIONS	18
3.1 INTRODUCTION	18
3.2 REQUIREMENTS AND GOALS	18
3.3 MODULARIZED ROBOTICS	20
3.4 QUADRUPEL SHAPE DESIGN	22
3.4.1 <i>System Simplification</i>	23
3.4.2 <i>Leg configuration</i>	24
3.4.3 <i>Customizable platform design</i>	27
3.4.4 <i>Extendable foot</i>	29
3.4.5 <i>Kinematic considerations</i>	30
3.5 MOTOR MODULES	34
3.6 ADDITIONAL COMPONENTS	38
3.6.1 <i>Sensor and power tray</i>	38
3.6.2 <i>Power supply</i>	39
3.7 ALTERNATIVE DESIGNS AND TEST PLATFORMS	40
3.7.1 <i>SARM- the Single Actuator Response Mechanism</i>	40
3.7.2 <i>Alternative module specification designs</i>	41
3.8 SUMMARY	42

4	ROBOT CONTROL FRAMEWORK	43
4.1	INTRODUCTION	43
4.2	MODULE INTERCONNECTIVITY	44
4.3	MOTOR MODULES	47
4.3.1	<i>Physical Motors</i>	48
4.4	SENSORS	49
4.5	COMMUNICATIONS MODULE	50
4.5.1	<i>Packet based communication oriented</i>	50
4.5.2	<i>Latency Considerations for Serial Communications</i>	51
4.5.3	<i>Asynchronous Communications</i>	53
4.5.4	<i>Passive communications optimizations</i>	55
4.6	SIMULATION MODULE	59
4.7	ADDITIONAL TOOLS AND UTILITIES	60
4.7.1	<i>Robot state recording</i>	60
4.7.2	<i>Multithreaded cache based comparison</i>	61
4.7.3	<i>Optimization</i>	63
4.7.4	<i>Interpolation</i>	67
4.7.5	<i>Extensible Synchronized Clock</i>	68
4.7.6	<i>Discrete System Stepper</i>	68
4.8	APPLICATION TO QUADRUPEL ROBOT AND SARM HARDWARE	69
4.8.1	<i>FTDI FT232R considerations</i>	69
4.8.2	<i>Dynamixel 1.0 protocol packet structure and optimizations</i>	69
4.9	SUMMARY	71
5	PHYSICAL SIMULATION	73
5.1	GOALS AND REQUIREMENTS	73
5.2	DISCRETE-TIME RIGID BODY SIMULATION ENGINE	74
5.3	CONSTRAINTS MODIFICATIONS	75
5.3.1	<i>Rigid body interactions via Constraints</i>	75
5.3.2	<i>Constraint error allowance and correction with CFM and ERP</i>	77
5.3.3	<i>Motorized Constraint Control Modes</i>	78
5.3.4	<i>Motor Model Hooks</i>	79
5.4	PERFORMANCE CONSIDERATIONS	80
5.4.1	<i>Collision Body Simplification</i>	80
5.5	DISCRETE-TIME SIMULATION OF MODULAR ROBOTS USING RIGID BODY METHODS	83
5.5.1	<i>Stability considerations</i>	84
5.6	REAL-TIME VISUALIZATION CAPABILITY	86
5.7	FASTER THAN REAL TIME METHODS	88
5.8	SUMMARY	88
6	MOTOR RESPONSE SIMULATION	90
6.1	INTRODUCTION	90
6.2	SIMPLIFIED MOTOR APPROXIMATIONS	90
6.2.1	<i>Kinematic Approximations</i>	90
6.2.2	<i>Saturation limit control</i>	91
6.3	ELECTRO-MECHANICAL TORQUE SUB-SIMULATION	92

6.4	CONDITIONAL CONSTRAINT METHOD	95
6.4.1	<i>Constraint based stability improvement</i>	95
6.4.2	<i>Conditional Constraint Method</i>	95
6.4.3	<i>Simplified controller for optimization</i>	97
6.4.4	<i>Conditional Constraint Experimental Validation</i>	99
6.5	SIMULATION ACCURACY	100
6.6	SIMULATION PERFORMANCE	105
6.7	SUMMARY.....	107
7	MODEL IDENTIFICATION AND SOFT SENSING	109
7.1	INTRODUCTION	109
7.2	MOTOR PARAMETER IDENTIFICATION BY PARTICLE SWARM OPTIMIZATION	111
7.2.1	<i>Application to SARM platform</i>	112
7.3	REAL-TIME SOFT SENSING	115
7.4	OPTIMIZATION BASED SOFT-SENSING	116
7.5	VIRTUAL SENSORS.....	121
7.6	SUMMARY.....	121
8	CONCLUSIONS.....	122
8.1	FUTURE WORKS.....	123
	REFERENCES	124
	ACKNOWLEDGEMENTS	130
	APPENDIX A. HARDWARE MODELS AND SPECIFICATIONS.....	131
	APPENDIX B. CONTROL FRAMEWORK IMPLEMENTATION	135
	PUBLICATIONS.....	139

Table of Figures

FIGURE 2.1: SECOND GENERATION SONY AIBO WITH INTEGRATED CAMERA AND WI-FI CONNECTION [2]. FEW PLATFORMS HAVE HAD AS MUCH WIDESPREAD USE IN ROBOTICS RESEARCH AS THE AIBO LINE.	5
FIGURE 2.2: BOSTON DYNAMICS LITLEDOG RESEARCH PLATFORM WALKING OVER AN OBSTACLE. MOTION CAPTURE AND WI-FI CONNECTIVITY ACCESSORIES HAVE BEEN ADDED TO ENABLE REMOTE VISUAL SERVOING [4].	6
FIGURE 2.3: PUBLIC PRESENTATION OF THE BIGDOG, AN ADVANCED QUADRUPEL ROBOT FOR MILITARY APPLICATIONS [5].	7
FIGURE 2.4: A REFINED PROTOTYPE OF THE LS3 PLATFORM, THE JOINT DEFINITIONS WERE SIGNIFICANTLY CHANGED FROM THE BIGDOG DESIGN. ADDITIONAL EXTERNAL PAYLOAD WAS ADDED DURING DEMONSTRATIONS TO SIMULATE IN-FIELD USE [5].	7
FIGURE 2.5: THIRD GENERATION HYQ PLATFORM WITH REDUCED COMPLEXITY COMPARED TO THE ORIGINAL PLATFORM [7].	8
FIGURE 2.6: MULTIPLE JOINT SPACE SOLUTIONS FOR A REDUNDANT INVERSE KINEMATIC END-EFFECTOR POINT. THE GROUND PLANE CONTACT POINT IS IDENTICAL DESPITE SIGNIFICANTLY DIFFERENT DYNAMICS CONDITIONS.	10
FIGURE 2.7: KINEMATIC LEG CONTROL KEY POINTS AND INTERPOLATED COMPOSITE BÉZIER CURVES FOR INTERMEDIARY JOINT ANGLES.	11
FIGURE 2.8: QUADRUPEL ROBOT GAIT PROJECTION, WITH THE FIRST CYCLE IN RED AND SECOND CYCLE IN BLUE. THE GAIT TYPE IS DETERMINED BY THE PHASE AND GROUND CONTACT DUTY CYCLE OF LEG MOTIONS RATHER THAN THE TRAJECTORY OF THE LEG IN THE AIR.	12
FIGURE 2.9: COLLISION RESPONSE COMPARISON BETWEEN RIGID (A), SOFT (B), AND PARTICLE BODY REPRESENTATIONS OF A RIGID BODY WITH SHARP EDGES IMPACTING A SECOND BODY. THE DEFLECTION AMOUNTS ARE EXAGGERATED BEYOND WHAT WOULD NORMALLY BE SEEN IN A ROBOTIC SYSTEM FOR THE PURPOSES OF ILLUSTRATION.	14
FIGURE 3.1: EXAMPLES OF MODULARIZED ROBOTS IN VARIOUS CONFIGURATIONS. ALTHOUGH MOTOR COMPONENTS ARE IDENTICAL, THE RESULTANT SHAPES AND GAITS ARE SIGNIFICANTLY DIFFERENT.	20
FIGURE 3.2: MODULARIZATION PROCESS FLOWCHART FOR A TYPICAL ARTICULATED ROBOT. A TYPICAL DESIGN WILL GO THROUGH SEVERAL ITERATIONS TO PRODUCE A VIABLE HARDWARE PLATFORM.	22
FIGURE 3.3: TWO MAIN QUADRUPEL CONFIGURATIONS, LIZARD-LIKE ON THE LEFT AND MAMMAL-LIKE ON THE RIGHT. THE WIDER BASE OF THE LIZARD-LIKE CONFIGURATION PROVIDES ADDITIONAL STABILITY OVER THE MAMMAL-LIKE CONFIGURATION.	23
FIGURE 3.4: APPROXIMATE DOG JOINT LOCATION IN 2D PLANE. THE ACTUAL JOINT DEFINITION FOR THE SHOULDER AND HIP CANNOT BE GIVEN ON A STRICTLY 2D REPRESENTATION	24
FIGURE 3.5: DETAILED 2D JOINT APPROXIMATION (A) WITH THREE ROTATION JOINTS PER LEG AND FLEXIBLE SPINE (DOUBLE YELLOW LINE) AND SIMPLIFIED VERSION (B) WITH TWO ROTATION JOINTS AND RIGID SPINE (SOLID YELLOW LINE). BOTH KINEMATIC AND DYNAMIC PROPERTIES OF EACH JOINT ARE DIFFERENT, BUT OVERALL SYSTEM DYNAMICS IS COMPARABLE.	24
FIGURE 3.6: PROPOSED LEG CONFIGURATIONS IN A NON-SINGULAR POSE.	26
FIGURE 3.7: COMPLETED ROBOT IN MINIMAL PAYLOAD STATE (LEFT) AND THE CAD MODEL WITH ADDITIONAL COMPUTING UNIT (RIGHT). THE PLATFORM CAN BE RECONFIGURED TO SUPPORT A NUC COMPUTER AND RGBD SENSOR (NOT SHOWN) FOR SEMI-AUTONOMOUS SLAM ASSISTED LOCOMOTION	28
FIGURE 3.8: LEG END EFFECTOR MODULE WITH ALTERNATIVE LEG LENGTHS. LEG EXTENDERS USE STANDARDIZED CONNECTIONS AND CAN BE REPLACED WITHOUT ADDITIONAL TOOLS.	29
FIGURE 3.9: TOOL-LESS LEG EXTENSION COMPONENT WITH OPTIONAL REPLACEMENT PARTS. LEG LENGTH CAN BE MODIFIED WITHOUT ADDITIONAL TOOLS.	30
FIGURE 3.10: INDIVIDUAL LEG WITH CENTER OF GRAVITY AND JOINT AXIS DEFINITIONS. THE JOINT AXIS DEFINITIONS ARE OFFSET FROM THE CENTER OF GRAVITY TO SIMPLIFY THE KINEMATIC MODEL CALCULATIONS. JOINTS ARE GIVEN WITH LOCAL X AXIS IN RED, Y AXIS IN GREEN, AND Z AXIS IN BLUE. THE CENTER OF GRAVITY IS GIVEN IN A LIGHTER SHADE WHERE IT DOES NOT MATCH THE JOINT LOCATION.	31
FIGURE 3.11: FRONT SIDE VIEW OF THE APPROXIMATE WORKSPACE OF THE QUADRUPEL ROBOT WITH MEDIUM LENGTH LEG EXTENDERS INSTALLED. ACTUAL WORKSPACE IN THIS IMPLEMENTATION IS SMALLER DUE TO INTERNAL COLLISIONS WITH THE MAIN BODY. ...	32

FIGURE 3.12: RIGHT SIDE VIEW OF THE APPROXIMATE WORKSPACE OF THE QUADRUPEL ROBOT WITH MEDIUM LENGTH LEG EXTENDERS INSTALLED.	33
FIGURE 3.13: TOP VIEW OF THE QUADRUPEL ROBOT, INCLUDING THE POWER CONNECTION PLATE, PROTOTYPING BOARD FOR LOW LEVEL SENSORS, AND OPTIONAL HIGH-POWERED MINIATURE COMPUTER. NECESSARY CABLES ARE NOT INCLUDED IN THE DIAGRAM... ..	33
FIGURE 3.14: MODULE SERVO MOTOR COMPONENT VIEW SHOWS THE MAIN COMPONENTS FOR DRIVING AND CONTROLLING THE INTERNAL DC MOTOR. THE POSITION SENSOR IS ATTACHED TO THE OUTPUT SHAFT AND CANNOT BE SHOWN WITHOUT IRREVERSIBLE DAMAGE TO THE FUNCTION OF THE MOTOR.....	36
FIGURE 3.15: DRIVING ELECTRICAL LOAD (PWM) DURING THE FIRST 100MS OF A POSITIVE STEP INPUT AS A FUNCTION OF MOTOR VELOCITY FOR AN MX-28T SMART SERVO MOTOR. VALUES RANGE FROM 0 (NO POWER TO MOTORS) TO 1 (100% POWER SUPPLIED)	37
FIGURE 3.16: SERVO MOTOR DYNAMIC RESPONSE TO AN 4π Hz $\pm\pi$ 4 SINE WAVE UNDER VARIOUS LOADINGS. MAXIMUM MOTOR SPEED ORDER ON THE UP-STROKE (LEFT) IS REVERSED DURING THE DOWN-STROKE (RIGHT) DUE TO EXTERNAL FORCES FROM GRAVITY.....	38
FIGURE 3.17: REINFORCED POWER AND SENSOR TRAY. THE “BACK” OF THE TRAY IS SHOWN ON THE BOTTOM LEFT WITH THE “FRONT” IN THE TOP RIGHT OF THE IMAGE. ADDITIONAL CLASP OPTIONS FOR A COVER WERE NOT USED IN THE INITIAL PRINT.....	39
FIGURE 3.18: EXTERNAL POWER SUPPLY FOR THE ROBOT PLATFORM WITH WEATHER RESISTANT ROBOT POWER CONNECTOR.	40
FIGURE 3.19: SIMPLIFIED SCHEMATIC OF THE TEST APPARATUS. THE SARM TEST PLATFORM MINIMIZES ERRORS DUE TO INERTIA APPROXIMATIONS AND SUPPORT FRAME DEFLECTIONS.....	41
FIGURE 3.20: RENDERED SIMPLIFIED CAD REPRESENTATION OF THE SARM TEST APPARATUS IN THE EXPERIMENTAL ORIENTATION. LOAD MASS, CLAMPING MECHANISM, AND MOTOR CONTROL CABLE ARE NOT SHOWN.	41
FIGURE 4.1: THE CONTROL FRAMEWORK CONNECTIVITY DIAGRAM IN A TYPICAL ROBOT CONTROL CASE. THE GENERAL PURPOSE BEHIND EACH MODULE IS COLOR CODED INTO FOUR OVERARCHING CATEGORIES.	45
FIGURE 4.2: MAJOR CONTROL FRAMEWORK MODULES AND THEIR COMPONENTS. CLASSES ARE CENTERED ABOUT THE ROBOTSERVO MOTOR AND ROBOT COMMUNICATIONS SUPER-CLASSES.	46
FIGURE 4.3: CLASS DIAGRAM OF PUBLICLY AVAILABLE FUNCTIONS AND MEMBERS FOR THE ROBOTSERVO MOTOR (TTLSERVO MOTOR) SUPER-CLASS. RETURN TYPES ARE GIVEN, WITH FLOAT TYPE AND BTSCALAR BEING FUNCTIONALLY IDENTICAL.....	48
FIGURE 4.4: DIGITAL PACKET TYPES OFTEN USED IN INDUSTRIAL ROBOTICS, BiSS (A) AND ModBus (B), WITH RELATIVE DATA SIZES FOR EACH COMPONENT. BOTH PACKETS INCLUDE A DISTINCT START VALUE S, A TARGET ADDRESS ADD., AND A CRC IN ADDITION TO DATA.	51
FIGURE 4.5: LOW COMMUNICATIONS BANDWIDTH (<100Hz) RESULTS FROM A BOUND GAIT ON THE QUADRUPEL ROBOT. THE LATENCY COMPENSATED TARGET (-45ms) AND SMOOTHED EXPERIMENTAL DATA (50ms SMOOTHING) HAVE MINIMAL RELATIVE ERROR, HOWEVER THE RAW EXPERIMENTAL POSITION ERROR STANDARD DEVIATION IS LARGE.	52
FIGURE 4.6: BUFFERED COMMUNICATION METHOD FOR PACKET BASED COMMUNICATIONS FOR A ROBOT. A WORST CASE SCENARIO IS SHOWN, WHERE COMMAND ORDER REQUIRES AN ADDITIONAL COMMUNICATIONS FRAME.	56
FIGURE 4.7: ASYNCHRONOUS COMMAND QUEUE SYSTEM FOR HARDWARE COMMUNICATIONS. THREE CALLBACK QUEUES ARE USED TO PROVIDE COMMUNICATIONS HARDWARE INDEPENDENT OPTIMIZATIONS	58
FIGURE 4.8: OBJECT DEPENDENCY IN THE FRAMEWORK AND APPROXIMATE DATA FLOW. CLASSES SHARED BETWEEN MAJOR COMPONENTS STRADDLE BOTH GROUPS, WHILE CLASSES ONLY REFERENCED BETWEEN COMPONENTS ARE CONNECTED BY ORANGE LINES. THE GREEN LINES REPRESENT INTERNAL CONNECTIONS AND DATA FLOW BETWEEN MORE COMPLEX SUB-CLASSES WITHIN THE MAIN COMPONENT.	60
FIGURE 4.9: ROBOT STATE RECORDING MODULE AND RECORDED STATES API SURFACE. DATA STORAGE AND INTERPOLATION BETWEEN POINTS ARE INTEGRAL TO THE ROBOT PLATFORM RECORDER CLASS.	61
FIGURE 4.10: A THREAD SAFE CACHED DATA COMPARISON FILTER FOR CALCULATING ROBOT FITNESS. READ OPERATIONS SHARE A MULTIUSER MUTEX WHILE COMMANDS THAT CHANGE SHARED DATA EXCLUSIVELY LOCK THE MUTEX TO PREVENT UNSAFE MEMORY ACCESS.	62
FIGURE 4.11: OPTIMIZATION TOOLSET FOR THE CONTROL FRAMEWORK. THE SAMPLING FUNCTION CAN BE ANY SUBSECTION OF THE FRAMEWORK INCLUDING A FULL ROBOT SIMULATION OR HARDWARE ROBOT.	65

FIGURE 4.12: COMPARISON OF THE STANDARD GOLDEN RATIO NON-LINEAR FUNCTION SEARCH (A) AND SAMPLED SPLINE MINIMIZATION METHOD (B). ALTHOUGH SAMPLING USES MORE CPU TIME, THE OVERALL EXECUTION TIME IS GREATLY DECREASED.	67
FIGURE 4.13: GENERIC DYNAMIXEL COMMUNICATIONS PACKET STRUCTURE. THE SPECIFIC ORDER OF DATA VALUES DEPENDS ON THE INSTRUCTION TYPE USED.	70
FIGURE 5.1: BULLET SIMULATION CORE COMPONENTS. THE <code>btDYNAMICWORLD</code> AND <code>btCOLLISIONWORLD</code> ARE HIGHLY INTEGRATED, WITH THE DYNAMICS WORLD REQUIRING COLLISION WORLD INFORMATION TO FUNCTION AS MORE THAN A SPECIALIZED <code>MLCP</code> SOLVER.	75
FIGURE 5.2: BULLET CONSTRAINTS OPERATE ON ONE OF THREE SOLUTION SPACES WHEN RESOLVED DURING A SIMULATION STEP. THE INCLUSION OF ADDITIONAL ERROR TOLERANCES CAN CHANGE THE RESULT FROM THE THEORETICAL RESPONSE.	76
FIGURE 5.3: LOCAL RIGID BODY ORIENTATIONS AND CONSTRAINT BASED JOINT FOR USE IN RIGID BODY SIMULATION WITH A JOINT ROTATION ABOUT THE Z AXIS BY $\theta = 90^\circ$. BODY 2 IS SHOWN WITH NO JOINT ROTATION, THE DEFAULT JOINT DEFINITION, IN WHITE.	77
FIGURE 5.4: MOTOR SIMULATION EXPERIMENT FOR A SIMPLE QUADRUPEL. THE SIMULATION FREQUENCY RESPONSE SHOWS HIGH VARIATION DUE TO ERP EFFECTS.	78
FIGURE 5.5: THE SIMULATION FLOW GRAPH OF A TYPICAL BULLET SIMULATION STEP WITH EXTERNAL MOTOR MODEL ATTACHED. THE GENERAL PROGRAM FLOW IS GIVEN ON THE TOP, WITH BULLET SPECIFIC IMPLEMENTATIONS ON THE BOTTOM.	80
FIGURE 5.6: COLLISION SHAPES FOR A SINGLE MOTOR MODULE. THE HIGH RESOLUTION TRIANGULATED MESH OUTPUT FROM CAD FILES IS SHOWN IN (A), THE CONVEX HULL OF THE TRIANGULATED MODEL IS SHOWN IN (B), AND A BOUNDING BOX APPROXIMATION IS GIVEN IN (C). THERE IS AN ORDER OF MAGNITUDE OF DIFFERENCE IN THE COMPUTATION TIME FOR COLLISIONS AT EACH SIMPLIFICATION STEP.	81
FIGURE 5.7: COLLISION SHAPE REPRESENTATIONS OF THE LOWER HALF OF THE ROBOT LEG. THE CAD MODEL CONVERTED INTO A TRIANGLE MESH IS SEEN IN (A), WHILE A POSSIBLE CONVEX HULL REPRESENTATION IS GIVEN IN (B). THE COMPOUND PRIMITIVES SHAPE IN (C) SHOWS A COMPROMISE BETWEEN COLLISION ACCURACY AND PERFORMANCE.	82
FIGURE 5.8: INSTABILITY DUE TO LOW SYSTEM INERTIA WHEN USING LOW SIMULATION FREQUENCIES SHOWN IN (A) CAN RESULT IN SIGNIFICANT JOINT STATE ERRORS, EVEN IN THE CASE THAT INCREASED SYSTEM INERTIA PRODUCES STABLE SIMULATIONS, SEEN IN (B). SYSTEM WAS TESTED WITH A 4π Hz SINE WAVE STARTING AT 3s USING THE ELECTRO-MECHANICAL MODEL DESCRIBED IN SECTION 6.3 AT 240Hz AND 2000Hz, AND USING A SEPARATE FRICTION (SF) IMPLEMENTATION AT 240Hz.	85
FIGURE 5.9: A SIMULATION AND VISUALIZATION CLASS WITH MUTEX BARRIER BETWEEN THE SIMULATION AND VISUALIZATION. EXTERNAL TRIGGERS CALL THE FUNCTIONS RESPONSIBLE FOR SIMULATION AND VISUALIZATION INDEPENDENTLY AND MAY CAUSE A COLLISION.	88
FIGURE 6.1: SIMULATION ERROR OF SYSTEM USING A LATENCY COMPENSATED KINEMATIC APPROXIMATION. THE ORIGINAL TARGET AND EXPERIMENTAL DATA ARE PLOTTED ON A SECONDARY AXIS. A DYNAMIC APPROXIMATION IS SHOWN (EM) FOR COMPARISON. ...	91
FIGURE 6.2: SATURATION CONTROL OF SIMULATION DURING A BOUNDING GAIT. SAMPLING LIMITATIONS IN THE RAW DATA LEADS TO PERIODS OF NO DATA BETWEEN SATURATED MOTION RESPONSE.	92
FIGURE 6.3: ELECTRO-MECHANICAL MOTOR MODEL FOR A SERVO-MOTOR SUB-SIMULATION WITHIN A RIGID BODY SIMULATION PACKAGE. THE FRICTION APPROXIMATION, MOTOR CONTROLLER, AND MOTOR MODEL CAN BE USED WITH A BLACK BOXED JOINT MODEL. ...	94
FIGURE 6.4: SIMPLIFIED CONDITIONAL CONSTRAINT METHOD FOR SERVOMOTOR SIMULATION. THE IDEAL MOTOR MODEL AND MODE SWITCH CAN BE IMPLEMENTED INDEPENDENTLY OF THE MOTOR AND FRICTION MODELS AS A SECONDARY STATE PREDICTOR. ...	97
FIGURE 6.5: SIMPLIFIED CONDITIONAL CONSTRAINT MOTOR MODEL FOR A SERVO-MOTOR SUB-SIMULATION WITHIN A RIGID BODY SIMULATION PACKAGE. THE FRICTION APPROXIMATION AND MOTOR CONTROLLER ARE DEVICE DEPENDENT.	98
FIGURE 6.6: MOTOR RESPONSE TO A 4π Hz SINE WAVE (A) AND STEP RESPONSE (B) UNDER UNLOADED AND 700G LOAD CONDITIONS FOR EXPERIMENTAL AND CONDITIONAL CONSTRAINT METHOD OPERATING AT 240Hz SIMULATION RATE. ERRORS ARE ORDERS OF MAGNITUDE SMALLER THAN THE INPUT COMMANDS.	100
FIGURE 6.7: SIMULATION ACCURACY OF THE ROBOT PLATFORM WITH UNLOADED (A) AND 700G (B) ATTACHED LOAD UNDER A STEP FUNCTION FOR THE ELECTRO-MECHANICAL (EM) AND CONDITIONAL CONSTRAINT (CC) METHODS. THE TARGET FUNCTION AND EXPERIMENTAL DATA ARE SHOWN ON THE SECONDARY AXIS, WITH A POSITION RESOLUTION OF 0.00153 RADIANT.	102

FIGURE 6.8: SIMULATION ERRORS OF THE ROBOT PLATFORM WITHOUT ADDITIONAL LOAD (A) AND WITH A 700G LOAD (B) DURING A $4/\pi$ HZ SINE WAVE INPUT FOR THE ELECTRO-MECHANICAL (EM) AND CONDITIONAL CONSTRAINT (CC) METHODS. THE RAW TARGET FUNCTION AND EXPERIMENTAL DATA ARE SHOWN ON THE RIGHT-HAND AXIS.	103
FIGURE 6.9: QUADRATIC MEAN ERROR OF THE ELECTRO-MECHANICAL AND CONDITIONAL CONSTRAINT MODELS DURING A 4π HZ SINE WAVE TARGET ANGLE INPUT. AS THE ERROR IS OVER A MAGNITUDE OF ORDER LARGER, THE RESULT OF THE LATENCY CORRECTED KINEMATIC MODEL IS PROVIDED ON THE SECOND AXIS.	105
FIGURE 6.10: MAXIMUM ERROR OF THE ELECTRO-MECHANICAL AND CONDITIONAL CONSTRAINT MODELS DURING A 4π HZ SINE WAVE TARGET ANGLE INPUT. AS THE ERROR IS OVER A MAGNITUDE OF ORDER LARGER, THE RESULT OF THE LATENCY CORRECTED KINEMATIC MODEL IS PROVIDED ON THE SECOND AXIS.	105
FIGURE 7.1: A SINE WAVE INPUT DISTURBANCE SIMULATION FOR AN UNCALIBRATED AND CALIBRATED MODEL OF THE SARM PLATFORM. UNCALIBRATED ERRORS ARE ONLY marginally better than a pure kinematic response.	109
FIGURE 7.2: THE RESULTS OF A SINGLE OPTIMIZATION RUN FOR IDENTIFICATION OF NINE VARIABLES USING MCP SO, USING VALUES NEAR THE KNOWN SOLUTION TO IMPROVE CONVERGENCE TIME. THE BEST FITNESS (A) DURING EACH ITERATION VARIES SIGNIFICANTLY UNTIL PARTICLES SETTLE TO A MINIMUM SOLUTION. THE OUTLINE OF THE TRAJECTORY OF ONE PARTICLE CAN BE SEEN IN THE GAIN (B) AND STATIC FRICTION (C) AS A SINUSOID WITH DECREASING AMPLITUDE.	114
FIGURE 7.3: THE SARM RESPONSES AS A FUNCTION OF THE NO-LOAD CONDITION RESPONSE TO A $\pi/4$ AMPLITUDE STEP FUNCTION SHOWS STRONG CORRELATION TO LOAD CONDITIONS.	115
FIGURE 7.4: THE SARM RESPONSES AS A FUNCTION OF THE NO-LOAD CONDITION RESPONSE TO A 4π HZ SINE WAVE SHOWS AREAS OF STRONG CORRELATION AND WEAK CORRELATION TO LOAD CONDITIONS.	116
FIGURE 7.5: THE FITNESS FUNCTION RESULTS FOR MULTIPLE ACTUAL AND SIMULATION LOADING COMBINATIONS IS HIGHLY UNIMODAL, WITH FUNCTION MINIMUMS APPROACHING THE ACTUAL LOAD IN A HIGH DEGREE OF LOADINGS.	117
FIGURE 7.6: THE CONNECTION GRAPH FOR AN OPTIMIZATION BASED SOFT SENSOR. THE "SENSOR" EMPLOYS SUB-NETWORKS BASED ON THE ACTUAL CONTROL NETWORK. SUB-NETWORKS ARE INDEPENDENT OF THE FULL NETWORK AND CAN BE RUN IN PARALLEL.	118
FIGURE 7.7: SOFT SENSING RESULTS FOR A STEP FUNCTION (A) AND $4/\pi$ HZ SINE FUNCTION (B) FOR EACH 250MS LONG SENSING WINDOW STARTING AT 3.0s, 3.25s, 3.5s, 3.75s, 4.0s, AND 4.25s. THE LARGEST ERRORS COINCIDE WITH DYNAMIC RESPONSE SINGULARITIES SHOWN IN FIGURE 7.4.	119
FIGURE 7.8: THE 1 SECOND LONG SENSING WINDOW RESULTS FOR BOTH STEP AND 8HZ SINE FUNCTIONS (A), USING THE FRAMEWORK BASED STATE ESTIMATOR METHOD AND A TRADITIONAL TORQUE OBSERVER (OBS.). THE ERRORS FROM THE TESTED CONDITIONS (B) ARE WITHIN 20G OF THE MEASURED LOADING FOR ALL TRIALS, ESTIMATING THE ACTUAL MASS TO WITHIN 10%.	120
FIGURE A.1: CAD DESIGN SPECIFICATIONS FOR THE QUADRUPEd ROBOT, TOP VIEW.	131
FIGURE A.2: CAD DESIGN SPECIFICATIONS FOR THE QUADRUPEd ROBOT, FRONT VIEW.	132
FIGURE A.3: CAD DESIGN SPECIFICATIONS FOR THE QUADRUPEd ROBOT, SIDE VIEW.	132
FIGURE B.1: MAJOR CONTROL FRAMEWORK MODULES AND THEIR RESPECTIVE EXECUTION THREADS IN A FULLY ASYNCHRONOUS CONFIGURATION. SUBROUTINE FLOWS START AT THE TOP OF EACH EXECUTING SECTION AND FOLLOW THE GREEN ARROW, WITH ORANGE ARROW DELINEATING DATA TRANSFER OUTSIDE THE THREAD.	135
FIGURE B.2: MAJOR CONTROL FRAMEWORK MODULES AND THEIR RESPECTIVE EXECUTION THREADS IN A CONTROLLER AND SIMULATION SYNCHRONOUS CONFIGURATION. SUBROUTINE FLOWS START AT THE TOP OF EACH EXECUTING SECTION AND FOLLOW THE GREEN ARROW, WITH ORANGE ARROW DELINEATING DATA TRANSFER OUTSIDE THE THREAD.	136
FIGURE B.3: MAJOR CONTROL FRAMEWORK MODULES AND THEIR COMPONENTS. CLASSES ARE CENTERED ABOUT THE ROBOTSERVOMOTOR AND ROBOTCOMMUNICATIONS SUPER-CLASSES.	137
FIGURE B.4: CORE COMPONENTS OF THE INTEGRATED SIMULATION AND CONTROL FRAMEWORK SHOWING THE INTEGRATION BETWEEN THE SIMULATION AND OTHER COMPONENTS.	138

Nomenclature

DOF	<i>Degree of Freedom</i> An allowable direction of linear or angular motion
RMS	<i>Root Mean Square</i> The geometric mean of a set of data which is useful in providing a meaningful error value for periodic data that is not exclusively positive
TTL	<i>Transistor–Transistor Logic</i> A transmission voltage level which can be produced and received by logic transistors in a device, generally between 3.3V and 5V.
UART USART	<i>Universal Asynchronous Receiver-Transmitter</i> <i>Universal Synchronous and Asynchronous Receiver-Transmitter</i> A communications device type which transmits data sequentially without need of an additional clock pulse transmission (asynchronous)
IMU	<i>Inertial Measurement Unit</i> A device using accelerometer and gyroscope data to estimate motion
API	<i>Application Programming Interface</i> An externally accessible set of functions and data used in software development to define an information and control boundary
CRC	<i>Cyclic Redundancy Check</i> Binary data integrity checking method that can detect randomly introduced errors with minimal computations
Serial (communication)	A form of communication in which information bits are passed sequentially, often referring to TTL UART/USART implementations.
Baud Rate	The speed at which symbols can be transmitted across a communications device. Depending on the encoding of the transmission this can be smaller or larger than the effective bit-rate.
Object Oriented Programming	A software development methodology that uses objects, self-contained data and methods defined by a class, for execution of similar ideas encompassed by the class
C style interface	An interface using ANSI C definitions and methodology for use in integrating with a wide variety of languages
Context (switching)	A stream of processor commands and memory which define a currently executing operation. Only one context is active per logical processor at a time.
Atomic Operation	An operation which is exclusive to a single context, either because the operation can be executed in a single cycle, or because of locking mechanisms in hardware or software.

1 Introduction

1.1 Motivations

The field of biomimetic robotics has attempted to solve many design and control problems in robotics by looking at nature for inspiration. Legged robots have been studied for use in dangerous environments where terrain traversal would be difficult for wheeled and tracked robots. Currently available quadruped platforms are prohibitively expensive for academic research, requiring tens of thousands of dollars for basic hardware and its service. It is typical to see a single platform shared between several laboratories across various universities due to cost, limiting the ability of researchers to experiment directly on the hardware.

Modularization of the robot hardware and software minimizes manufacturing and maintenance costs typically associated with highly articulated robots, especially if the number of sensors per module are reduced to the minimum number necessary for control of the servomotors in the modules. Although modularized robots using low-cost parts have been widely used for biomimetic robot research, several problems remain in the design and control of such systems due to their black-box nature. While some of these problems are shared with additional robot types, such as pathfinding and environment localization, there are a significant number of problems unique to modularized robots that require a capable platform to develop and validate. The lack of available low-level control and estimation systems which can deal with the intricacies of modularized hardware has limited the applicability of low-cost modules for research.

As such, the development of a hardware platform and a full simulation and control framework was started with the intent to apply this to various available modularized robots. Hardware, communications, and simulation performance are highly intertwined, with limitations in one affecting the overall performance of the others. A comprehensive methodology to minimize these issues, and mitigate any that cannot be resolved through careful design, is necessary to overcome the restrictions due to the low-cost devices employed. To further improve the performance of the framework in the context of robot research, tools and techniques to calibrate the system in various aspects are required.

1.2 Contributions

This thesis makes the following primary contributions:

- A low-cost, portable quadruped robot and motor test platform which satisfies the needs of research oriented robotics at a reasonably low cost.
- A control and simulation framework for modularized robots
- Motor identification methodology and a simulation optimization based soft-sensor

1.3 Outline

The core components of the thesis will be discussed in the following structure:

Chapter 2: The background of the current state of quadruped hardware, control frameworks, and simulation research will be explained in this chapter. Commercial, military, and research oriented quadruped hardware designs that influenced the modular quadruped design will be discussed. Robot simulation techniques that are applicable to real-time systems will also be introduced.

Chapter 3: The design of a modularized quadruped robot addressing the problem of a low-cost quadruped platform will be introduced and kinematic and payload specifications analyzed. The primary components and their physical and control properties will be discussed, with an emphasis on self-contained servomechanisms that comprise the core of actuation modules. This chapter will also discuss simplified test platforms used in identification and calibration of the control framework and simulation.

Chapter 4: This chapter will introduce a low-level control and simulation framework which can be used with low cost modularized robots. The chapter presents a solution to the problem of integrating simulation and control at a lower level. The core aspects, interactions, and API surface of the framework will be outlined, including the actuator, communications, platform, data recording, and simulation as well as their interactions and optimizations. Additional components of the framework that work in conjunction with the core components will be discussed, in particular the non-linear and meta-heuristic optimization methods developed to improve automation of system identification and develop a basis for alternative soft-sensing methods.

Chapter 5: This chapter will discuss the methodologies necessary to provide a high-fidelity simulation of the robot platform and its environment. Assumptions and simplifications used within the simulation method will be analyzed in terms of their effect on the problem of simulation fidelity and performance. Discussion of simulation stability as a limiting factor in performance and accuracy will introduce the need for stable joint models.

Chapter 6: This chapter will address the imperfect joint problem for low cost quadrupeds in terms of a traditional and a new approach. As the joint model is separated from the simulation itself, simulation independent models of the joints must be used. The traditional electromechanical model is introduced and analyzed in terms of accuracy and stability within the simulation environment. Next, a novel rigid body constraint based model for the simulation of internally controlled articulated robot modules is proposed which can reduce simulation instabilities at larger time-steps without requiring simulation specific non-physical damping or closed-form dynamic solutions. The analysis of these two methods in terms of the robot platforms is discussed and the performance improvement for the novel method is demonstrated.

Chapter 7: This chapter addresses the soft-sensing solutions to the environment and robot state sensing problem using the techniques described in the previous chapters. High fidelity simulations require significant tuning, and the preliminary identification problem for a modularized robot is addressed by means of meta-heuristic optimization of the models discussed in Chapter 5. Next, the soft-sensing problem is explained in terms of an optimization problem in a similar form to that of the identification problem, and currently available methods are introduced. An optimization based abstracted state estimation method is proposed and experimental results of the method are used to validate the method.

Chapter 8: This chapter concludes the thesis and provides additional discussions on the previous chapters in the context of additional problems in quadruped robotics.

2 State of the Art

2.1 Quadruped Robotics Electromechanical Hardware Design

Mobile robotics, and in particular legged robotics, have long been explored for a variety of uses from replacing humans in hostile environments to commercialized entertainment.

Although simplification of the mechanical and control aspects of these robots improves the cost-benefit ratio significantly, certain tradeoffs may need to be made in order to achieve these lower costs. In particular, the number and quality of sensors greatly impacts the cost and also data accuracy of a robot, with lower cost sensors typically giving lower resolutions and accuracies than higher cost models.

2.1.1 Sony Aibo

Often considered the most advanced commercial robot, the Sony Aibo was the forerunner to a significant number of quadruped test platforms focusing on path planning and cooperation. Although the product was first and foremost a consumer entertainment device, the mixture of features and physical design made this a popular robot platform for quadruped research. Averaging about \$1500 at launch, these robots were also very affordable, allowing even smaller research groups to obtain the necessary hardware for research.

The mechanical design of three active degrees of freedom (DOF) in a hip (2DOF) and knee (1DOF) allows for both in-sagittal-plane motion and limited out-of-plane motion [1]. This arrangement has been replicated for most quadruped robots since the initial release of the Aibo. Despite a large number of sensors and control equipment for its time, the Aibo had dimensions of approximately 27cm by 27cm by 16cm with a mass of only 1.6kg. This small size allows for research in confined spaces, as well as easy transport by researchers. A novel feature of some second generation models, as seen in Figure 2.1, was the modularization of the mechanical and control systems, allowing each leg unit or main body to be replaced with newer versions, as well as simplified repair of broken hardware. Starting with the second-generation models, sensors and communications equipment necessary for semi-autonomous operation were available.



Figure 2.1: Second generation Sony Aibo with integrated camera and Wi-Fi connection [2]. Few platforms have had as much widespread use in robotics research as the Aibo line.

The research capabilities of the device were very good for the time, featuring vision and contact sensors that could be used to facilitate path planning in dynamic terrains. Although a reduced functionality scripting language was available, the configurability of the primary control software allowed for the robot to be significantly mechanically modified without major changes to control schemes. Servomotor control is separated into a high level joint planning computed by a central microprocessor and a low-level motor control computed per leg module, allowing some configurability of the lowest level control algorithms through the use of gain scaling and joint target modifications.

Due to decreasing commercial popularity, the line was discontinued in early 2005. However, Aibo based research continued to be a popular topic among path planning and cooperative robotics researchers until the introduction of capable bipedal robots with reasonably stable control algorithms.

2.1.2 Boston Dynamics Quadruped Line

Starting in 2005, Boston Dynamics, a Massachusetts based robotics company, was funded by the Defense Advanced Research Projects Agency (DARPA) to develop a semiautonomous quadruped robot for the Legged Squad Support System (LS3) infantry troop support. This project was rolled out under several stages and subprojects, including the Learning Locomotion project, and produced three noteworthy quadruped platforms, the LittleDog, BigDog, and AlphaDog. These three platforms demonstrated sensory based control concept, semiautonomous capability, and product worthiness respectively. Although this project was ultimately rejected due to mechanical limitations, the advancements in quadruped motion and control were substantial.

Of these products, only the LittleDog had potential as an academic platform, having been tested by various universities as a part of the DARPA project to demonstrate robust locomotion over rough terrain. This platform is very similar in size to the Aibo systems, 34cm by 18cm by 14cm in size and weighing

3kg with the full complement of batteries, sensors, and onboard computer [3]. This small robot is powered by three high performance electric motors per leg with an additional passive degree of freedom in the ankle, with sensor feedback at each joint through the use of high resolution encoders in the active joints and force sensors in the passive joints. A model modified to allow for visual servoing can be seen in Figure 2.2. Although no official bill of materials or product cost has been publicly released, a development and build contract for units to be provided to six universities was initially set at \$1.6 million dollars. Accounting for development costs, each robot likely cost over \$10000 per unit, plus additional costs for non-standard sensors such as vision equipment.



Figure 2.2: Boston Dynamics LittleDog research platform walking over an obstacle. Motion capture and Wi-Fi connectivity accessories have been added to enable remote visual servoing [4].

The BigDog was simultaneously created with the LittleDog as a test platform for lower level control, balance, and different levels of control autonomy. This platform is significantly larger, with the largest dimensions being around 75cm and 90cm, as well as three orders of magnitude heavier at 120kg. Size constraints and actuation performance necessary to move the increased mass required the use of hydraulic actuators rather than electric motors generally seen in smaller platforms. As with prior platforms, the BigDog uses a two DOF hip and one DOF knee configuration, with an additional passive linear joint design shared with the LittleDog. Although the overall design is a continuation from prior platforms, the level of sophistication in the mechanical design is significantly improved. This improved design however lead to a significant increase in cost, with the original robot costing over \$1 million, in addition to development costs for sensor improvements added after the preliminary demonstrations.

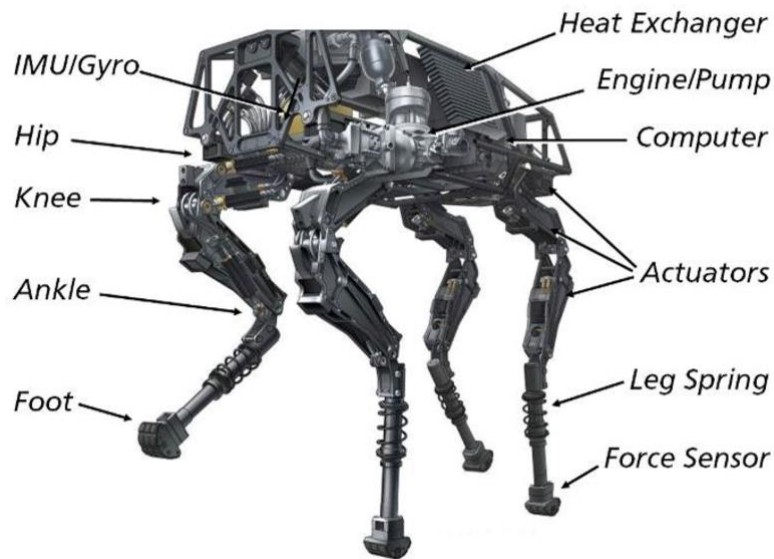


Figure 2.3: Public presentation of the BigDog, an advanced quadruped robot for military applications [5].

The resounding success of the BigDog led to the development of a pilot LS3 design named the AlphaDog (Figure 2.4). This platform further refines the design to improve payload transport capability, allowing the platform to carry nearly 200kg in load for further distances with improved autonomy. The physical configuration is identical to the original BigDog, however all aspects were redesigned for increased robustness and load carrying capability. Several AlphaDog units were eventually built, however the final hardware and designs released well after the program had little additional impact on robotics research and quadruped hardware design compared to the BigDog and LittleDog.

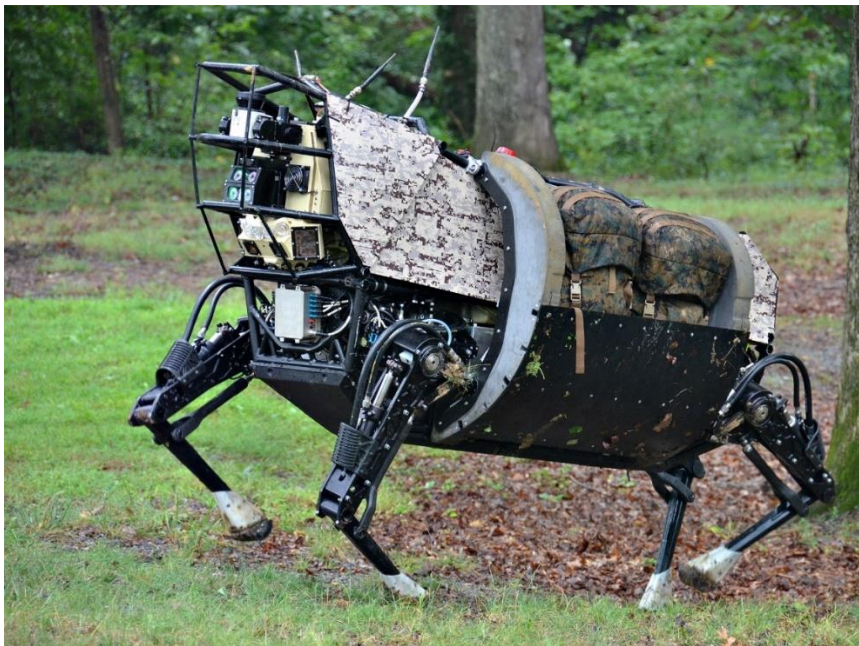


Figure 2.4: A refined prototype of the LS3 platform, the joint definitions were significantly changed from the BigDog design. Additional external payload was added during demonstrations to simulate in-field use [5].

2.1.3 HyQ

In 2010, the Istituto Italiano di Tecnologia (IIT) developed a robot platform very similar to the BigDog with a focus on academic research capabilities [6]. As with earlier robots, the HyQ uses a two DOF hip and one DOF knee configuration. However, to reduce weight and hydraulic pump requirements, the HyQ uses a hybrid electric motor and hydraulic actuator design for the hip joint, powering the less responsive out of sagittal plane motion with a highly geared electric motor and the higher performance in sagittal plane component with hydraulic actuators. As with the Boston Dynamics robots, IIT does not disclose the development or BOM costs of the HyQ. From known information about the use of Harmonic Drive gears, Moog servo valves, and additional components, a minimum bound of \$15000 in materials costs alone can be assumed, however the actual production costs could be ten times larger.

In 2015 an updated HyQ with a smaller, lighter build was developed [7]. The most significant change from a control perspective is the switch to a hydraulic actuator for the out-of-plane hip joint and replacement of a linear hydraulic in-plane hip joint with a rotary hydraulic actuator. As can be seen in Figure 2.5, the mechanical design is close to the Boston Dynamics AlphaDog, however the size and mass are significantly lower. Research with the HyQ is ongoing, however the number of participating institutions and small production scale have limited the number of publications on novel control and sensing using this platform.

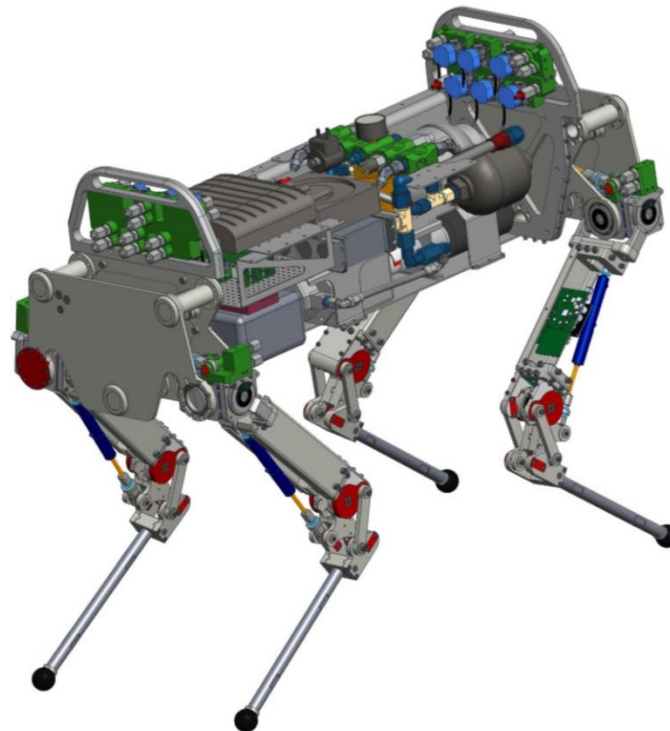


Figure 2.5: Third generation HyQ platform with reduced complexity compared to the original platform [7].

2.2 Robot Pose

In order to move a robot with respect to a target position, is required. A well-defined robot internal and external state, generally referred to as the robot's pose, and has been the topic of a significant number of papers due to the complexity of defining a robot pose for non-trivial systems.

Robot pose can be determined by a prescribed gait, a template joint motion characteristic to the particular robot and locomotion type. These gait motions can be continuous, as in walking or galloping, or discrete robot motions. Gaits can be described by several parameters:

- Gait width
- Leg phase
- Leg ground contact duty cycle
- Gait frequency

2.2.1 Kinematic control and inverse-kinematics

Kinematic control of robots has been used to produce a target pose in industrial and commercial robots due to the simplicity in trajectory definition and use. For simple mechanisms, generally traversing only two dimensions, well-defined closed loop joint angle equations are possible when the number of actuation DOF match the trajectory dimensionality. These equations can map the desired end-effector trajectory in a world frame space and time to a semi-unique joint angle set and time. In the case of the articulated robot in Figure 2.6, the joint angles may not be unique for a given endpoint trajectory, as there are two configurations for the leg in which the end-effector position is identical. This type of system is generally solved using a preferred direction for each joint, which reduces the possible workspace but allows for simplification of the kinematic equations, allowing for unambiguous solutions to the inverse kinematics problem.

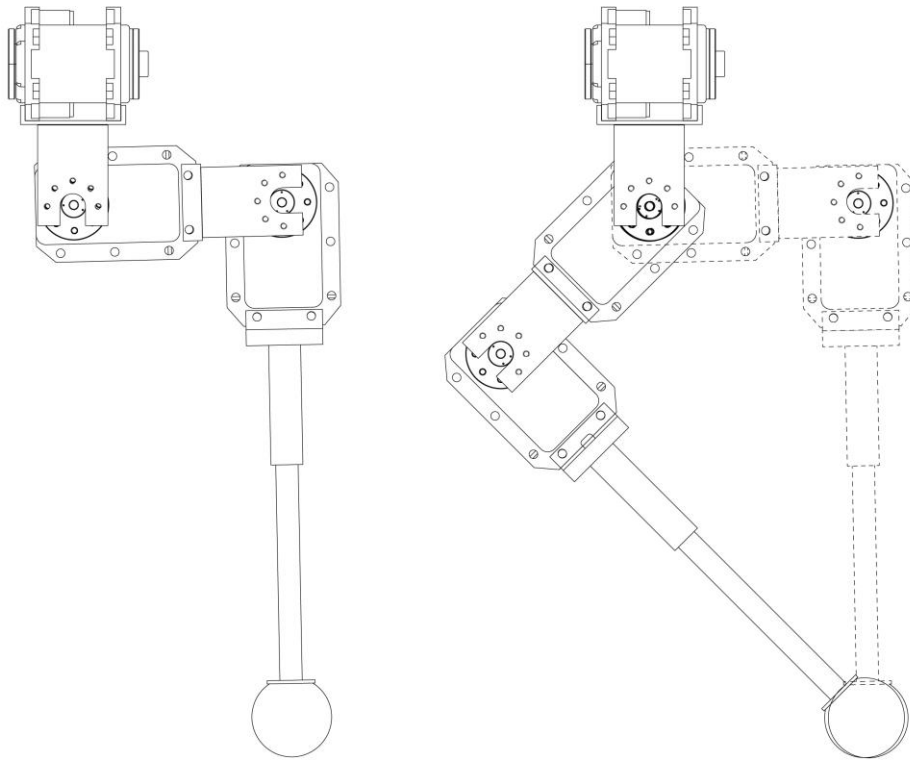


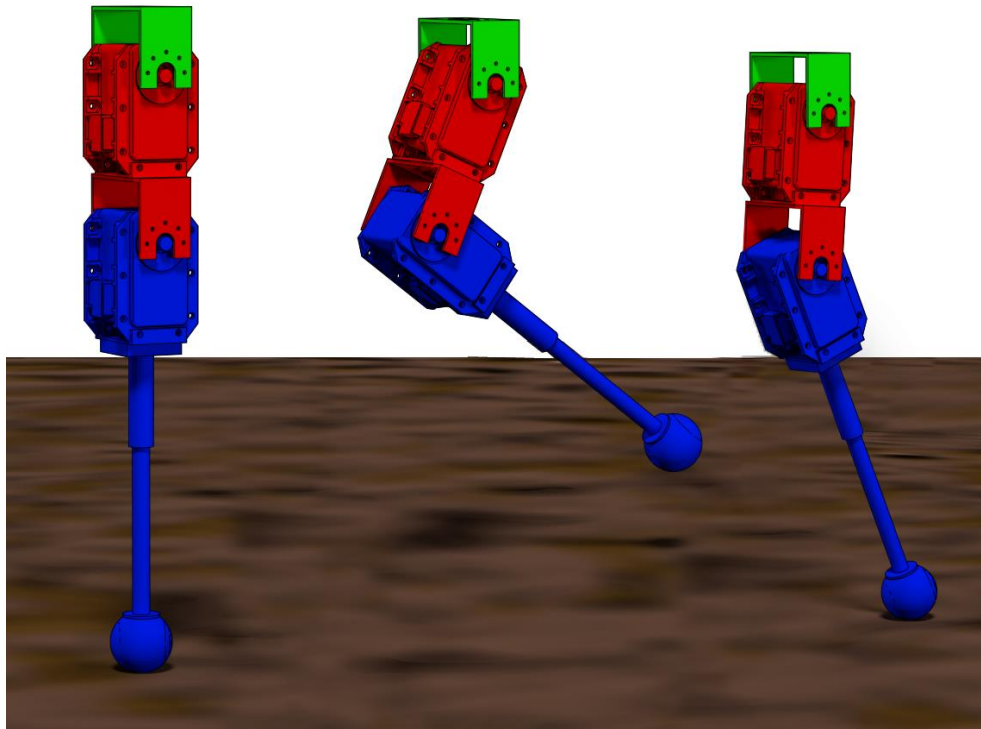
Figure 2.6: Multiple joint space solutions for a redundant inverse kinematic end-effector point. The ground plane contact point is identical despite significantly different dynamics conditions.

For kinematic chains which are not at a singularity point, where all possible solutions converge to a single, trivial solution, the preferred direction methodology can be expanded using prior joint angle knowledge. In this solution, the joint angle changes are minimized between the redundant possibilities, dictating a single solution as an optimal one. Only the prior step is required in addition to the inverse kinematic equations for this methodology, allowing for quick computation without additional information on the system. For more complex robots and trajectories, non-kinematic solutions to the inverse-kinematics problem are possible, with a wide variety of benefits and computational costs associated with each.

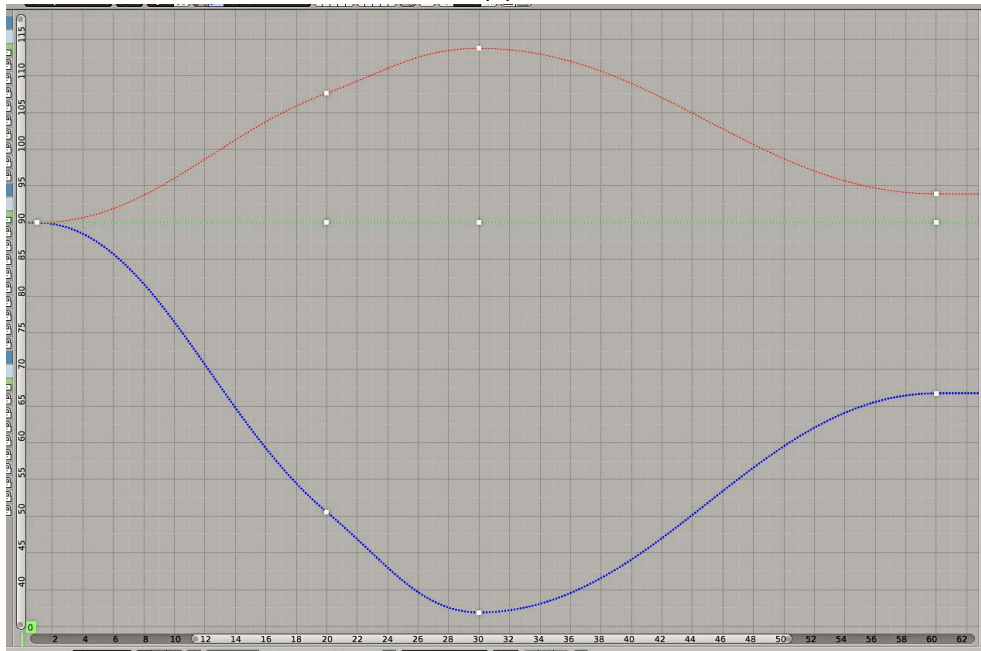
2.2.2 Scripted Gaits

An alternative solution is the application of forward kinematics methods, in which the joint space is directly controlled to produce a pre-calculated script for a gait fully describes the robot pose in an unambiguous way. The most common method is the development of via points, key pre-determined joint space pose definitions at a specified point in time that the robot gait must pass through to accomplish the locomotion goal, and subsequent interpolation between these key points at run-time. Given a specific robot configuration, robot pose in the joint space can be precomputed with kinematic transformations derived by manual, motion capture, or even optimization based key point playback. Interpolation between points during an actual motion can be accomplished by spline or linear interpolator, often a Bézier curve, to provide a smooth trajectory target throughout the motion. Figure 2.7 shows an example kinematic chain motion in cartesian coordinates (a), and the required joint space

Bézier curve for the trajectory. As the motion is only described in joint space, additional information for the motion of the origin point is required. In the case of complex highly articulated robots, kinematic chains may be represented in terms of the endpoints of other kinematic chains, simplifying the conversion between joint space and world coordinates.



(a)



(b)

Figure 2.7: Kinematic leg control key points and interpolated composite Bézier curves for intermediary joint angles.

When these gaits are set such that the start and end of the script are the same pose, the gait can be looped through as necessary to produce continuous motion between multiples of the gait length. As

target points rarely produce exact time multiples of a given gait, scripted gaits are often calculated in time-normalized terms, separating the gait script from the gait trajectory implementation [8]. In the case of a walking gait for a footed robot, each leg cycles through a predetermined script at a rate prescribed by a second, higher level script. Each script is separated into a ground contact and air motion segment, and the higher level script controls the duration of each segment for all legs. As such, the gait can be broken down, at a high level, into a duty cycle, the ratio between ground and air, and phase, the timing between legs, as shown in Figure 2.8.

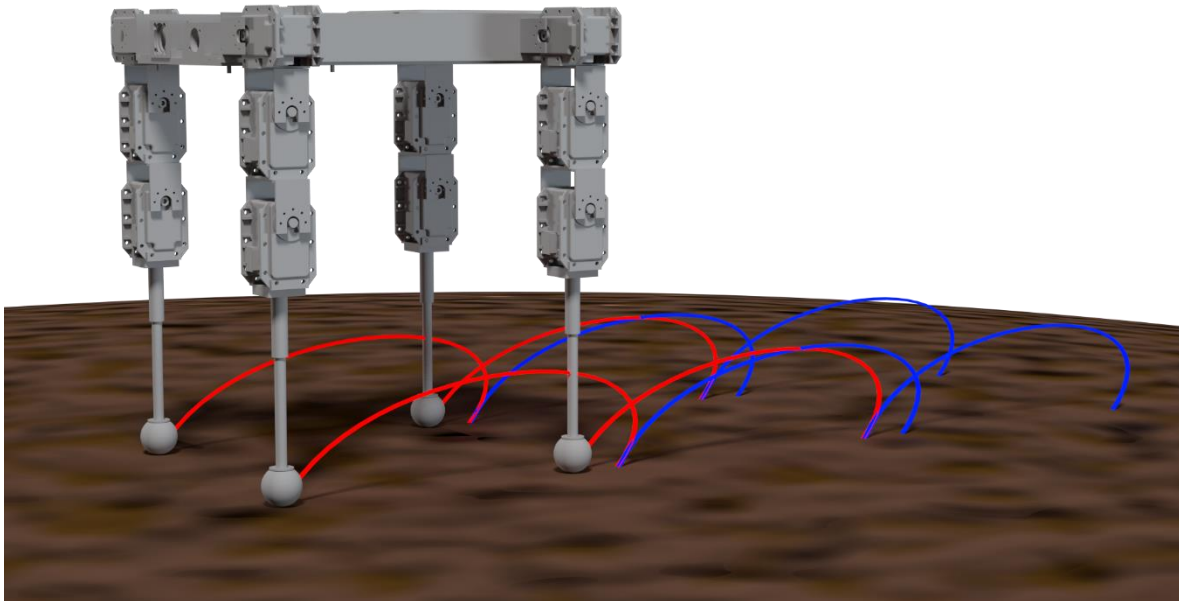


Figure 2.8: Quadruped robot gait projection, with the first cycle in red and second cycle in blue. The gait type is determined by the phase and ground contact duty cycle of leg motions rather than the trajectory of the leg in the air.

Fixed script gaits have near instantaneous computation due to the simplicity in position computation, however they are limited in application due to the additional requirement of position multiples of the base scripted gait. Although velocity can be modulated through the time-normalized method, the length of a single gait cycle cannot be changed. However, it is possible to again separate the lower level script into the desired end-effector relative to the start point through the script, and the actual trajectory required to achieve the desired pose. This produces a hybrid approach, where the positions of the kinematic chain endpoints are scripted, but the final joint motions are computed through inverse kinematics or more complex means. This approach is the basic building block of highly articulated robot motion, with further layers of end-effector and joint position control adding additional functionality for more locomotion in more demanding environments.

2.3 Simulation and estimation in robotics

Due to the complexity of hardware systems and difficulties in development of hardware and control systems, virtual representations of robotic systems are widely used to test new concepts, both hardware and control. Kinematic and dynamic models have widely been used for trajectory simulations

and feed-forward controls, especially in the industrial robotics realm where robot hardware is well understood and built to high quality standards. These methods compute the robot state from a fixed base point to a mobile end effector with little computational effort, however all joint angles must be accurate and the base point exactly known in cartesian space. In contrast to traditional, fixed base robotics, mobile robots require state estimation that cannot be fully described by their construction and joint positions due to an unknown orientation within the world they occupy. When hardware orientation sensors, particularly inertial measurement units, are available, a relative base can be utilized to estimate the world position given an assumption of ground contact and sensor based orientation. This system state information requirement for direct kinematic approximations limits its use to a limited number of problems in closed-loop control of hardware.

To address this limitation, extensions to the kinematic approximations, such as the virtual chassis method used by Choset et al [9], have been used for mobile robots due to high computational performance and no requirement on the knowledge of the internal system dynamics. This virtual chassis attempts to extend the standard kinematic approach to floating contact models such as a walking quadruped by means of a shape minimization function which redefines joint positions based on an average of all components [10]. The method can be used to approximately place a robot with respect to the ground plane and gravity, as well as simplify calculation of the next orientation based on the current ground contact. While this model has been successfully applied to high performance biomimetic snake robots, the methods shortcomings prevent it from being useful when imperfect motors and quadrupeds are included. In particular, the methodology described by the virtual chassis method can only apply while a robot has full, stable ground support. During high energy gaits where ground contact may be unstable or non-existent, as in a gallop or a non-static walk, this method may attempt to ground the model incorrectly and result in misidentified ground contact or orientation.

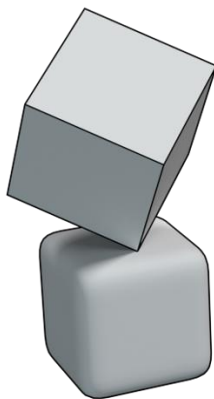
For more accurate and robust approximations of a mobile robot state, especially when using servo-motors with internal controllers, a dynamic model including any internal controller loop of individual joints is required. In a trivial implementation, this can be achieved by setting a torque and velocity saturation limit and applying forward dynamics based on these limits. Agüero et al. showed that a simplified dynamics equation with reasonable accuracy is possible if the system inertias are well known and component rigidity is high [12]. Similarly, Akdağ et al. have shown that much higher simulation quality can be achieved by defining the full dynamic model including material deformations in relatively rigid devices [11].

Although simulation generally provides viable results with generic models that are reasonably close to the actual system, the accuracy of the simulation compared to experimental data is dependent on knowledge of the system and numerical stability of the simulation stepping [13]. The dynamics of brushed and brushless motors used in typical smart-servos have been well documented [14] [15] [16] and can provide high accuracy when solved with implicit integrators or very small time-steps on explicit integrators. Electric motors of various types have been accurately simulated by calculating the current-induced magnetism and magnetism-induced forces as an electrical current is applied, as well as the effect of the changing magnetic fields on the current. The traditional approach for direct-current motor based servo-motor simulation applies torques derived from the electro-mechanical properties

of the motor and additional friction parameters directly to the joint constraints. Sobanski and Orłowska-Kowalska [17] showed that very detailed electrical models, simulating electrical properties down to the level of internal current switching, could be used to estimate the motor accurately enough to detect partial failures at high speeds. Reverse engineering of the electrical components is often required to obtain the motor and controller parameters [18] [19], however certain parameters, such as friction, must be experimentally determined for a given platform for best performance of the simulation.

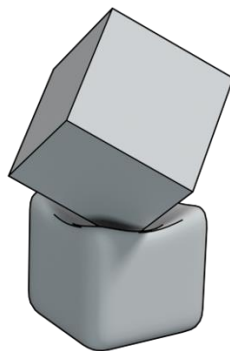
Although there are a large number of unique solutions to the problem of system dynamics, there are three main groups that are commonly used to represent robots and the physical environments in which they are moving. Dynamic simulations can be separated into these categories based on the assumptions used to estimate the internal response to externally applied forces and surface constraints. In terms of a mostly rigid robot where force applications propagate almost perfectly to the center of gravity, this can be referred to as the collision response of the system. While actual methodology varies, dynamic simulations fall into rigid body, soft body, or particle fluid types as shown in .

Rigid Body



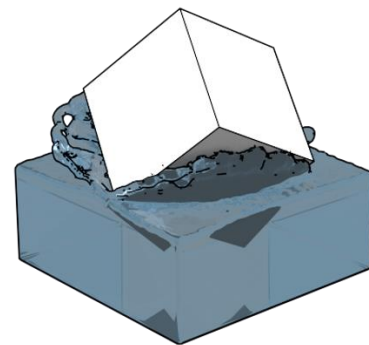
(a)

Soft Body



(b)

Particle Fluid



(c)

Figure 2.9: Collision response comparison between rigid (a), soft (b), and particle body representations of a rigid body with sharp edges impacting a second body. The deflection amounts are exaggerated beyond what would normally be seen in a robotic system for the purposes of illustration.

2.3.1 Pure Rigid Body Method

Rigid body approximations offer the simplest formulation and fastest simulation by simplifying the system to the minimum degrees of freedom necessary to treat each overall component independently. Objects can be treated as a collection of bodies that affect other bodies only by forces that can be resolved to equivalent transformations of the respective center of gravities.

Each individual body is assumed to have a fixed collision shape and invariant inertia within the time step, and all external forces are applied on center of gravity based on the equivalent force at the external surface of the collision shape. As the shape and mass distributions are fixed, the effects of plastic deformations and elasticity of the object are for the most part ignored. One exception for this is restitution forces during collisions, which are the effective compression and decompression of the impact site as objects overlap during imperfect collision tracking. The deformation area and depth are

invalid parameters within a rigid body definition as object overlapping is not possible, however fixed time stepping dynamics results in most object collisions experiencing small amounts of overlap when relative velocities are non-zero. For this type of system, restitution is instead calculated based on additional collision information in addition to the object momentums and coefficient of restitution, including unresolved collision penetration and contact pair relative velocities.

The relationships between bodies may be computed by several methods in order to join independent bodies into a connected system. Dynamic models of mobile robots can be created using a variety of closed form [20] and bond graph representations [21], however, floating base systems with collisions are significantly easier to model using constraint based methods utilizing classical dynamics concepts and bounding equations to limit or permit motion at well-defined joints [22] [23] [24]. Although any number of objects may be interconnected through the same constraint, such as in the case of geared systems with multiple linkages, the vast majority of available rigid body method solvers simplify systems to equivalent two-body representations of each constraint separately. As such, physical connections to multiple bodies are represented as multiple independent constraints between connected body chains.

Discrete-time simulation as a set of rigid bodies connected by constraints is particularly useful in the case of modular robotics, as each component is already considered an independent rigid body with pairwise constraints. The calculation of the equations of motion for each joint connected body is achieved by formulating velocity domain constraints and external forces as a set of mixed linear complementary problems (MLCP), which are then solved within a desired threshold for each constraint. Generalized MLCP solvers can be used, however performance and stability gains can be obtained from a rigid body physics optimized solver with constraint tolerances [23].

Robotic simulation frameworks using generalized rigid body physics solvers [25] [26] [27] for simultaneous collision, joint, and friction resolution have become the norm, even for complex robot platforms [12] [28]. In particular, constraint and impulse based velocity domain physics solvers based on the work of Baraff [22] and Mirtich [24], including Bullet Physics Engine [29], Box2D [23], and Open Dynamics Engine [30], have been used within major robot simulation frameworks including Gazebo [27] and Webots [26] for over a decade.

2.3.2 Soft Body Method

Soft body approximations, an alternative to the rigid body approximation, consider bodies to be flexible components with changing collision shapes and mass distributions rather than permanently defined solid objects. Although a variety of solutions exist for soft body representations, the two most common methodologies are sparse, shell approximations and dense, finite element methods. Shell approximations generally defined as a floating surface comprised of interconnected vertices with well-defined relationships only to adjacent vertices. The more common of the two methods is the floating surface approximation, as the surface tension is easier to calculate using spring-damper models that can be implicitly integrated [31]. As the calculations are limited to the object surface this method is most often used to provide high performance simulation of cloth and other flexible sheets, however it

can also be used for three dimensional shapes of arbitrary shape. When used with solid shapes, the effect is equivalent to that of a balloon, with the deformation only dependent on local contact.

The second most common approach is that of grid based deformations, including finite element (FEM) and mesh deformation methods, which provide high quality deformation mechanics [32] [33]. Finite element methods use well-defined internal nodes with closed form solutions [34] to calculate the state of each node, and by extension the system. These nodes are distributed throughout the object, and the individual nodes may be of any size or convex shape with a closed form solution. The exact formulation of each node varies based on the element equations used, and may include any number of variables including strain, stress, temperature, and adhesion, with more advanced formulations requiring significantly more computation than simpler, stress-strain only formulations. Mesh methods, on the other hand, approximate the finite element method using local displacements of nodes with reduced matrix density compared [35]. In the method described by Debunney et. al., a reduction in system matrix density removes additional computations that should, for all internal nodes, be equivalent to zero, while adaptive level of detail minimizes node density in volumes further from input forces.

Due to this distributed calculation of the shape, constraints to the object as a whole are difficult to implement, reducing viability for use in robotics. Except for trivial cases where the entire body is comprised of a single node, where the same definitions as in the rigid body case apply, any constraint would need to be calculated based on surface or even volume which can change shape during or between simulation steps. This increased computational complexity limits the viability of these methods for real-time computation, often requiring very large and costly high performance computers to approach viable simulation accuracy when closed loop control is included.

2.3.3 Particle Fluid Method

Particle fluid can be considered a large-scale implementation of the rigid body methodology with using point collision spheres with a large, non-linear restitution margin. Individual particles have infinitely small collision radii and inertia, but well-defined mass and position information, as well as parameters for adhesion to other particles within an effective radius. As the particles are independent, the problem can be efficiently calculated in parallel on modern general-purpose graphics processor units, allowing for real-time simulation in certain situations. As with soft-body systems, a constraint on the system becomes an action upon a surface or volume, where any particles within that sphere of influence participate in the constraint. However, the computation of physical constraints such as hinge joints, becomes significantly more difficult than even in soft-body systems, as the definition of such a constraint is non-intuitive and difficult to optimize due to the nature of these particles not necessarily having a defined orientation. For the purposes of robot control and simulation, particle methods were ruled out early in the development of the simulation framework. However, several tests on the viability

of such a system for use with aquatic robots were conducted, utilizing a hybrid model to represent a rigid or soft body robot with a particle fluid environment.

2.3.4 Hybrid Methods

In many cases, an approximation to both non-rigid methods can be implemented by connecting contactless rigid bodies with constraints, allowing these methods to be used within rigid body methods with minimal additional effort, albeit at a high-performance cost. The Bullet Physics Engine provides samples using FEM and particle fluids implemented as a collection rigid bodies, without self-collision, connected by a variety of constraints including 6-DOF spring models. At high simulation rates and node count these methods provide reasonable results for internal nodes, however external nodes may show collision artifacts due to surface irregularities. Although the use of soft-body computation within rigid-body packages is common for visually pleasing cloth collisions, the physical accuracy of the full interaction is often not high enough to be viable as a physical simulation.

3 Hardware and Communications

3.1 Introduction

Until recently, highly articulated quadruped motion was rarely used in academic circles, as the calculations involved in foot placement and balance were too slow to produce fast, accurate motion of the minimum of twelve joints necessary for relatively dynamic motion. In recent years, quadruped platforms like the Istituto Italiano Di Tecnologia's HyQ and the Boston Dynamics LS3 and BigDog have gained a large amount of press coverage thanks to their high level of control and fluid motion. These quadruped robots share many similarities in their design, including:

- Hydraulic actuators at all leg joints, onboard pump
- On-robot control computer with RTOS
- Off-robot computer for accessory functionality
- Very long operational time, on order of hours
- Dimensions of roughly 1m long and above 90kg mass
- Dozens of sensors: position, force, gyroscopic, lasers, cameras

With these high specifications, there is a commensurate price premium for both initial cost and operational expenses. Few institutions in the world are capable of supporting such robots, and in many cases the cost of development and operation significantly outweigh the benefits of advancements to the field of robotics these projects can bring.

This chapter will initially introduce design goals for an academic research focused robot platform, as well as hard and soft constraints on the electromechanical to effectively meet as many of the design goals as well as possible. The following sections will focus on the overall electromechanical hardware design methodology and construction of a quadruped robot and several test platforms, including comparisons to biological counterparts and required kinematic simplifications to produce a viable robotic platform that meets the desired criteria. The chapter will conclude with specific hardware implementations that were used to validate the simulation and control aspects of the robot system.

3.2 Requirements and Goals

A core component of robotics research is the physical robot platform used in experiment to collect real-world data to prove the validity and effectiveness of theoretical and simulated advancements. Although simulation based controller and mechanical design is a critical tool of academic focused robotics, the imprecise nature of physical test platforms can be used to validate the robustness of systems and techniques developed offline. In addition to testing the transferability of research into

finalized products, the hard-real-time limitations and cost considerations of a physical system validate the effectiveness of control, sensing, and design strategies beyond purely academic use. Tradeoffs in the development stages can require significant changes in the overall system specifications and purpose, and may require novel solutions to adapt a desired component into a system which would have previously been incompatible.

The overall purpose of this quadruped robot platform is to facilitate research in the field of locomotion, with an emphasis on minimal sensor control. The developed quadruped robot, as an academic oriented platform, was required to meet several performance and design goals:

- Robot should be able to carry a significant payload
- Robot should minimize control complexity
- Robot should be portable by a single researcher
- Robot should be easy to upgrade and expand
- Robot must be low cost and simple to build

To achieve these requirements, the designed robot platform uses a modularized approach previously implemented on snake-like robots used in pipe climbing and general locomotion [8] [36]. A trend towards low cost modularized designs for academic [37] [18], consumer [38], and commercial robotics [9] has been seen in recent years, particularly in biomimetic mobile robot systems. Modularization of articulated robots allows for simplified construction and maintenance while maintaining the ability to interchange actuator specifications as necessary. Rather than using specialized components for each body section, joints with standardized actuators and sensors are housed in a standardized structure, forming a module that can be connected to other components via universal physical connectors. Articulated modules are encased in a uniform physical container with any number of articulation points, and generally are able to be assembled in all possible configurations with few tools due to universal electrical and physical connections. Each degree of freedom is generally powered by a small DC motor with position encoders and additional sensors, as well as a digital control processor. In many cases, these modules are comprised of commercially available digitally controlled servo actuators that serve as partial black boxes, with only general performance and control specifications available.

As a result of modularization, system prototyping costs can be minimized, since any iteration that uses the same core modules requires minimal modification to achieve a new configuration. Figure 3.1 exemplifies several configurations of articulated mobile robots that can be achieved through modularized design methods. Robots can be made with any number of legs, or even no legs, by electing a new module mounting configuration. Each configuration can be controlled using gait patterns unique to a configuration family, such as sidewinding in snake-like robots or gallops in quadrupedal designs. Additionally, the robots can be made fully modular, as is the case with the snake-like configuration, or they may have additional configuration specific structures as in the case of the footed robots shown in Figure 3.1.

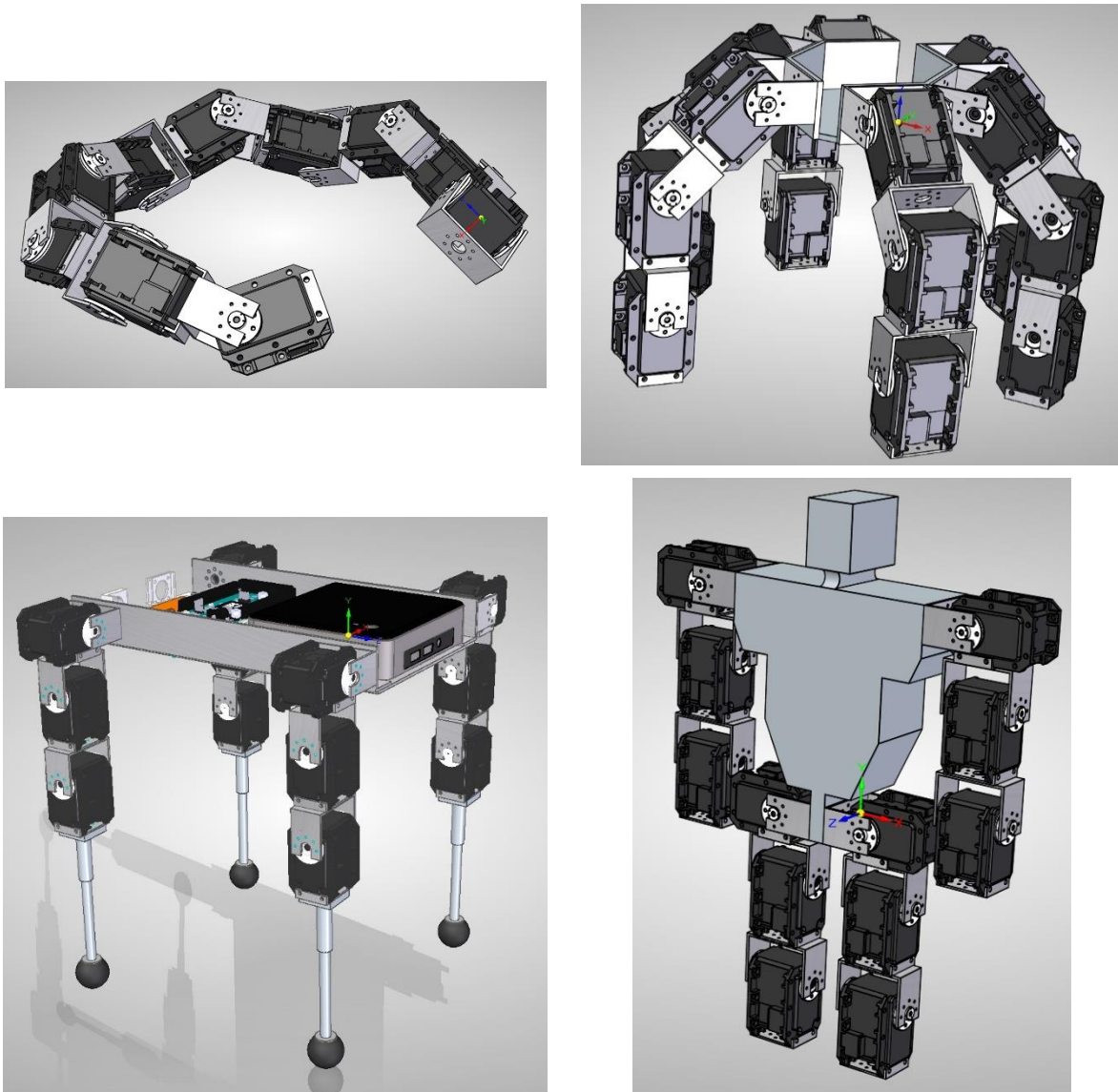


Figure 3.1: Examples of modularized robots in various configurations. Although motor components are identical, the resultant shapes and gaits are significantly different.

3.3 Modularized Robotics

The general practice in robotics is to produce, as closely as possible, a robot hardware design with closed form dynamic solutions that can be quickly integrated and used in the estimation and control of the system. Commonly used industrial and academic robots like the Puma 560 were designed to produce the simplest closed form solutions possible for kinematics and dynamics [39], at the cost of system complexity and inability to be re-specified when use conditions change.

Compared to traditional robot designs, where the design is optimized to simplify the calculation of dynamics for a particular workspace and load distribution, modular designs simplify the physical design of the robots without consideration of the effects of mechanical design on system dynamics. System dynamics for such systems use a generalized approach, considering each articulation independently of other articulations and any desired loading [40, 5] and calculating the system response as a whole.

While this increases dynamics complexity and reduces the peak computational performance over carefully tuned structures, the simplified construction and reduction of unique parts can significantly reduce the overall costs of a robotic system. With a proper control framework, the aforementioned disadvantages can be minimized and the system can be effective for locomotion research. For the modular design to be effective, it must fulfil several requirements:

- Modules must include standard physical and electrical connectors common to all modules and passive components
- Modules must have compatible digital communications protocols with pass through
- Modules must exclusively control any degrees of freedom it powers
- Modules must be independently controlled
- Large changes in inertia between joints should be avoided

The first two requirements are equivalent to general robotic platform component development procedures, creating constraints for the physical and logical connections between modules. This ensures that any module can be used to replace any other module within the robot, without a need to modify, hardware or software wise, any existing modules in the system. Modules that do not share these specifications would require modifications to be acceptable, which is generally solved by an adapter, either mechanical or electrical as necessary.

Active degree of freedom exclusivity and module independent control, ensure that the system is entirely modular, with no inter-module dependencies due to kinematic and communications specifications of the robot. In order to improve system stability, control independence is expanded to include not only that control should be independent of other modules, but also independent from external sources including the primary system controller. This in effect means that modules must include an internal controller that asynchronously controls any internal actuators to a given position or torque target without additional module external information including system IMUs, battery status, and condition of other modules. Although such MIMO control of modules are possible to develop, their control complexity would require several man-years to develop and validate for use in even a simple platform.

The overall process for the development of a modularized robot from an initial general robot concept is summarized in Figure 3.2. The core of the decision tree revolves around the selection of module hardware that can satisfy all sub-problems within the robot joint design, the design of supporting hardware, and meeting a simulation capability constraint in the form of inertia and lever arm restrictions. The minimization of module inertia ratios between consecutive modules is not a physical or control requirement, rather a necessary condition for simulation stability and optimization. Constraint and impulse based physics systems are most stable when the ratio of the inertias of connected bodies is under 32:1, with ratios closer to 1:1 increasing the accuracy of constraint resolution [41]. Additional simulation related hardware and software constraints are discussed in Section 5.4 and Section 4.3 respectively.

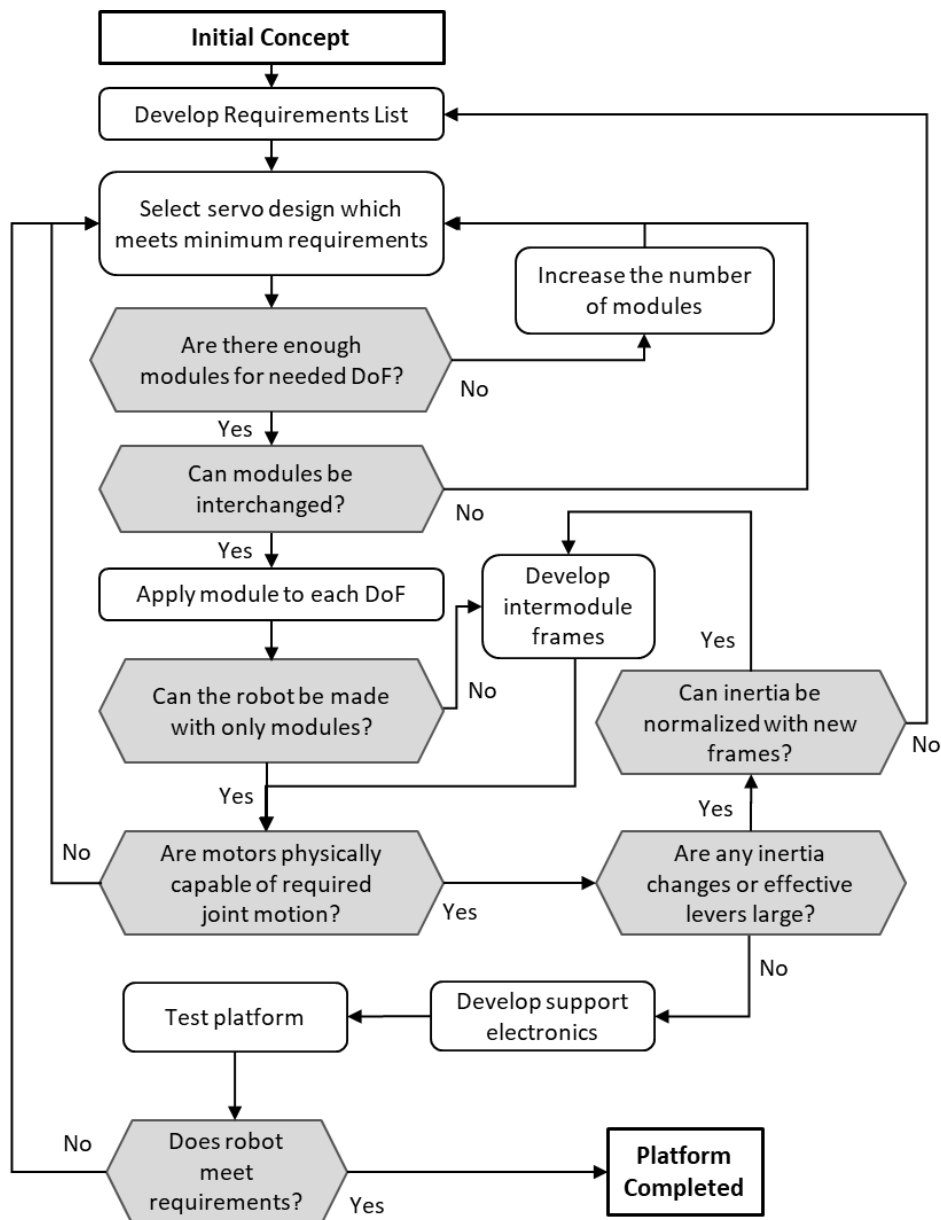


Figure 3.2: Modularization process flowchart for a typical articulated robot. A typical design will go through several iterations to produce a viable hardware platform.

This particular design process can be applied to any desired base robot shape, including both biomimetic and not biologically inspired types such as wheeled robots and

3.4 Quadruped Shape Design

Due to its unique static stability region and control manifold, the quadruped body shape allows the use of open-loop pose control, static balance, and dynamic balance gaits without significant hardware modifications required for other robot types. In particular, a cursorial mammal design was chosen in order to improve possible gait performance for a given joint velocity limit. Compared to the lizard-like setup on the left in Figure 3.3, the mammal-like configuration, shown on the right in Figure 3.3, is less stable in all locomotion methods and during static balance. The mammal-like design, however, allows

for larger forward linear motions as the majority of the leg is tangential to the instantaneous direction of the relative ground plane motion. As such, the gait efficiency is significantly higher and allows for a higher forward velocity under identical joint velocity constraints.

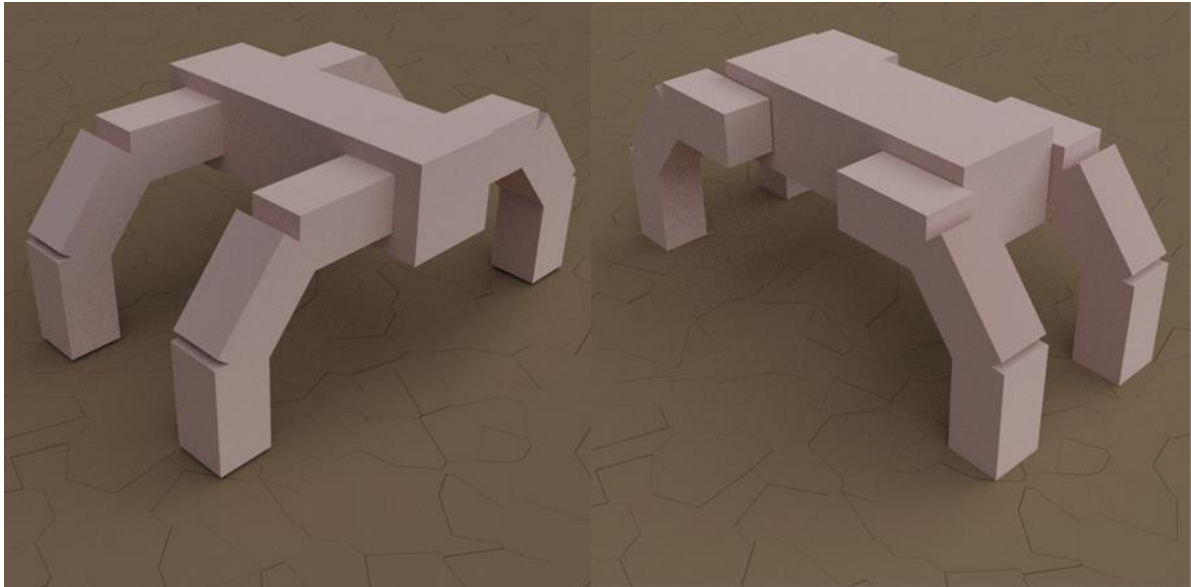


Figure 3.3: Two main quadruped configurations, lizard-like on the left and mammal-like on the right. The wider base of the lizard-like configuration provides additional stability over the mammal-like configuration.

3.4.1 System Simplification

Quadruped animal skeletons and muscles are too complex to recreate in general use robots. In general, quadruped animals have three planar rotation degrees of freedom, in the hip, knee, and ankle, and at least one additional degree of freedom at the hip that allows for rotation of the leg plane away from the sagittal plane of the animal about a line parallel to the spine, as seen in Figure 3.4. In certain cases, there may be additional, limited rotation degrees of freedom present at the hip and ankle, however these are not necessary for forward locomotion and used primarily during turning.

In addition to these rotational degrees of freedom, many cursorial mammals have a flexible spine that adds additional degrees of freedom. This flexible spine can bend in the sagittal plane to store energy and increase the overlapping area of the leg workspaces, and it can bend about the spine axis to offset the back and front leg position resting positions.

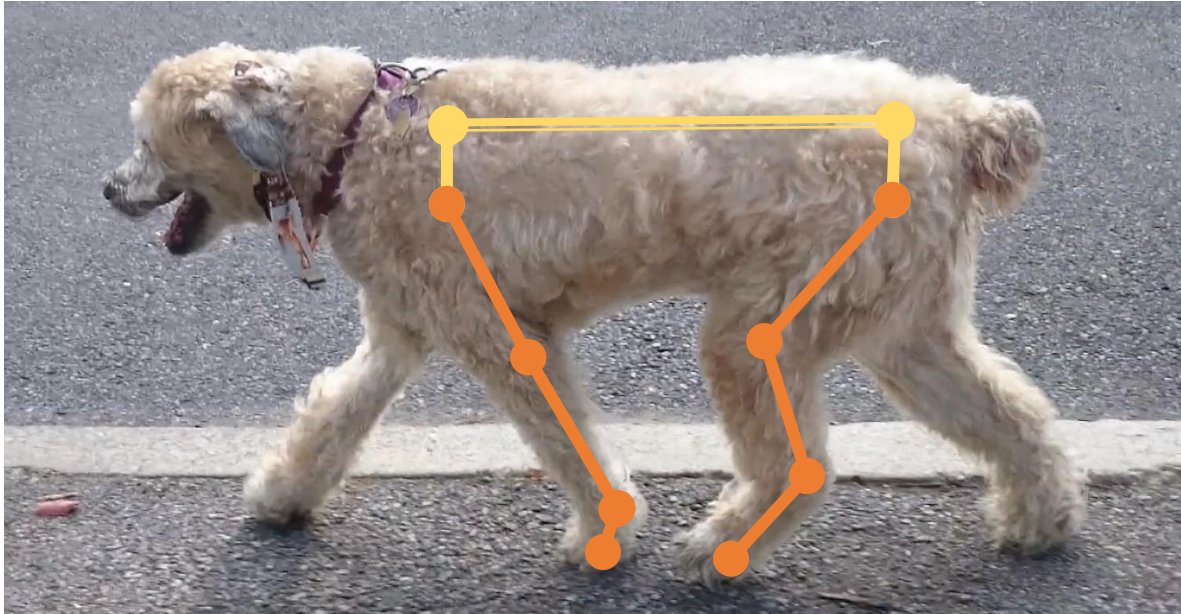


Figure 3.4: Approximate dog joint location in 2D plane. The actual joint definition for the shoulder and hip cannot be given on a strictly 2D representation

The complexity of a biological analog would require immense computational power to control, as well as strength to weight to velocity ratios of actuators that are difficult to achieve. Instead, quadruped robots generally use an approximation of a real animal, outlined in Figure 3.5, using three active degrees of freedom per leg and a rigid spine, with an optional passive fourth DOF [42] [42]. Although exceptions to this design philosophy exist, the actuation and control complexity of such systems make them ill optimized for generalized experimentation.

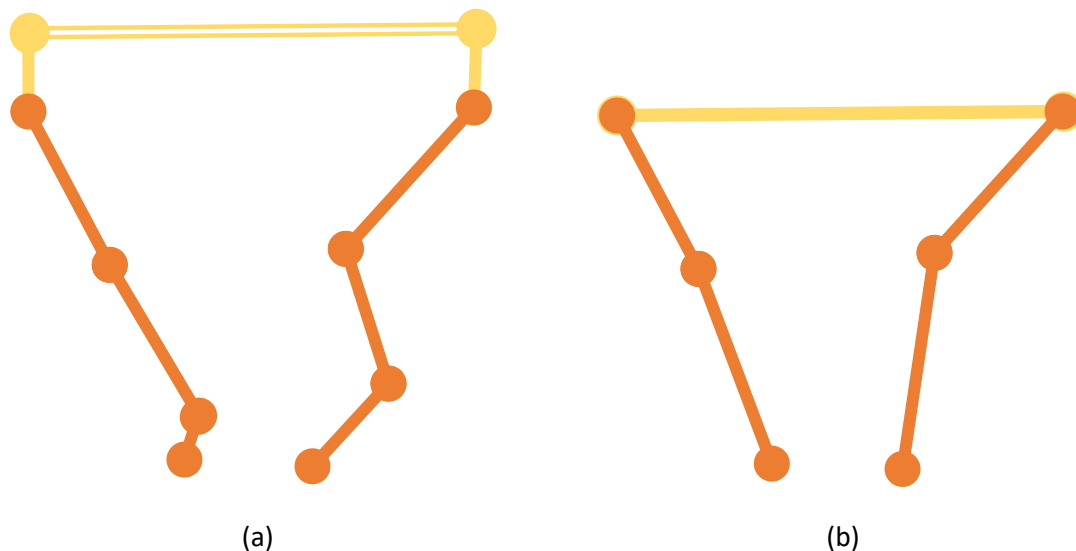


Figure 3.5: Detailed 2D joint approximation (a) with three rotation joints per leg and flexible spine (double yellow line) and simplified version (b) with two rotation joints and rigid spine (solid yellow line). Both kinematic and dynamic properties of each joint are different, but overall system dynamics is comparable.

3.4.2 Leg configuration

The configuration of the leg modules is critical in determining the overall performance and capabilities of the robot. Many of these performance metrics are related and often contradictory, acting as both

constraints on valid configurations and measures of their overall performance. In order to achieve quadruped motion, each leg is required to move both in the sagittal plane and perpendicular to the plane. As the modules are each a single degree of freedom, any leg design must include at least one motor in the plane and one perpendicular to it. Figure 3.1 shows four possible configurations that were considered during early prototyping that met the above stipulations.

Configurations (a) and (b) use 13.5% lower torque on bottom motor than (c) and (d) and up to 42% less torque on upper motor in the relaxed pose in Figure 3.6 below. As a tradeoff, configuration (a) and (b) have approximately 19% slower endpoint motion in this particular stance and is almost 100mm/s slower at full extent. This is due to the length of the individual joint components, with shorter legs reducing necessary torque but longer legs achieving higher linear velocities in the end effector.

Of the designs in Figure 3.7, configuration (b) was not considered on the grounds of leg workspace issues. With both motors are mounted in the same direction, the overall leg motion is limited to a few degrees past singularity (fully extended leg) in one direction while allowing over 180 degrees of motion in the other direction. This configuration does not provide faster potential leg movement than (c) or (d), nor does it improve joint torque loading beyond (a).

Configuration (c) was also not considered as it does not follow the design targets for the modularized components as closely as the other configurations. In particular, the overall mass of the final leg section is significantly smaller at just under 41g as compared to 114g in the other configurations. As the other modules are 87g and the main body is 370g, this would be a 47% and 11% relative mass to the modules and body respectively. Although there is a hard limit of roughly 32:1 due to precision in the numerical calculations in the simulation, mass ratios in the 2:1 range can cause issues for the foot contact models and are not conducive to low sampling rates. The remaining configurations all have a relative mass of 31% and 30% respectively, which can significantly improve simulation stability at low integration rates.

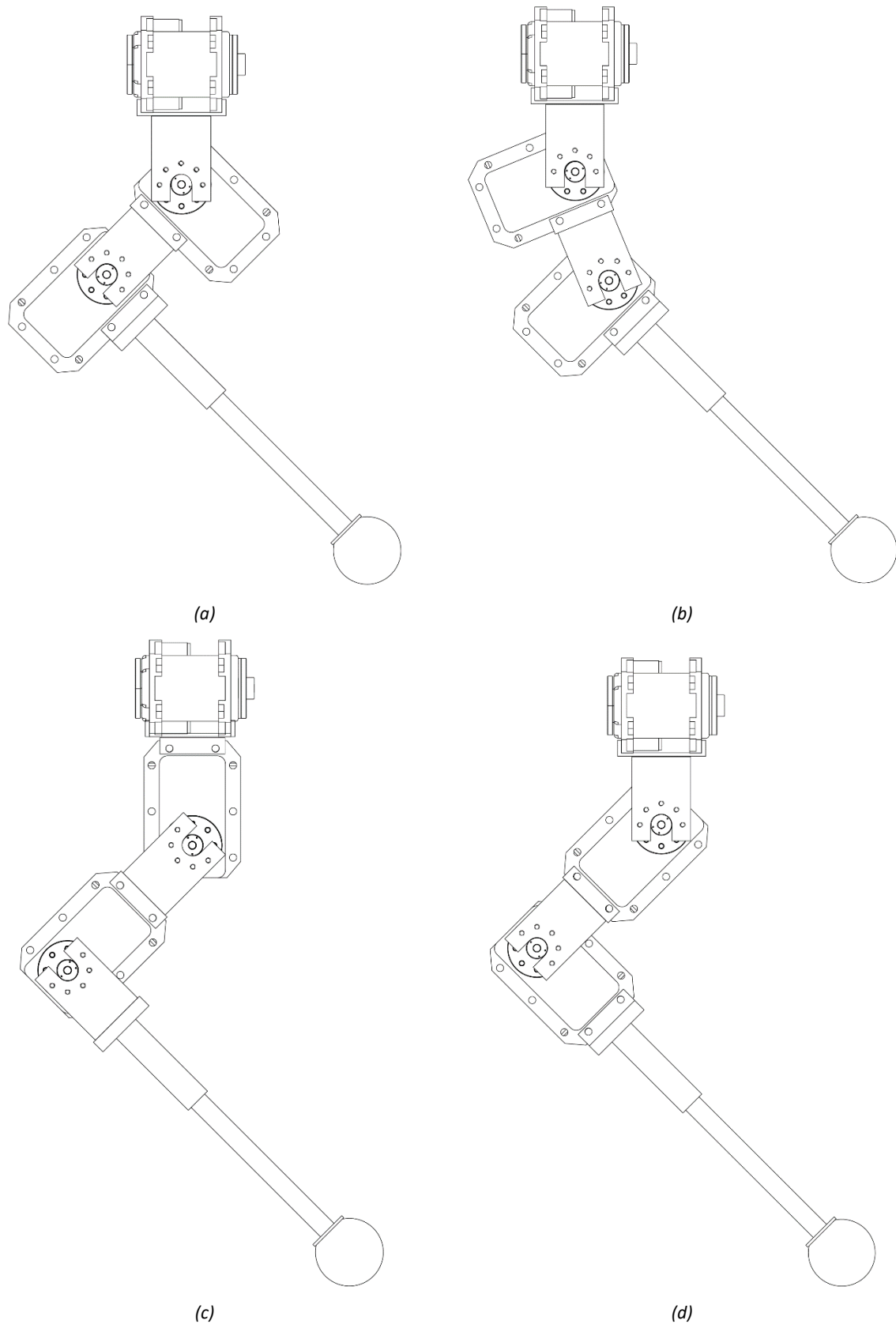


Figure 3.6: Proposed leg configurations in a non-singular pose.

Configuration (d) was used in the final construction to minimize the effects of incorrect inertia approximations, however the ability to use configuration was left available to facilitate optimizations of the system configuration in future projects.

3.4.3 Customizable platform design

The current quadruped robot design, seen in Figure 3.7, uses servomotor modules that are connected by standardized physical interfaces and requires just two hex-head tools to fully assemble and disassemble in under ten minutes. In this design, the legs can be easily replaced in the field with minimal part redundancy, allowing the robot configurations to be changed to suit different environments and locomotion methods. Legs can be either entirely symmetric, as shown, or may be asymmetric with shorter front legs and longer rear legs to mimic gaits in real animals such as felines, canines, and cervids that require a flexible spine. Compared to the most known research oriented quadrupeds [42] [43], the robot platform is significantly more compact, and can be easily transported in its fully assembled state, while maintaining a large payload compartment. In order to allow for forward compatibility with future research opportunities, the platform was developed with the ability to be entirely autonomous, when used with a miniature personal computer system such as the Intel NUC and 3D scanning system such as the Intel RealSense R200, or controlled by an attached computer using USB. The system dynamics are calculated using rigid body physics solvers, providing a reasonable tradeoff between calculation speed and accuracy for any desired configuration, including non-quadruped configurations.

Minimal tooling for assembly was a major concern in the design of the quadruped robot, both to reduce expected cost and to simplify assembly procedures. Due to the design of the servomotors used, two hex-head machine screw sizes were required for the servomotor mounting. The servomotor horn and inter-module connections were required to M2 size screws by the manufacturer, while the mounting bracket connections to the motors use M2.5. Additionally, the main body connections, leg extension base extrusion, and power connection tray were designed with M2.5 screw options. To minimize the number of parts, the remaining mounting components were specified to one of these two screw and hole sizes.

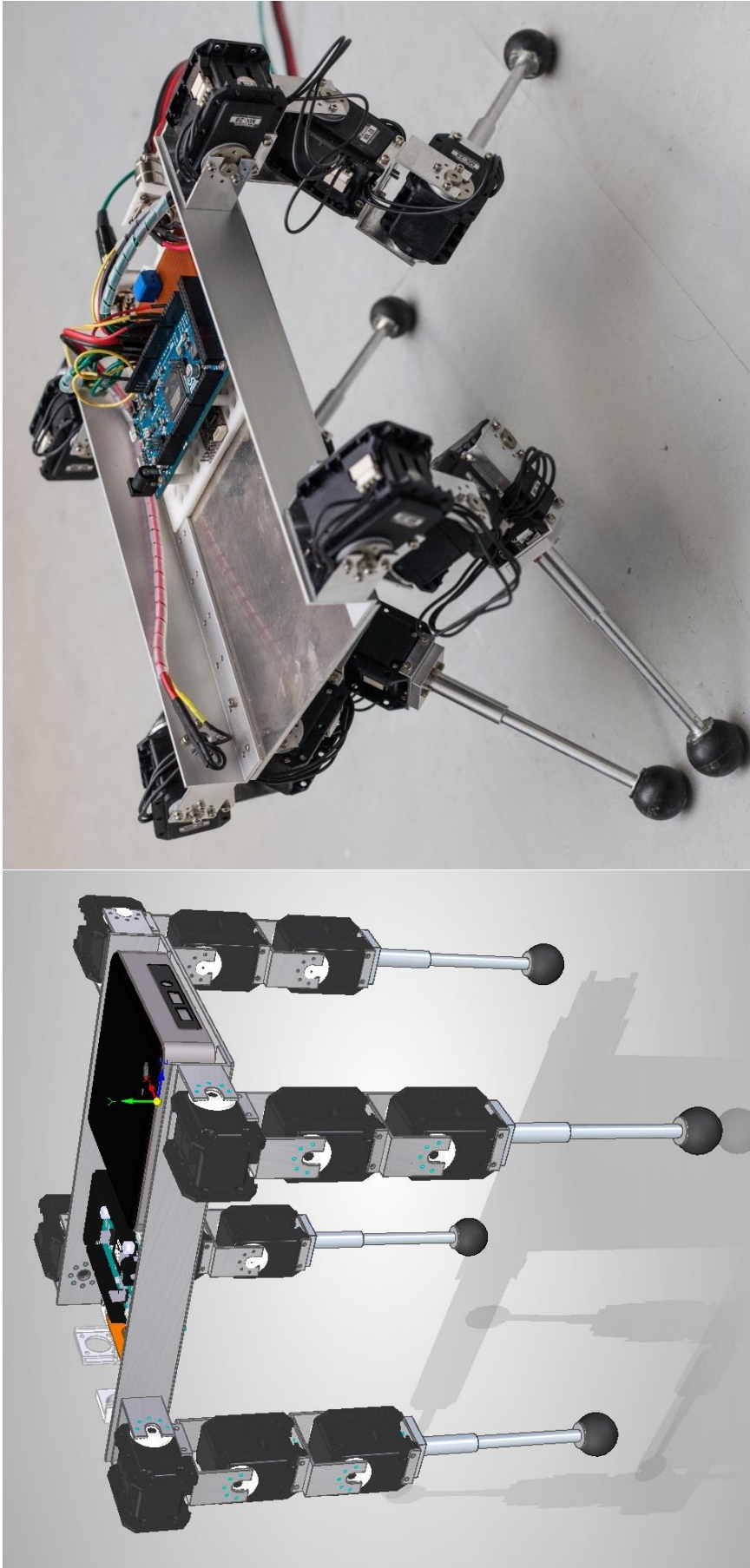


Figure 3.7: Completed robot in minimal payload state (left) and the CAD model with additional computing unit (right). The platform can be reconfigured to support a NUC computer and RGBD sensor (not shown) for semi-autonomous SLAM assisted locomotion

3.4.4 Extendable foot

In its base form, the robot platform highly stable general motions due to the low center of gravity due to fairly even mass distribution. The system stability can be modified to a certain degree by increasing the elevation of the center of gravity, either through joint redundancy to change the effective leg length to the ground or by physical extension of the leg length. Although pose changes can affect the stability of the robot, lowered mobility may result as the leg approaches its kinematic singularity at full extension. Since a certain amount of mobility is necessary for balancing purposes, a hardware solution to increase the distance from the center of gravity to the end effectors is necessary to modify the base stability of the system. Additionally, the torque requirements of the robot can change drastically depending on the leg component lengths, and as the final robot payload mass was not determined at the prototyping stage, multiple interchangeable leg lengths would be required in the event the payload mass was significantly larger than first anticipated. The final leg extension design is shown in Figure 3.8 with three possible extender lengths.

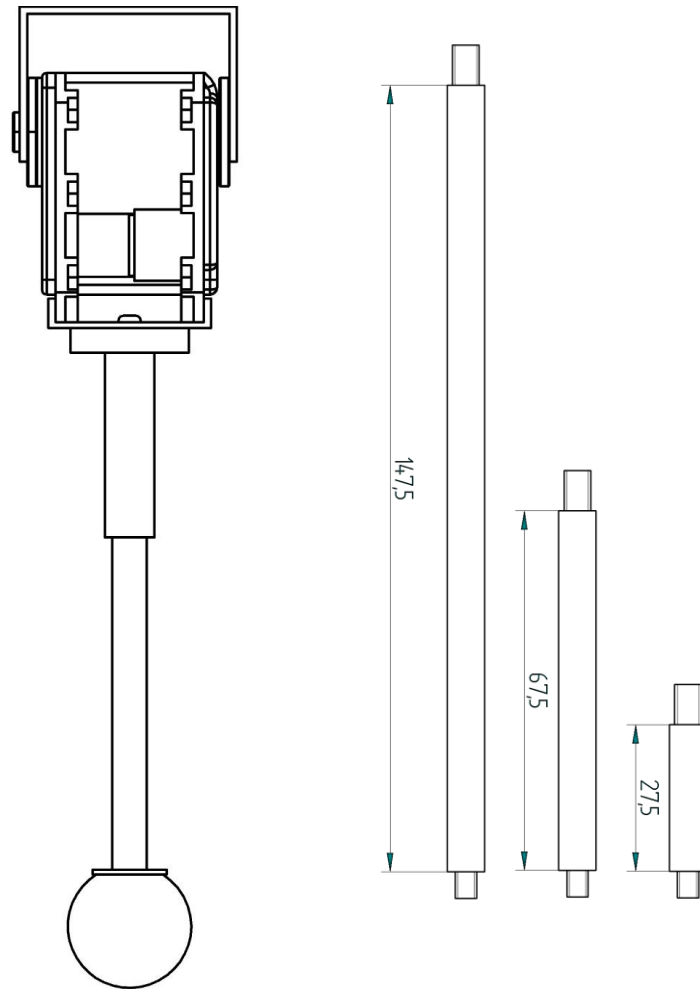


Figure 3.8: Leg end effector module with alternative leg lengths. Leg extenders use standardized connections and can be replaced without additional tools.

Due to manufacturing limitations, the extender was designed with three parts which can be separately manufactured from high strength aluminum as seen in Figure 3.9. A base of aluminum uses the standard module connector layout to provide a compatible mounting option. This base is attached to

the extension mount by a small standoff component that is secured by a machine screw, and the leg extension part is then connected to this component. A low restitution rubber foot component can be screwed into the leg extenders. The use of integral screws allows the component to easily disassemble in a matter of seconds.

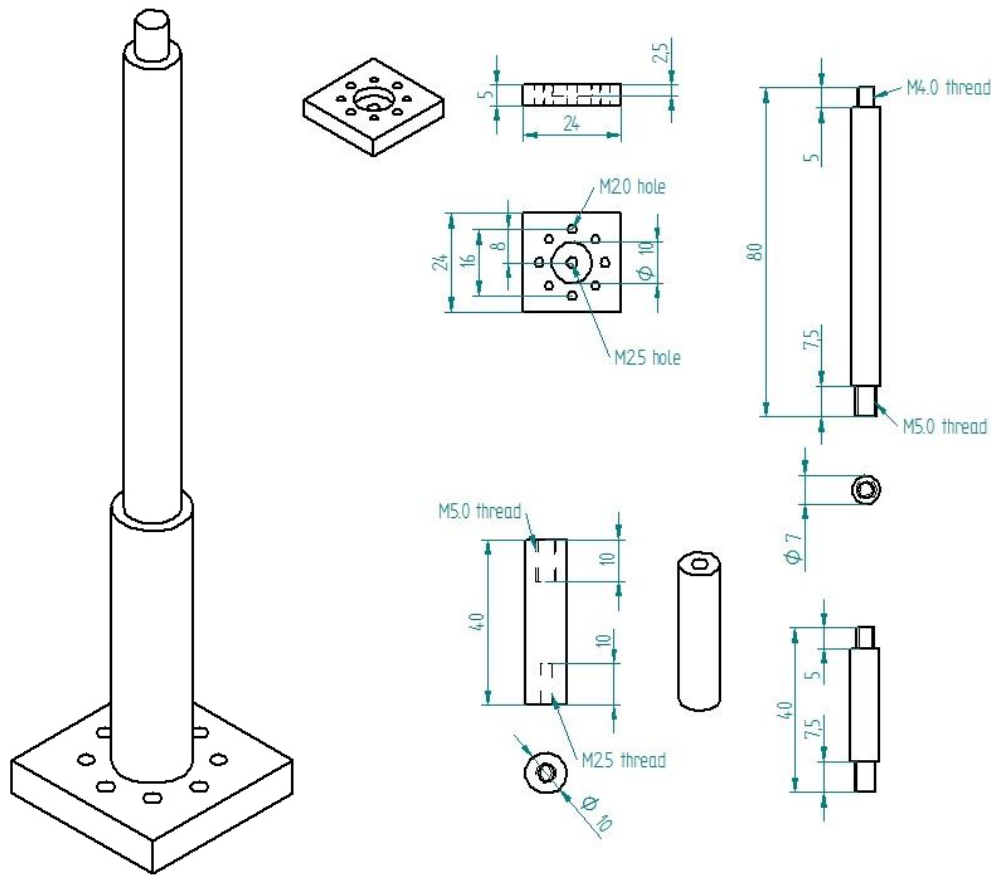


Figure 3.9: Tool-less leg extension component with optional replacement parts. Leg length can be modified without additional tools.

3.4.5 Kinematic considerations

Simplification of the system's kinematic representation can be used to improve readability of the robot structure without affecting the workspace calculations. Within the simulation framework, rigid bodies and their constraints are created independently. By separating the joint space representation from the module geometric center of rotation, we can simplify the kinematic equations for non-dynamic calculations and the simulation joint setup. As the simulation requires geometry definitions based on the center of gravity (Section 5.2), a one-time transformation from the joint space rotation center to the body center of gravity is required. The partial kinematic simplification can be seen in Figure 3.10, where rotation axis definitions have been skewed along the Z axis for the lower two motors to ensure that the X axis vectors are co-linear.

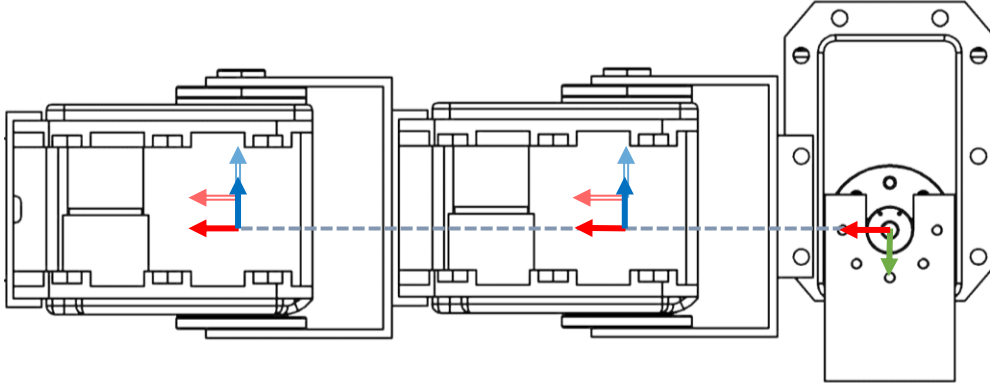


Figure 3.10: Individual leg with center of gravity and joint axis definitions. The joint axis definitions are offset from the center of gravity to simplify the kinematic model calculations. Joints are given with local X axis in red, Y axis in green, and Z axis in blue. The center of gravity is given in a lighter shade where it does not match the joint location.

We can estimate the kinematic workspace of the robot legs using joint-space limits and cartesian transformation definitions and iterating over the joint limits. While several methods exist, the most commonly used is the Denavit-Hartenberg (DH) and modified DH method, where chain transformations are used to define the joint space to Cartesian coordinate transfer. Using the modified subscript DH parameter definition method from Craig [44], we can get the necessary parameters as seen in Table 3.1 from the CAD definition files for the robot parts and the simplification from Figure 3.11.

Table 3.1: Denavit-Hartenberg parameters for the quadruped robot. Foot end effector is calculated to the center of the ball.

Link	i	a	α	d	θ
Hip	1	0	0	0	θ_1
Knee	2	44.5	-90°	0	θ_2
Ankle	3	65.3	0	0	θ_3
Foot	4	161.3	0	3.75	0

These parameters can then be used to build the joint space to local Cartesian transformations in Eq. 3-1 and local Cartesian to world Cartesian transformation chain in Eq. 3-2 [44]. As the coordinate transformations are fixed except for the θ_i parameter, it is possible to expand the transformations into a differentiable closed form at each joint, however with modern computing speed this does not provide significant benefits over the simpler chain transformations.

$${}^{i-1}_iT = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 3-1$$

$${}^0_N T = {}^0_1 T {}^1_2 T {}^2_3 T \dots {}^{N-1}_N T \quad 3-2$$

Iterating the above system over the joint space for each module and recording the resultant endpoint location in three-dimensional space, we can visualize the effective workspace of the robot end effector. As seen in Figures Figure 3.11, Figure 3.12, and Figure 3.13, the end effector of each leg has a sufficiently large workspace to allow for linear end effector motion in a significant portion of the space. However, considerations for possible internal collisions within the overlapping regions must be taken into account due to the significant redundancy in the workspaces of adjacent legs. Additionally, certain sections of the workspace are inaccessible in normal conditions as collision with other legs and the center body prevent meaningful access to the remaining area.

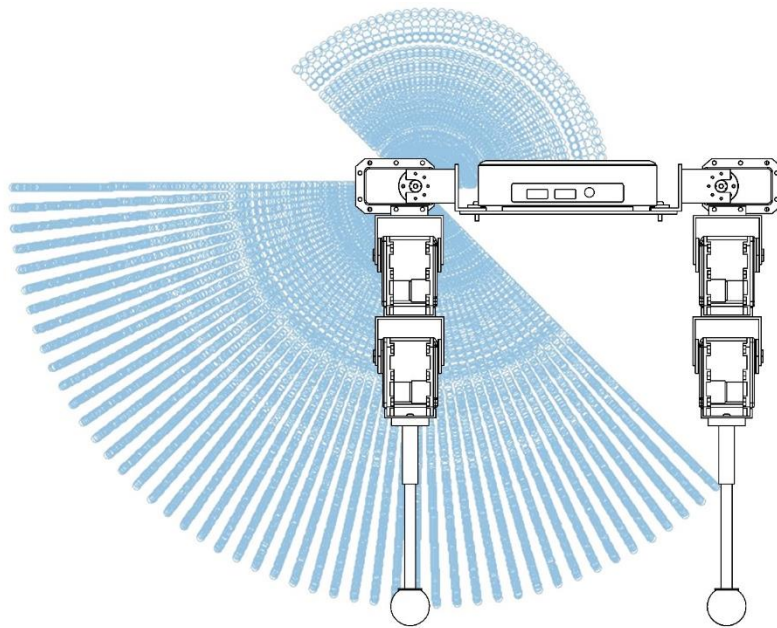


Figure 3.11: Front side view of the approximate workspace of the quadruped robot with medium length leg extenders installed. Actual workspace in this implementation is smaller due to internal collisions with the main body.

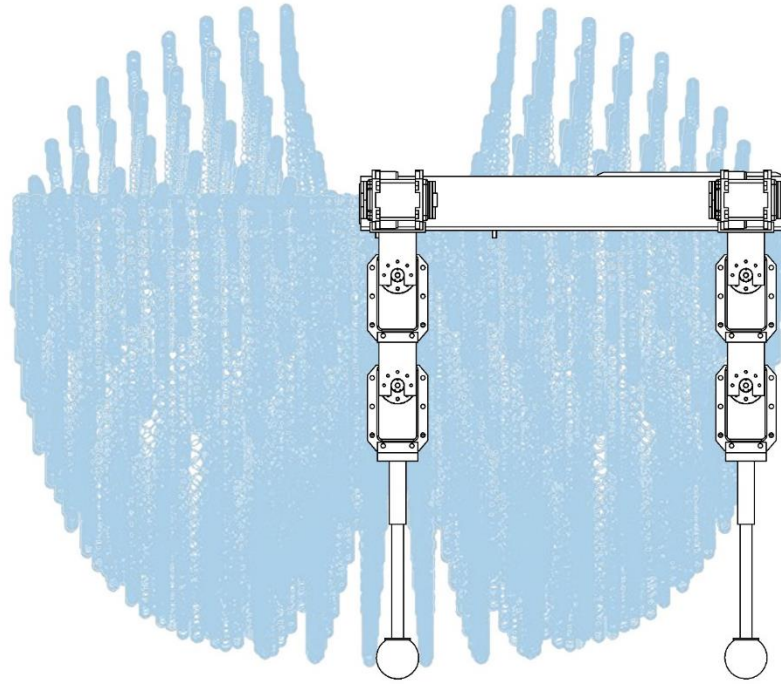


Figure 3.12: Right side view of the approximate workspace of the quadruped robot with medium length leg extenders installed.

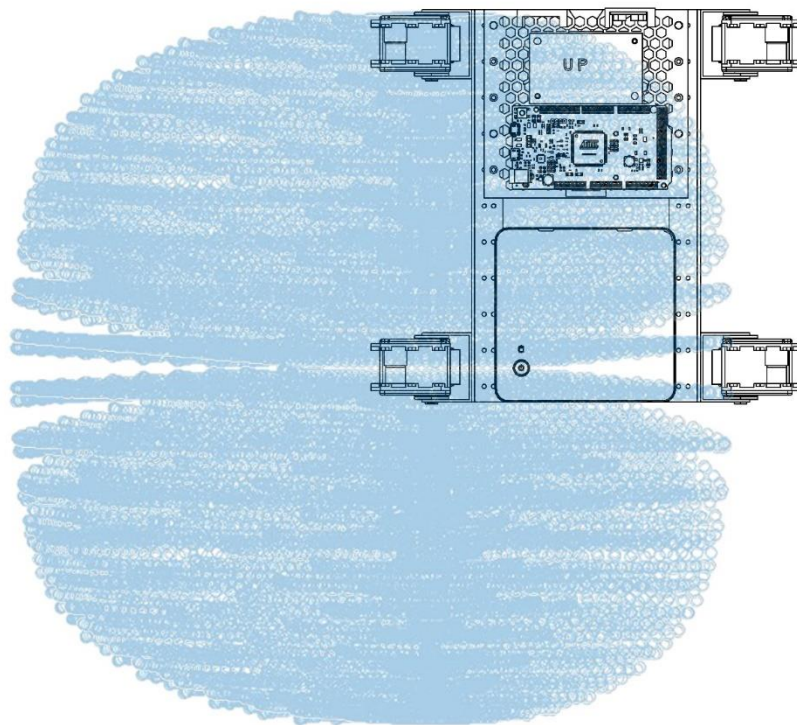


Figure 3.13: Top view of the quadruped robot, including the power connection plate, prototyping board for low level sensors, and optional high-powered miniature computer. Necessary cables are not included in the diagram.

3.5 Motor Modules

The primary component of modules used in the modular quadruped robot and other test platforms are highly integrated servomotors and their connecting interfaces. The performance of the robot platform in modular robotics is dependent on the individual performance of each module, requiring high space and weight efficiency to achieve required performance levels. Several motors were considered as the base component of the modules including commercially available and custom-built servomotors. Three of the commercially available servomotors considered are outlined in Table 3.2 and their features compared to a custom servomotor system.

Table 3.2: Comparison of servomotors used in robot joint. The AX and MX series motors represent low and medium quality “smart” servomotors and the HS series motor represents “hobby” style analog servomotors.

Motor Type	Dynamixel AX-12A	Dynamixel MX-28T	Hitec HS-925MG	Direct-Current motor
<i>Communication type</i>	Digital packet	Digital packet	Pulse Position	Duty cycle
<i>Internal controller</i>	Piecewise proportional	Saturated PID	Proportional	None
<i>External Control frequency</i>	~11 kHz per port *	~33 kHz per port *	50Hz	**
<i>Actuation precision</i>	10bit (0.29°)	12bit (0.088°)	<8bit equivalent (>1°)	Controller and encoder dependent
<i>Readable information</i>	Position, velocity, load	Position, velocity, load	None	Load (via external sensors)
<i>Additional Required Hardware</i>	None	None	None	Position encoder, Controller
<i>Stall Torque</i>	1.5Nm	2.3Nm	0.755Nm	**
<i>Maximum Velocity</i>	6.18rad/s	5.76rad/s	9.52rad/s	**

* THE FIGURES ARE GIVEN USING MAXIMUM POSSIBLE COMMUNICATIONS BAUD RATE WITH MINIMUM SIZE CONTROL PACKETS. ACTUAL RESULTS ARE LIMITED BY UART CONNECTION SPECIFICATIONS IN PRACTICE.

** DC MOTOR SPECIFICATIONS DEPEND ON MOTOR, CONTROLLER, AND GEAR RATIO

For the servomotor type to be viable for use as the core component of a module compatible with the simulation and control framework, the following conditions must be met:

- Motors should be rotational in nature with only one controllable axis
- Motors are digitally controllable with a multi-drop control signal
- Motor controllers must provide a minimum of angle feedback
- Motor simulation model is provided or simple to calculate

The primary module motor, shown in Figure 3.14, is a Robotis Dynamixel MX-28T digital servomotor. As Robotis and other servomotor manufacturers do not disclose the properties of the DC motors used their servomotors, examination of the internal components was used to determine the specifications used in the electro-mechanical motor model, which can be found in Table 3.3. The gear ratio and the high-level controller algorithm were provided by Robotis with a relatively high 193:1 gear ratio using coated metal gears. This particular model was determined to contain a 12V, 4W RE-max 17 brushed DC motor with electro-mechanical properties available from Maxon Motors. These properties are required for electro-mechanical simulations discussed in Section 6.3.

Table 3.3: Reverse engineered motor parameters required for the simulation of servomotors using an electro-mechanical model.

Motor Property	Value
<i>Ra</i>	8.29 Ω
<i>Ks</i>	889 rpm/V
<i>La</i>	0.205mH
<i>Vmax</i>	12V
<i>Kt</i>	10.7 mNm/A
<i>Gear ratio</i>	193:1

On the control side, the servomotor output shaft includes a contactless magnetic absolute rotation encoder connected to a 12bit DAC channel in the 32bit ARMv7 Cortex-M3 based STM32-F103 microprocessor. This positioning and type of the encoder ensures that the exact position of the output shaft is known regardless of gear health and backlash characteristics, providing a maximum resolution of 0.088° per step throughout the 360° rotation arc. Control accuracy is dependent on the PID parameters specified in the motor EEPROM, with a minimum steady state error of 0.088°.

The STM32 microprocessor uses additional motor state information including input voltage, motor current, and servomotor temperature to control the RE-max 17 to achieve a desired goal. The motor implements a PID controller, however the specific controller output to motor driving voltage transfer function is unknown. Discussion on the estimation methods for this unknown controller will be introduced in Sections 7.2. The software control structure of these servomotors can be found in section 4.5.1, and additional methods unique to this particular model can be found in Section 4.8.1.

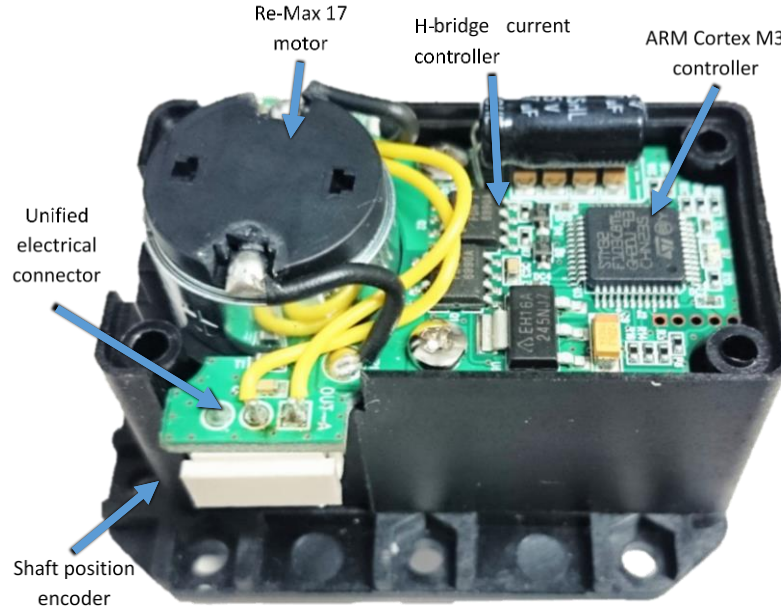


Figure 3.14: Module servomotor component view shows the main components for driving and controlling the internal DC motor. The position sensor is attached to the output shaft and cannot be shown without irreversible damage to the function of the motor.

Motor performance is of considerable concern, particularly the response under significant load, as the robot uses directly driven joints within the legs. To properly model the system for use in simulation based control, certain features must be known at an electromechanical level. In particular, the type of motor drive and holding characteristics can significantly impact the step response and tracking performance of the system. The load-velocity curve is an important tool for the verification and specification of the robot motors, showing possible directional friction irregularities, as well as providing an expected value for certain electrical characteristics of the motor. The internal load and position sensors of the MX-28T motor were compared using data was collected from the first 100ms during a $+\frac{\pi}{4}$ radian step input for various loadings using the SARM system discussed in Section 3.7.1. This internal load can be converted to actual current in accordance with a steady state approximation of the electromechanical model equation 6-1 outlined in Section 6.3,

$$i = \frac{1}{R_a} (V_{target} - K_s \dot{\theta}_m) \quad 3-3$$

which in turns additionally allows a calculation of the motor load, the effective PWM variable L , as

$$(V_{target} - K_s \dot{\theta}_m) = V_{max} L \quad 3-4$$

This in turn allows for the calculation of the current in terms of the effective PWM load through substitution, resulting in

$$i = \frac{V_{max} L}{R_a} \quad 3-5$$

The default operation of these servomotors saturates the motor driving voltage V_{target} to the maximum value when target position is changed by a large enough value, allowing the motor back-emf,

the velocity dependent voltage generation K_s , to be verified as a function of the velocity $\dot{\theta}_m$ motor line voltage V_{max} alone. Figure 3.15 demonstrates this relationship with high linearity, allowing a maximum speed forward drive speed to be calculated. A maximum speed is achieved when the effective PWM is zero, resulting in a value of 63RPM for the data represented in Figure 3.15. Although this is in the 50-67RPM range dictated by the manufacturer, non-electrical limitations including static and dynamic friction can further limit the motor performance from achieving the theoretical maximum. The actual no load data in this figure shows a maximum velocity of 51.7RPM with 17.8% Effective PWM, which combined with Eq. 3-3 and 6-2 gives a rough estimate of the total friction upon the system.

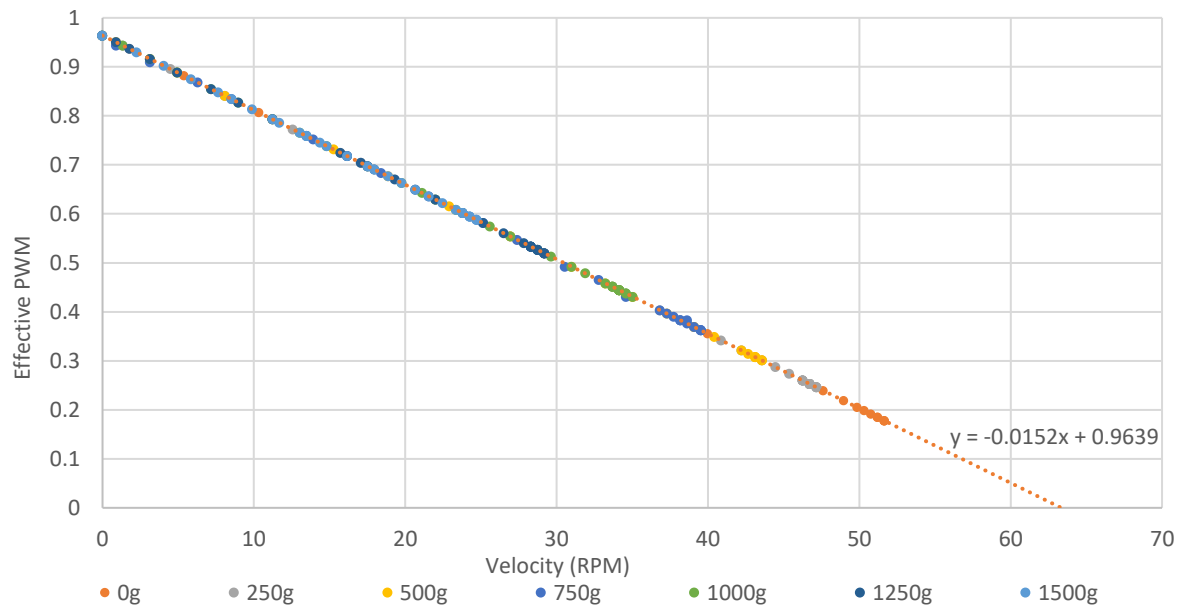


Figure 3.15: Driving electrical load (PWM) during the first 100ms of a positive step input as a function of motor velocity for an MX-28T smart servomotor. Values range from 0 (no power to motors) to 1 (100% power supplied)

As the instantaneous output torque is equivalent to the instantaneous current times the motor K_t value when the rotor inertia is negligible, this graph additionally serves as a normalized torque-velocity function due to the dependence of current—and therefore torque—on the PWM load alone. This torque-velocity curve in turn dictates the primary motor dynamics when exposed to an external load, as is the case in Figure 3.16 where the maximum velocity decreases as the external load increases. The invariability of the load-velocity ratio with respect to passive external loadings shown in Figure 3.15 can be used to determine possible simplifications to the electromechanical model of the servomotor. This is also a necessary requirement for the generalized DC motor equations to be solvable independent from the external dynamics, such as load direction.

When the external loading is dynamic, as in the case of mass with non-negligible inertia being moved under the effect of gravity, the motor may occasionally perform significantly outside the torque-velocity curve in Figure 3.15 as can be seen in Figure 3.16 (b). Due to external forces, the motor experiences speeds faster than maximum forward drivable joint velocity. Rather than achieving a lower maximum velocity than the no load condition as in Figure 3.16 (a), the maximum velocity in the

overdriven case is higher than the no-load velocity, signifying that motor is operating outside the bounds of the curve in Figure 3.15. During this time negative torque generation occurs, as expected by the torque velocity curve beyond the maximum drivable velocity when effects like back-EMF are computed. Further discussion of this characteristic and its impact to simulation is discussed in Section 6.

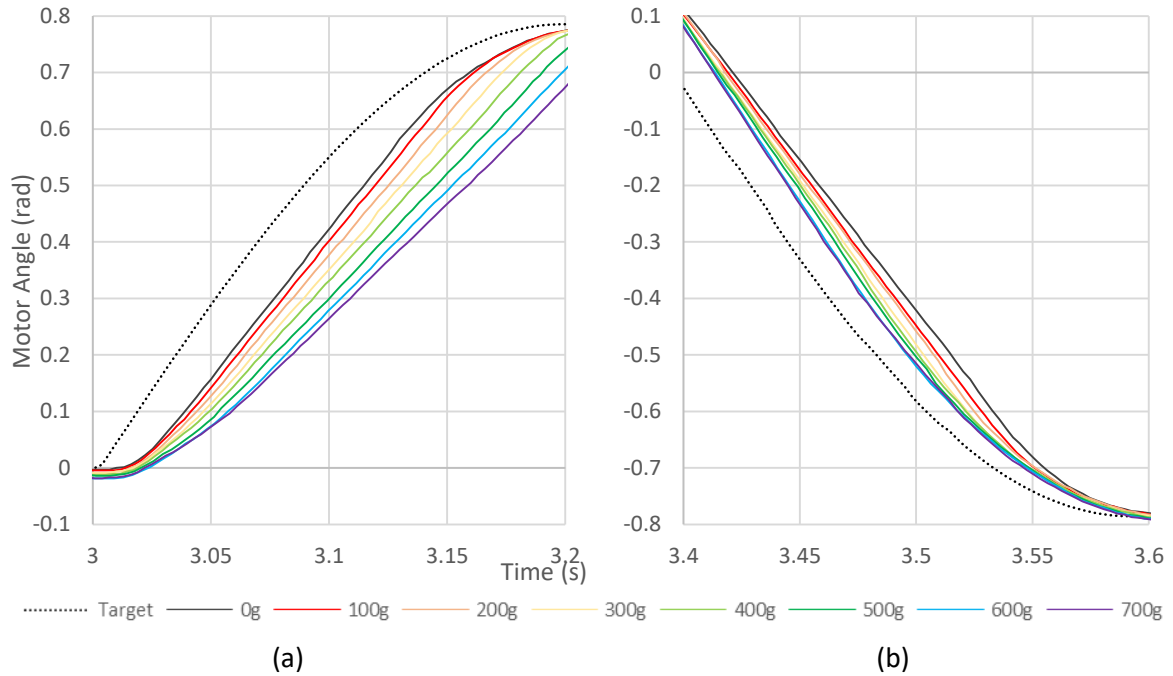


Figure 3.16: Servomotor dynamic response to an $\frac{4}{\pi} \text{ Hz} \pm \frac{\pi}{4}$ sine wave under various loadings. Maximum motor speed order on the up-stroke (left) is reversed during the down-stroke (right) due to external forces from gravity.

3.6 Additional Components

In addition to the core components, several other parts are necessary for the effective operation of the robot platform in a test environment. The specific models and designs of these components do not affect overall performance or function equivalently as long as the parts meet the minimum required specifications.

3.6.1 Sensor and power tray

The simplified all-metal construction of the main body makes the inclusion of additional electrical components difficult due to a lack of mounting points and electrically insulated space. In order to improve sensor location stability while isolating any electrical circuit boards, a sensor specific tray was 3D printed with high quality ABS plastic. The tray, seen in Figure 3.17, includes several key features that improve overall robot functionality. A CAL-30-454 4-pin male CB connector is mountable on the rear-left and is compatible with a CES-31-1004 4-pin female CB connector to provide a partially weather sealed connection with at least 15A current carry capability. The rear-right houses a mini-XLR four pin connector that allows up to three signal wires and one shared ground with quick release capability.

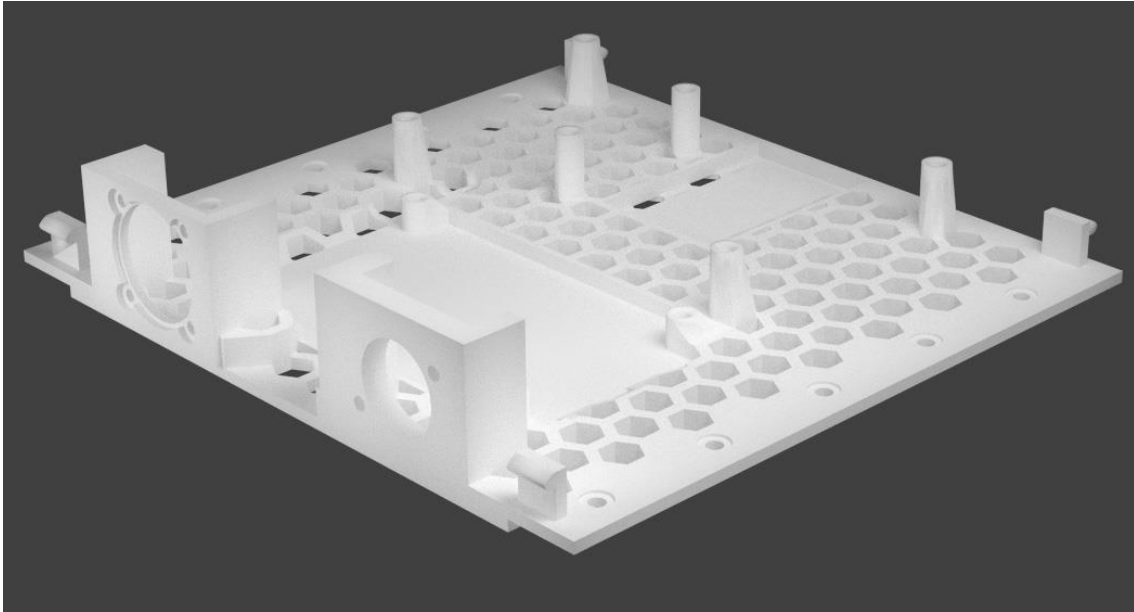


Figure 3.17: Reinforced power and sensor tray. The “back” of the tray is shown on the bottom left with the “front” in the top right of the image. Additional clasp options for a cover were not used in the initial print.

In addition to the external connections, the tray includes three mounting areas for power distribution and sensor circuit boards. The rear most mount fixes the power and control distribution board, with additional cutout sections allowing for battery terminal and NUC power connections. The elevated standoffs support a microprocessor board used to facilitate communications with additional sensors. Below the standoffs is a contact fit cutout for an ArduIMU orientation sensor board, with a significant distance between the sensor and high-power electronics to reduce interference with the integrated magnetometer.

Although the board is complex in shape, it has well defined inertial properties that are trivially accessible in the CAD files of the part. In practice however, a simplified box approximation with the same total mass can be used for the mount and all additional electrical hardware including the IMU and accessory processor board. The effect of deviations from a box approximation on the quadruped base module is negligible due to the very large difference in inertia between the metal components and the lightweight ABS and fiberglass parts.

3.6.2 Power supply

As the weight and safety of high current batteries can be a concern, an external power supply can be used with the robot platform. The primary power supply available for the platform was an ETA Electrical Industry Co. ESS150-12 DC power supply, which provides up to 13 amperes at $12V \pm 1.2V$ from any 86V-260V outlet. The output voltage setting for the power supply was set to 11.1V rather than the default 12V in order to ensure electrical compatibility with internal batteries voltages and minimize calibration errors when using lithium polymer (Li-Po) batteries. A CES-31-1004 4-pin female CB connector rated at greater than 7.5A per pin is hard wired into the power supply such that two pins each are used for current and ground to provide a safe connection to the robot. Although the transient response is unavailable from the manufacture, the power supply voltage for quadruped in motion is stable due to peak current timing offsets as the four legs generally do not exhibit torque changes at the same time.

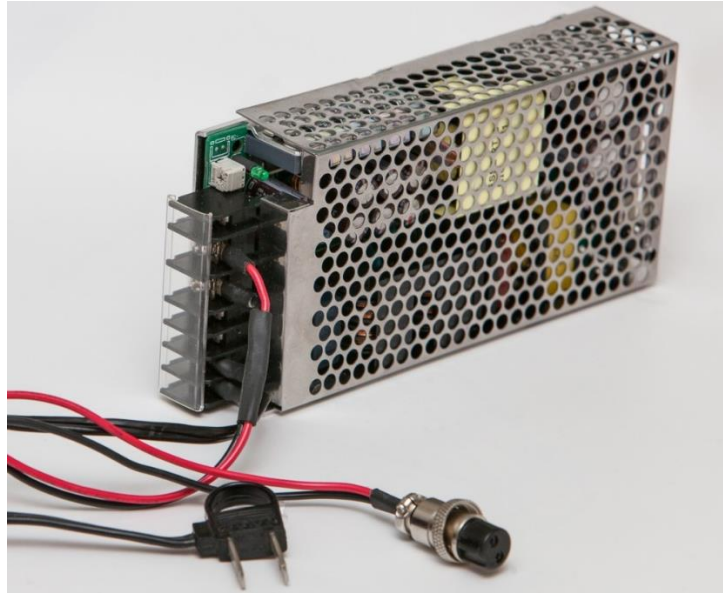


Figure 3.18: External power supply for the robot platform with weather resistant robot power connector.

3.7 Alternative designs and test platforms

In addition to the primary quadruped platform prototype, several test platforms were developed for calibration and proof of concept test experiments, including motor simulation, discussed in Section 5.4, and identification, introduced in Section 7.2.

3.7.1 SARM- the Single Actuator Response Mechanism

For the purposes of servomotor communications and control testing and simulation optimization, a test platform utilizing MX-28T and AX-12A servomotors was designed using the existing quadruped robot components. The test platform is comprised of one Dynamixel digital servomotor and connecting frames used in the quadruped robot. The single actuator response mechanism (SARM) holds a load at a distance of 56mm from the rotation axis, using the motor module itself as the primary supporting frame to increase the minimum inertial load of the platform. This is an important feature, as the platform, seen in Figure 3.19 and Figure 3.20, exhibits simulation instability under the condition of low loads with large time steps even with this increased inertia, while not experiencing real-world instability in experimental tests.

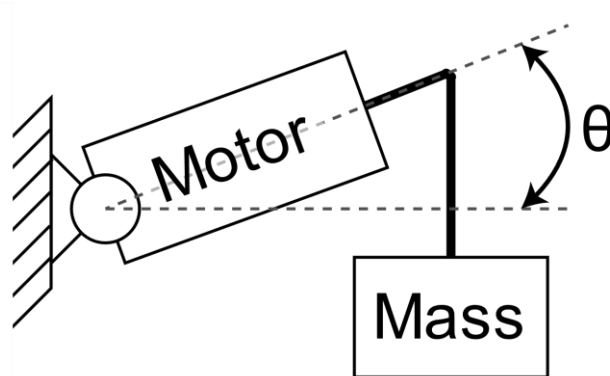


Figure 3.19: Simplified schematic of the test apparatus. The SARM test platform minimizes errors due to inertia approximations and support frame deflections.

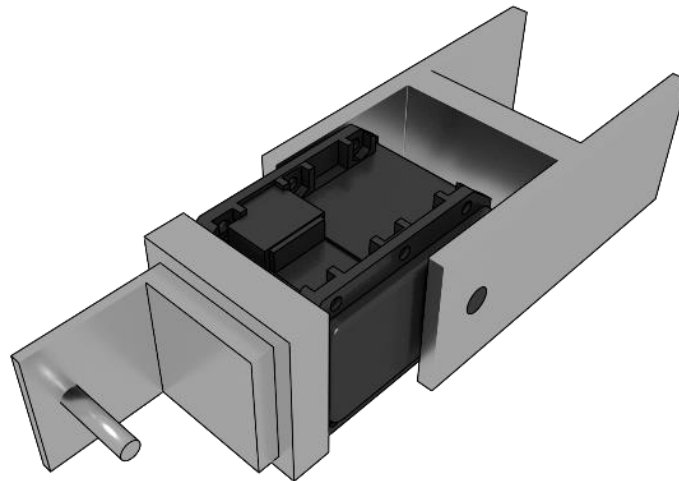


Figure 3.20: Rendered simplified CAD representation of the SARM test apparatus in the experimental orientation. Load mass, clamping mechanism, and motor control cable are not shown.

The well-defined inertia characteristics of the SARM unit and the single active DoF simplifies modeling of the system, allowing motor parameters to be identified without significant contributions from the external system dynamics and control. One limitation of the implemented system however is the use of a cable based load mass connection rather than a ridged link, which may induce effective forces which are different than in a rigid system particularly at high speed.

This SARM system can be run with a configurable input voltage, however for all experiments conducted with this system the voltage was calibrated to 11.1V to ensure that results are transferable. As a single motor is used, communications and data recording are conducted at the highest possible rates, generally in the range of 300Hz through the software framework discussed in Chapter 4.

3.7.2 Alternative module specification designs

In addition to the external power option, the robot may be powered by 3-cell Li-Po batteries operating at a nominal voltage of 11.1V, either connected through the external power connector or by internal mounting. When paired with wireless transmission of data from the robot, this allows the robot platform to move without tethers, which can improve simulation accuracy when used in the field.

Although system operational time may be significantly limited and overall performance decreased with internal batteries, the increase in stability during transient loads may allow an overall increase in performance.

Although the primary robot specifications are designed for MX-28 servomotors, the system can be used with Dynamixel AX-12A servomotors at reduced cost and performance. The aluminum frames can be replaced with Dynamixel FP-04 and FP-03 plastic frames standard in the Robotis Bioloid system, which are commercially available. The overall dimension changes with this substitution are minimal, and are unlikely to cause significant issues control issues with applied gaits and balance methods after minor tuning. However, the lowered performance of the AX-12A servomotors is likely to decrease gait performance, fine position control of the leg end effectors, and motor data acquisition rates, requiring a larger reliance on simulated systems.

3.8 Summary

A modular design based on hardware shared with earlier work in biomimetic locomotion was developed to improve the capabilities of a modular system and the control framework it requires. Commercially available components, when connected by specially designed frames, allow the overall system cost to be reduced to orders of magnitude less than many of the platforms currently used for research. The reduction in cost does provide some challenges, including a reduction of sensory information and reduced quality of control in servomotors, which can in part be mitigated by more advanced software frameworks to improve servo efficiency and reduce the need for additional sensors.

4 Robot Control Framework

4.1 Introduction

As with any automated system, in order to create meaningful uses for robotic hardware, a control structure which actuates the robot to further a specific goal is required. Simply constructed multi-module robots can often be directly controlled by a monolithic control program that includes all the components necessary for hardware communications and input command parsing, especially when precise control and sensor feedback is not essential. One such system has been used to move highly articulated snake-like robots to prescribed poses in research leading up to the current system [36]. However, the complexity of such an approach increases exponentially as additional functionality is added, particularly when sensor feedback and advanced control algorithms are required for robots with hardware sensor limitations. Even with a relatively simple robotic system like the twelve-joint quadruped robot a monolithic, synchronous control program would be inefficient and could result in control failure if the hardware specifications changed or novel control algorithms are needed. As such, a robust control framework which can operate concurrently with the quadruped hardware as well as to allow the control of any similarly designed and actuated system was required.

The non-trivial control of complex robots requires a well-structured communications, management and control framework to improve the flow of control development and allow for system hardware and software portability. Although additional functionality is often necessary to facilitate implementation, the overarching requirements of such a framework include:

- Aggregating, recording, and filtering acquired data
- Simplifying the robot control for the end user
- Translating software commands to hardware responses
- Abstracting and managing communications between the hardware, controller, and external sources

Open source frameworks that meet many of these needs have been developed in the past, including ROS, the Robot Operating System developed by Stanford Artificial Intelligence Laboratory and Willow Garage [45], and MIRA, the Middleware for Robotic Applications [46]. As a message passing based systems, both frameworks focus on data collection and distribution, using techniques that allow for multicasting and anonymous operations to manage tens to thousands of interconnected but ultimately independent devices that can comprise a robot. In the case of ROS, a system of asynchronous publishers and subscribers manage the transmission of information that is publicly accessible, while also allowing services and clients that serve as an equivalent to a standard function call in most programming languages. Although this allows for distributed systems to easily aggregate information

based on availability rather than need, it adds a layer of complexity and significant performance overhead when used on tightly integrated systems, particularly in the creation and transfer of broadcast data, which is required to be managed by a single node. Additionally, existing framework require the creation of individual sub-components such as simulation and motor controls when the devices are not available in the repository of known robots, which at the time of the development of the quadruped robot platform were not available. For the purposes of our simulation integrated control methods, the framework provides little benefit as the vast majority of necessary functionality would be outside the scope of existing ROS implementations and plugins. Additionally, systems like ROS are built with single threaded processes in mind limiting the number of concurrent changes within one object.

Although development of the necessary components in the context of ROS, MIRA, or other existing middleware was an option, a more streamlined framework that included tighter integration with simulations at all levels was chosen. In this chapter, a light weight and low level asynchronous framework to simplify development of quadruped control methods and improve system performance is proposed. This framework does not seek to replace or expand on competing products like ROS, rather it is focused on providing a compact but still modular method to integrate hardware and simulation sources with the robot control methods. One key distinction in the implementation is the ability to asynchronously address key modules, improving performance when the accessing components run independently and may induce collisions. Integration into such middleware platforms is possible by adding applicable functionality to the framework components, however this is left as an exercise for future research.

4.2 Module interconnectivity

The driving concept of the framework, keeping in line with the modularized hardware platform discussed in Section 3.3, was modularity of components with abstraction of the internal workings. As such, the framework features only a few core components, some of which are optional in certain use cases with the exception of the necessary motor functions that describe a robot. This is in line with how many other frameworks, including ROS [45], MIRA [46], USARSim [25], and Webots [26], use nodes hierarchies for control abstraction, exposing only the input and output surface of components rather than their internal workings.

An additional design choice which differentiates this approach from ROS type systems is the inversion of control based pattern based data sharing methodology as opposed to the publisher-subscriber model. Rather than publicly announcing state changes through a subscriber list, the components in the framework are generally passive, idling until an outside source, whether a hardware interrupt or a timer, calls upon functionality of that component. These passive components allow externally driven changes to occur concurrently in many cases, so long as the modified data set is not concurrently accessed. The fully asynchronous implementation contrasts with the callback queuing system of ROS, where callbacks are sequenced during a `ros::spin()` command exclusively. This allows hardware response based and simulation based information to be quickly assimilated and closed-loop control responses to include these newest values if necessary.

As such, unlike the completely anonymous node method of ROS where the type of data and ID of the message are not dependent on the original service of the node or even the node of origin [45], component type in this framework fit specific functions and are not fully interchangeable outside their original role. These specifications of module function interoperability limits improve the legibility of the API as a whole, while additionally allowing for more thorough internal optimization of several features including the communication between hardware robots, simulations, and the end user as can be seen in Figure 4.1. Bidirectional communication between components is limited to minimize the necessary API surface, as well as to streamline the use of sensor and control fusion results where applicable.

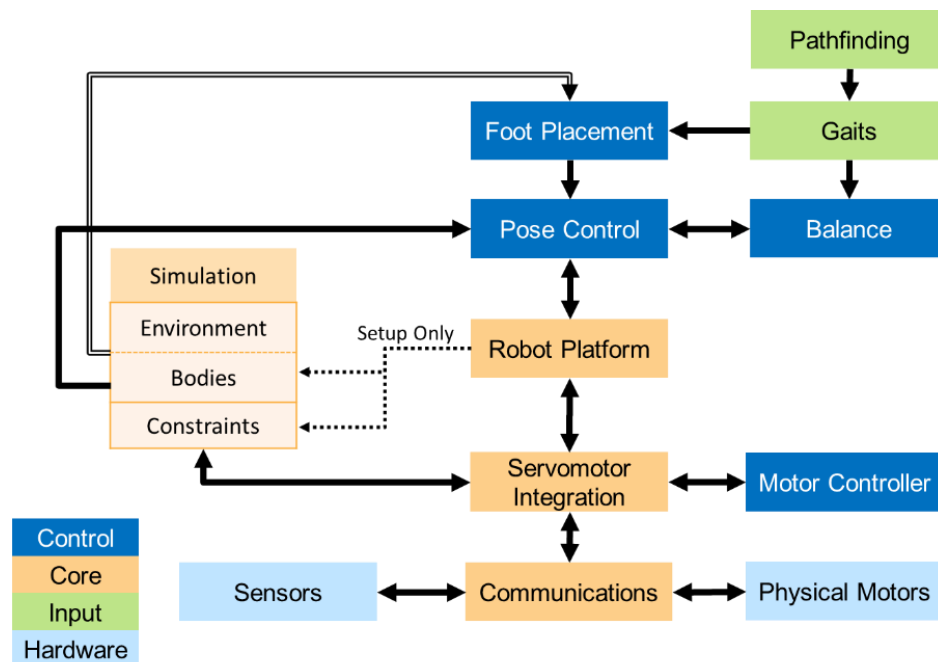


Figure 4.1: The control framework connectivity diagram in a typical robot control case. The general purpose behind each module is color coded into four overarching categories.

The control and simulation methods can be separated into four categories—Control, Core, Input, and Hardware. Of these, two are internal to the proposed framework, the Core components and the Hardware specific implementations of physical devices, while the two remaining categories are specific to the robot implementations desired which use the exposed framework surface to function.

The current core modules and brief descriptions are given in Table 4.1 and a full list of the components can be found in the framework documentation. Core components, while functionally independent, have several software interdependencies due to the tightly integrated nature of components and limitations of the statically typed C++11/14 programming language used to implement them.

Table 4.1: Table of core modules in the framework. Key software files and overall function descriptions are given. The module names and superclass are used interchangeably except where otherwise noted.

Module Name	Superclass	Function
Motors	RobotServoMotor	Servomotor definition and control protocols
Communications	RobotCommunications	Hardware communications
Control Clock	rcClock	Synchronization capable clocks
Robot Platform	RobotPlatform	Robot definition and pose control
Simulation	RobotSimulation	System simulation parameters and utilities
Environment	RobotEnvironment	An extensible definition for non-platform objects in the simulation, including obstacles and ground

A simplified view of these software interdependencies can be seen in Figure 4.2. As is the case in many APIs, the API surface has changed over time, resulting in several name changes. In particular, the core motor class, TTLServoMotor, was replaced with a more complete RobotServoMotor class late in the development cycle. These changes were implemented to facilitate documentation and eventual publication of the framework library. API related figures in the following sections may include older names for certain components due to the late timing of these changes.

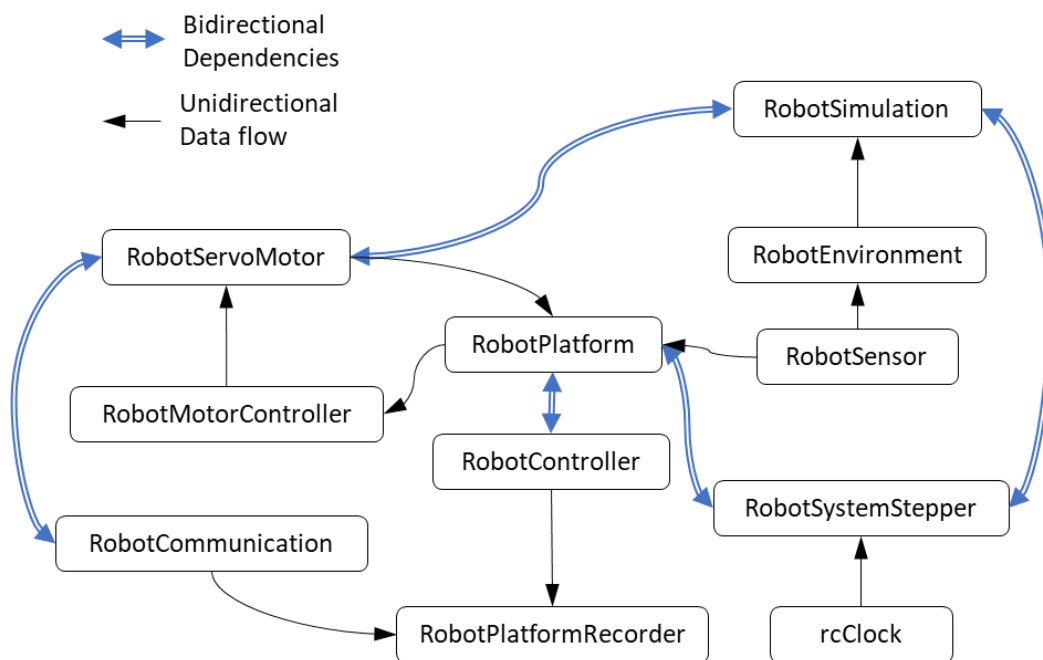


Figure 4.2: Major control framework modules and their components. Classes are centered about the RobotServoMotor and RobotCommunications super-classes.

Although the above can be implemented in using message passing systems like ROS using discrete nodes, the core framework was implemented entirely using language level polymorphism, with single and multiple inheritance, and pointer based shared memory. This low-level implementation in turn allows for high bandwidth duplication and zero-copy reuse of robot and simulation objects within sub-networks, such as the concurrent simulations in optimization based system identification and data

logging necessary for a variety of functions. Message passing based multi-process frameworks would produce significant overhead from data copying and serialization for similar uses, especially for high frequency, small data transfers where the overhead costs can be larger than the transmitted data. In the case that these sub-networks are otherwise independent from the main network, it is possible to execute the sub-networks faster than real time on a multi-core computer using threads within a single process to support full memory context sharing, and therefore minimize communications overhead. An example of such a system is discussed in Section 4.7.4 below.

For hardware robots with concurrent simulation, implementations of all core modules are required to obtain the full robot state and integrate that into the simulation and control. The modules can be initialized and interconnected in any order desired, with the exception of the motor modules, which must be initialized and attached to the robot platform before the platform is attached to the simulation. This is an internal dependency issue due to implementation related limitations of the simulation, as the constraints that represent motors must be created at the same time as the bodies they connect to. The RobotPlatform acts as a supervisor to both the motors and the kinematic representation – including rigid body definitions – mandating an order of dependency resolution allows the system to more easily initialize simulations and communication. A typical setup algorithm for such a quadruped robot is given in Algorithm 4.1, where specific implementations are used in place of the general class. In the case of the test quadruped, the motor classes used is DX_MX_28, which is a full control and simulation implementation of RobotServoMotor, and a custom quadruped implementation is available for the RobotPlatform class. Discussion on the implementation of the simulation in the topic of Section 5.

Algorithm 4.1: Quadruped Setup

- 1: for $i = 1 \dots N$ Motors
 - 2: Configure Motor[i]
 - 3: Attach Motor[i] to Communications
 - 4: Attach Motor[i] to Platform
 - 5: Attach Platform to Simulation
 - 6: Attach Environment to Simulation
 - 7: Start Communications
 - 8: Start Simulation
-

4.3 Motor modules

The hardware and simulated servomotors are managed by a RobotServoMotor derived class for use by control structures and communications devices, and a derived class is required for the proper operation of any simulated or hardware robot, defining the unique control and dynamics of a given motor type. The RobotServoMotor class manages at most one digitally controllable hardware device and one simulation based software device per instance, aggregating and integrating all necessary information from both to determine the observed motor state for use by other modules.

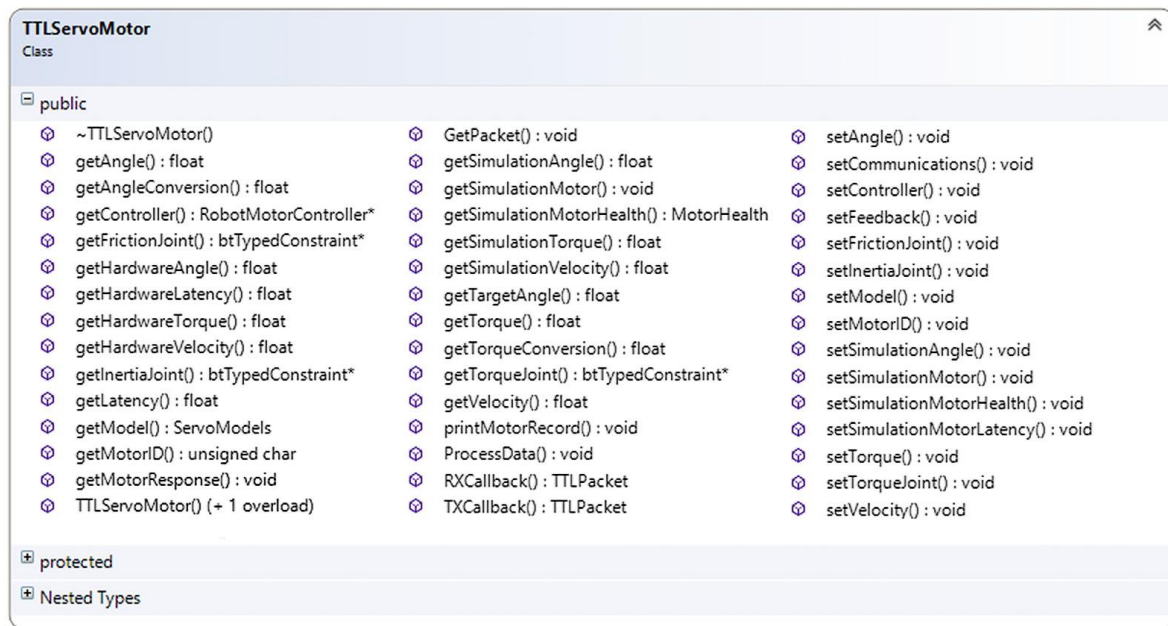


Figure 4.3: Class diagram of publicly available functions and members for the RobotServoMotor (TTLServoMotor) super-class. Return types are given, with float type and btScalar being functionally identical.

Data is segregated into hardware and simulation sources internally, with overrides to allow external access if the motor type includes both sources. The default return mechanism, however, is a fusion of hardware and software values that attempt to improve accuracy and allow for state estimation between hardware sample when hardware data has a low sampling rate or high noise. Although plans for custom sensory and state fusion algorithms for use with high latency, low accuracy motors exist, the current implementation is an automatic switch that returns the hardware values if possible, or returns the simulation values in the case that no hardware is connected. This allows for simple conversions from physical robot control to simulation only operations without the need for user interaction.

4.3.1 Physical Motors

Hardware based servomotors use the full RobotServoMotor API surface to synchronize with, control, and simulate physical motors within the framework. Each unique physical motor unit has a derived class implementation of RobotServoMotor that includes the full control and simulation specifications of the motor type, including secondary interactions between known motor status and its effects on both simulation and control. The current format of the RobotServoMotor expects several features and limitations from all inherited classes used to communicate with hardware and software devices to ensure proper operation within the framework.

Each physical motor type includes the following features:

- Packet communications encoding and decoding
- Motor sensor state and health information
- Motor communications latency estimation

Physical motor types are subject to the following limitations:

- Only one communications target can be used
- Only one simulation target can be used
- Hardware and simulation targets are synchronously controlled

Although physical communication is achieved through a separate module, an internal packet communications serialization method allows motor types to be swapped without modifications to the communications device implementations. This protocol separates read and write commands into separate and exclusive packet types depending on blocking behavior of the transfer. Further discussion of the packet structure and use is described in Section 4.5.1.

Often times available servomotors use brand common or even industry standard communications protocols for a group of related servomotors. For motor families that share communications and other specifications, it is possible to use an intermediate class for the motor family to further derive an additional class for specific motor types within the family. Some level of future compatibility with hardware changes can be maintained by using such implementations in the case a derived class specific to that motor type has not yet been added to the framework. One such case is described in Section 4.8.2 below.

In order to allow for additional framework testing and the use of simulation components for systems like inverse dynamics and kinematics, virtual motors – software implementations without physical counterparts or hardware interaction methods – were designed and implemented as motors with perfect transfer functions for control. Although the internal controllers are vastly simplified compared to hardware specific implementations, it is possible to use the same cascade control algorithms as the physical motors. This can be beneficial for the testing of new second level controllers including gait and balance controllers, as it is possible to test them on an ideal response prior to implementing them with little or no modifications on actual hardware.

Three main implementations were developed for various internal purposes, providing position, velocity, and torque control as necessary. The position control motor implementation is straight forward, using minimal internal controllers that drive the model to achieve the velocity necessary to reduce the position error to zero in a single step without consideration for control effort required. In the case a response more closely resembling a physical motor is required, simplified versions of known motor torque models and kinematic derived velocity approximations were implemented. Internal controller approximations for these types are discussed in Sections 6.2.1 and 6.2.2.

4.4 Sensors

Sensors, unlike motors, are less strictly defined in the core components due to their variety in functionality and control methodology. Unlike their motor counterparts, only hardware connected motors are currently specified as unique classes, with software-only methods being integrated directly into the simulation for performance reasons. The hardware control layer for physical sensors is identical to servomotor modules, using hardware specific packet based communications that can be transferred via existing communications devices. Simulated sensors, however, are significantly different in

implementation than the servomotor module representations due to differences in how the simulated environment treats various objects that these sensors may pull data from. More information on several virtual sensors is available in Section 7.5.

4.5 Communications Module

As robots are comprised of many interconnected, digitally controlled parts, a system of communication between different combinations of hardware and software is necessary to improve system integration. There is a trend in research platforms to utilize universal communications mechanisms, such as TCP/IP, throughout the hardware and software of robotics to simplify implementation. Software systems such as ROS use partial and full TCP/IP and UDP stacks to communicate between software components and hardware alike [45], while EtherCAT uses ethernet physical layers to allow hardware to communicate at high speed while still offering compatibility with TCP/IP and UDP based controllers. Although it is possible to use universal communications mechanisms including TCP/IP and ethernet for all devices, it is often prohibitively expensive to develop hardware utilizing these, and equally computationally taxing to implement the serialization and deserialization necessary for data transmission.

A more streamline system for communication with hardware devices with support for varied types of actuators and sensors was necessary to connect the robot control software to the physical devices. The base system was required to provide low latency, high throughput communications without full knowledge of the hardware specifics or quirks. Several significant challenges were presented due to the use of low cost hardware, which necessitated software based solutions to improve overall performance.

4.5.1 Packet based communication oriented

The communications methods for hardware components are often dependent on the device manufacturer and model, and may be incompatible with additional devices. To ensure that multiple, potentially unrelated devices can be accessed with minimal additional hardware, device communication for all hardware components was required to be conducted in a digital format with discrete and compatible packet structures using a physical transport layer shared by all devices on a particular communications hardware device.

A digital packet can be broken up into several components which are unique to the protocol being implemented. In general, packets include:

- Header – which defines the starting point of the packet
- Target location – which identifies the intended recipient
- Packet length information
- Data payload – which is the raw data being sent
- Validation – which provides a method to identify missing or corrupted data
- Terminator – which identifies the end of the packet

While most packets include this functionality, it is not necessary that each function be separate, and often times the packing is done such that the minimum optimal number of bits necessary to provide

the required reliability and functionality are used. Header, target, and packet length are often combined into a multi-byte heading that includes the functionality of all three, and data payload, validation, and terminator are often packaged through encoding mechanisms at the data and hardware levels to improve transmission speed by minimizing data bits necessary to validate and signal data termination. Several examples of packet structures often used in industrial robotic are shown in Figure 4.4 below, both of which are possible to implement in the current framework.

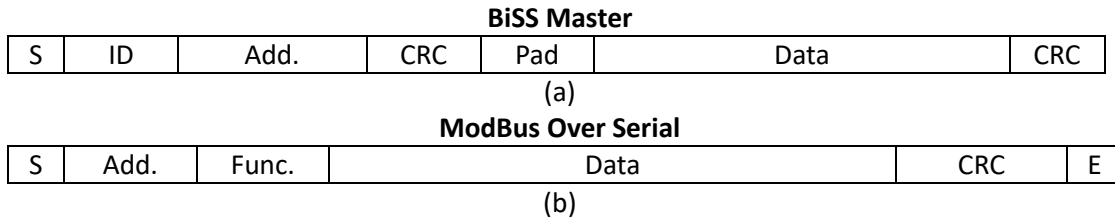


Figure 4.4: Digital packet types often used in industrial robotics, BiSS (a) and ModBus (b), with relative data sizes for each component. Both packets include a distinct start value *S*, a target address *Add.*, and a CRC in addition to data.

Although the framework is agnostic to the internal workings of the packet necessary for each device type, problems can arise in the case several devices use similar, but incompatible packet structures without proper validation mechanisms. In communications collisions like this, the information of one packet may be misread by devices as a valid command, and may result in control failures when impossible command target are set. As such, all data traffic through the communications module is expected to be comprised of data with a non-trivial header and a self-contained termination sequence which cannot be validated by more than one target device per communications device. This additionally opens the possibility of open-loop bit-bang control of analog devices under the condition that only one target device is attached per communications device. Control frequency and stability are not guaranteed for this particular use, however the ability to implement analog devices with certain conditions can allow for additional configuration options, particularly for devices that do not require high control bandwidth.

Since the earliest versions of the robot control framework, hardware control packets have been internally encapsulated in a generic packet structure for portability and performance as well as logging purposes. In addition to the raw, device specific data to be transmitted, data length, expected return length, timestamp, and data validity information were included into the `TTLPacket` class structure to assess the packet type without knowledge of the digital signature of each data type. In particular, the expected return length was included to allow the packet to request a response of a certain length, which can be used by the hardware communications module for data flow control, as well as to provide bounds checks to verify the length of the return packet for communications not necessary conducted with more robust communication layers like TCP/IP.

4.5.2 Latency Considerations for Serial Communications

When communications methods are implemented without consideration for transmission and serialization delay sources, there may be considerable effects on system performance and data accuracy in addition to feedback data phase shifts, the delay as a ratio of latency to input frequency between the actual data and the value when the data is received. In the low maximum bandwidth

communications case shown in Figure 4.5, the system shows an average latency of 45ms and a variance exceeding 6ms. The average error between the latency compensated response data and the unfiltered response data is $2e^{-2}$ radians, which is an order of magnitude larger than the sensor resolution dependent position error and larger than the maximum simulation error of a single motor – further discussed in Section 6.5. Although data may be smoothed to provide results that are more suitable for robot control, this adds undesirable additional control delay in the order of tens of milliseconds.

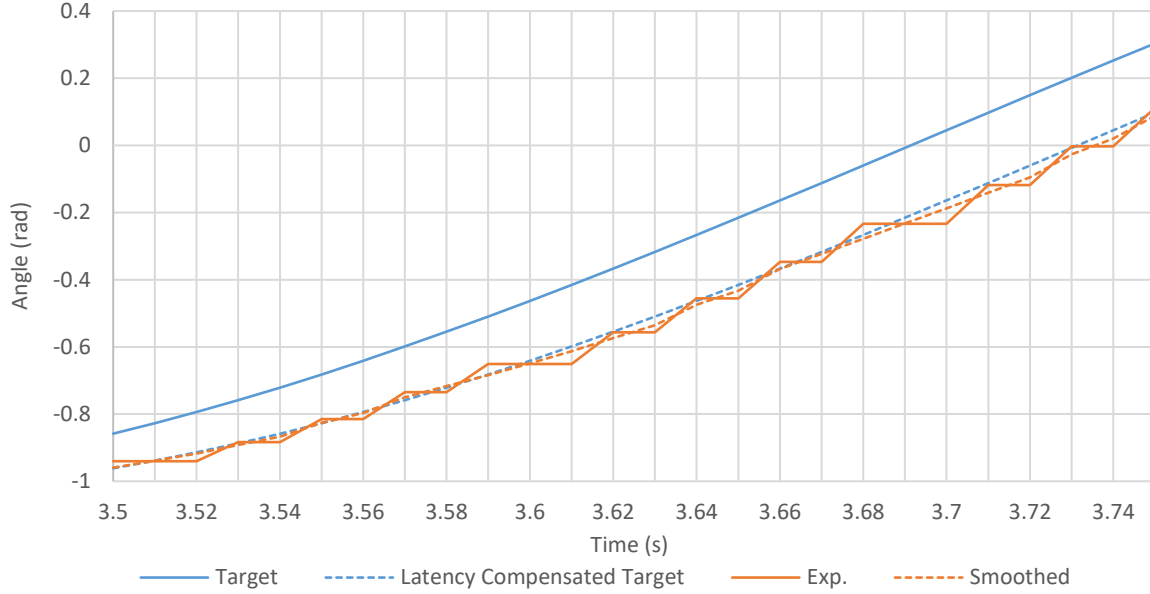


Figure 4.5: Low communications bandwidth (<100Hz) results from a bound gait on the quadruped robot. The latency compensated target (-45ms) and smoothed experimental data (50ms smoothing) have minimal relative error, however the raw experimental position error standard deviation is large.

Hardware device communications are by nature imperfect, requiring a certain amount of time to execute commands or return device state information, which affects the bandwidth and phase shift of control algorithms that require data from these devices. These imperfections can be separated into various layers of the Open Systems Interconnection device networking model [47], each with varying degrees of impact on the total communications performance.

At the physical layer, electrical level in which raw data is transmitted, digital bits are represented by symbols defined by voltage differentials on communications interconnects. Latencies at this layer are a result of the physical limits of changing magnitude and polarity of wires, due to imperfect amplifiers and even the length of the interconnect, as well as delays due to electrical filtering to produce reasonable noise rejection margins. This delay, also called a frame time, can be calculated by the communications packet size divided by the physical layer baud rate of the device and is generally measured in tens of μ s for typical physical layers used in robotics and automation. In older PC hardware and many available microcontrollers, OSI layers beyond the physical layer are generally negligible at lower control rates as the communications were directly tied to processor operation as an interrupt. In systems with this type of direct serial connection interrupts to the CPU, this secondary latency essentially becomes zero as the frame time is the minimum time required to transmit data plus any remaining CPU clock cycles necessary for resetting the communications.

For newer communications protocols and hardware however, the data link layer of the OSI model can become the largest contributor of latency, as communications framing has a significantly larger overhead due to the way in which data is aggregated into buffers before physical transfer of the data. Modern computer systems currently use PCIe and USB based devices instead of serial interrupts directly in the CPU due to platform simplification and performance considerations at an architectural level. More expensive PCIe based serial devices are generally limited by the PCIe frame times, generally in the 20 μ s range for an otherwise idle computer [48], although variability from system to system depends on the particular PCIe and southbridge implementation as well as overall system load.

In the case of less expensive communications used in the robot, the frame times are of particular concern when implementing communications through a USB interface, as earlier hardware uses frame polling of 1ms, allowing data to be sent with a delay of up to 1ms after it has been received in the device. Newer versions of the USB interface have improved the polling times to 125 μ s, however serial TTL adapters for faster USB versions are not common. Due to the nature of frame polling and transmission requests, a minimum of two frames must be used for any send-receive communications over a frame based communications device. Although command processing times and delays associated with the network and host layers can exist at high speeds with low powered CPUs, they are negligible in the robot platform when compared to the transmission of data using serial TTL and USB communications.

4.5.3 Asynchronous Communications

A general closed-loop control structure functions synchronously with sensor and motor commands to ensure the most up-to-date instructions are given to hardware devices. Sensor information is read before calculating new actuation targets based on the most current data and a control law that assumes this new information is effectively concurrent with the control action. This communications cycle, described in Algorithm 4.2, is most effective when the communications polling rate includes additional delays much lower than the time necessary to transmit new commands.

Algorithm 4.2: General closed loop robot control process

```

1: while robot is operational
2:   for each motor
3:     request position
4:   end
5:   calculate new command positions
6:   for each motor
7:     send command position
8:   end
9:   update simulation
10:  record data
11: end

```

In the case of USB communications, it is difficult to achieve such small delays while also executing computationally intensive simulation and controller calculations. Likewise, the execution blocking that would occur while waiting for the read-reply in a synchronous system can severely decrease performance in other control steps even when the increased data accuracy is expected to provide little additional information. It is possible to take advantage of multi-threaded programming to lower the communications overhead while improving throughput by utilizing an independent communications thread running asynchronously with the remaining code. The implementation chosen for the framework is an in-process thread creation with core affinity selection at the time of the communications device creation. Core affinity within multithreaded systems specifies a preferred CPU core, minimizing cache coherency issues that may occur when the communications thread is moved between different logical processors, which requires a time-consuming memory copy from the original cache location to the new one. This implementation allows the thread to run semi-exclusively within the hardware, reducing operating system and computer hardware overheads in the communications chain due to context switching, matching the ability of real-time OS implementations to switch faster than the data polling for the case of USB communications [49].

For the control and simulation systems to function asynchronously to the communications, latency recording and analysis must be added to any modules that employ systems that may have latency, particularly those representing hardware external to the framework execution computer. The sensors, RobotServoMotor and RobotCommunications modules include recording, analysis, and propagation of hardware latencies in order to properly simulate the system with latency feed forward compensation. The servomotor latency is calculated within the RobotServoMotor module using timestamp data from the RobotCommunications module during a callback command from the communications module, and stored as read command, write command, and reply latencies to better replicate the communications structure even in the case of unknown communications burdens on the system. The write latency is recorded as the time between write callbacks, while the read latency is that of the read command callback. The reply latency however is the time between a read command and the reception of the reply of that command, which can help minimize the uncertainty of data collection time by monitoring the bidirectional communications hardware delay independent of additional processing and polling rate delays.

As communications delays play a large role in the accuracy of simulated systems, all forms of delay are applied to the simulation to ensure that command values are propagated as in a hardware system. This is done through a delay budgeting system which configures the visible target commands based on this delay budget. In the RobotServoMotor implementations, the filtered values of the respective delays are combined to provide a maximum internal delay budget for simulations running in parallel with the data collection. This maximum budget is depleted at a rate dependent on the simulation stepping, updating the target commands exposed to the simulation when the budget is expended, additionally resetting the budget to the most up-to date maximum. This ensures that any unexpected delays due to communications failures or excessive communications loads are fed back into the simulations automatically.

4.5.4 Passive communications optimizations

In an ideal case, the packet based communications transfer time is given by the size, in bytes, of the command packet divided by the communications device payload transmission rate, in bytes per second. For a packet of the shape given in Section 4.8.1 below and a communications device with a rate of 50kB/s (500 kBaud), a single packet requires 180 μ s for transmission, allowing up to about 5500 packets per second to be transferred. For a typical quadruped robot that sends one packet and receives one packet per motor each control cycle, a control rate of over 230Hz is possible at the 50kB/s communications speed. In practice however, the maximum commands per second and communications rate fall well below this value due to additional time necessary for the processing and response of packets, as well as margins reserved to prevent communications collisions. Buffered communications with low delays between packets can mitigate some of the discrepancy between maximum command rate and maximum throughput by transmitting packets at a more optimal pace without collision issues. This can be a significant source of latency, as buffer queuing delays every packet and may cause phase shifts between the data transmitted and its intended transmission timing. Further discussions on command rate and buffer considerations due to hardware specific implementations are discussed in Sections 4.8.1 and Section 4.8.2.

Different hardware components have widely varying latency characteristics that must be considered on a device basis. The total latency of the command packets in a queue system is the sum of the hardware latencies plus a queue wait time equal to or greater than the communications cycle length, assuming the command cycle rate is lower than the communications cycle. The base cycle length is the number of frames required to transmit and receive any outstanding packets multiplied by the frame length, however internal buffering may allow multiple write commands if read and write commands are segregated into independent queues to prioritize certain types. The outline of this buffer method using a single buffer over a USB Full Speed device and three devices is given in Figure 4.6. In the case that the data is scheduled to send after the beginning of a frame, it will be sent in the following frame as shown in Figure 4.6.

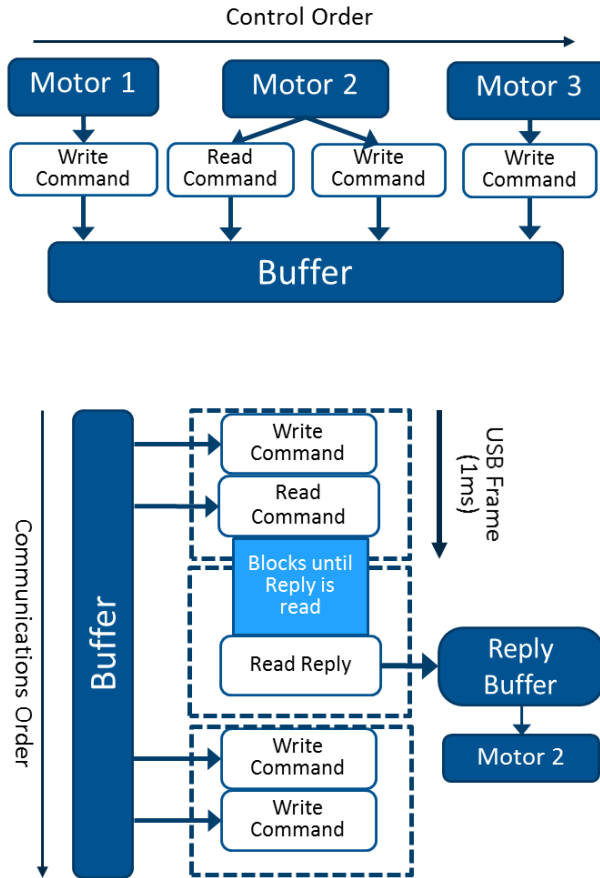


Figure 4.6: Buffered communication method for packet based communications for a robot. A worst case scenario is shown, where command order requires an additional communications frame.

Due to the synchronous frame timing in the USB protocol, read type commands block other communications until the frame following the reply, which can lower queue efficiency and introduce large, queue order dependent latencies. Previous robot control systems [8] [50] used buffered communications with packets added at every command, which introduced several complications during real world use with bandwidth and latency limitations.

The primary concern for a queue based system is bandwidth exhaustion leading to a near FINO (“First In, Never Out”) condition, particularly if the queue implements both a finite size and content timeouts. The primary method in which this communications error can be achieved is by filling the queue at a rate faster than it can be dequeued, for example when the command rate exceeds the physical transmission rate. In a worst-case scenario, when multiple read commands are issued in a row, the completion time can be significantly increased over the average, leading to queue stalls even when the command rate is lower than the average communication rate. Although this can be mitigated by decreasing the ratio between communications and command rates, the robot platform performance greatly decreases, especially when hardware based pose controllers are used. Prior methods limited the robot to a fairly low command rate of 10Hz with a communications rate of 118kBaud to prevent such issues.

Also of concern is the data promptness, the time delay of a command packet from the time it is entered into a queue until it is actually transmitted. Further variability in USB communications occur due to the

fairly large window for transmission and an unknown transmission order, as the packets may be sent or arrive at any time within the transmission frame window. In addition to the sensitivity of robot response to delayed command inputs, this delay is critical in the comparison of data, as the responses between trials may be significantly different. These delays are inherent to command buffer techniques and cannot be mitigated without an ability to modify the contents of the queue to ensure the most up-to-date commands are issued.

To eliminate the majority of latency sources and improve reliability of the communications stack when using variable latency communications, a software callback based queue system with return-type segregation was implemented as shown in Figure 4.7. The callback based approach allows the communications module to request a new target state when the queue slot is ready, minimizing the queue delay to near zero. Likewise, return data can be provided immediately upon receipt, allowing the receiving module to update motor states without waiting for an additional synchronization frame.

As with the standard buffer approach, two transmission queues are used, a non-blocking queue reserved for one-way communication to hardware (write queue) and a blocking queue that sends commands to devices to request data (read queue) that is paired with a response queue that is stepped with the read queue to receive data from the hardware (reply queue). For given hardware buffer limitations, the system can be configured to automatically send the maximum amount of non-blocking data before a blocking command is issued, based on the length of the commands and the total buffer sizes. For half-duplex devices, a maximum number of outgoing packets is determined such that a maximal number of one-way packets are terminated by a request packet while constrained to the output buffer.

This technique can be also be used within full-duplex devices, those which can send and receive data simultaneously, as the two types can be interleaved by setting the effective buffer size to the length of one write and one read command. In addition to full-duplex devices, this interleaving methodology can also be used for systems that use aggregate packet communications such as those discussed in Section 4.8.2. This callback technique ensures the lowest possible command latency for a given bandwidth regardless of the type or number of communications devices attached.

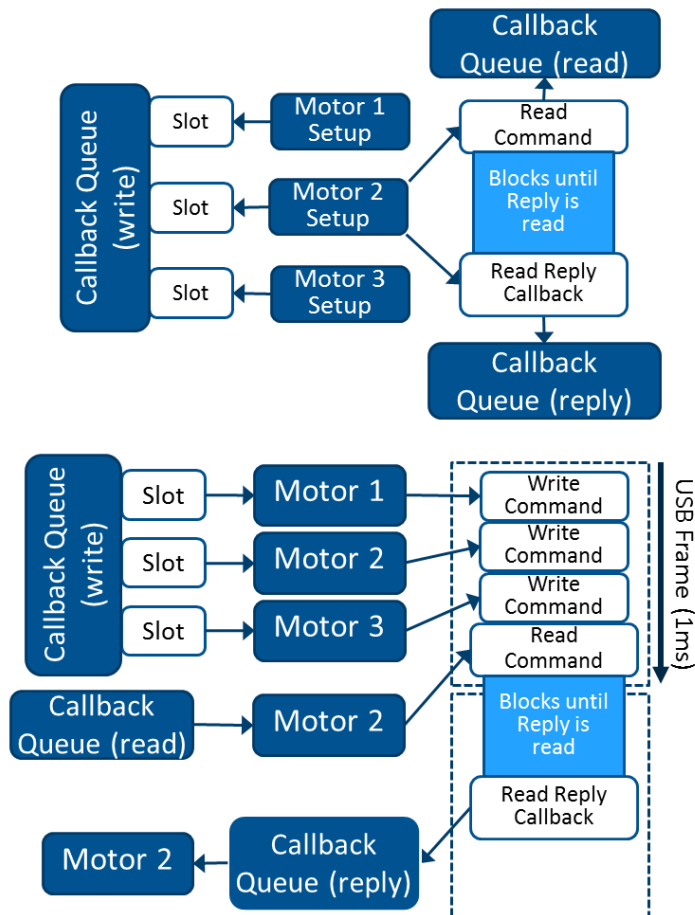


Figure 4.7: Asynchronous command queue system for hardware communications. Three callback queues are used to provide communications hardware independent optimizations

In practice, it would become unwieldy to limit the system entirely to callbacks, particularly for hardware initialization and setup communications. As such, the communications system includes both techniques to provide flexibility. The preferred communications method however is the more efficient callback queue. The implementation used for communications devices in the robot control framework, including the FTDI R232 based USB-UART used in the current robot platform, is outlined in Algorithm 4.3.

Algorithm 4.3: Optimized Communications cycle

```

12: while queue is operational
13:   for each in priority queue
14:     add packet to driver send buffer
15:     If packet has return size then
16:       wait for response
17:   end
18:   while send buffer < max size
19:     poll write-callback queue
20:     add packet to send buffer
21:   end
22:   poll read-callback queue
23:   add packet to send buffer
24:   wait for response
25:   add response to reply-callback queue
26: end

```

Unlike processes used in frameworks like ROS, threads run within a shared memory context, where main program thread and the communications thread are capable of using any module methods as if they were running within the same execution stream. As with any shared context simultaneous multiprocessing method, this may introduce data races when multiple execution streams access the same memory, unless the only modifiable data within the system are atomic types that always hold a valid state. For more complicated data structures however, it becomes necessary to include protections at the data access boundaries to enforce data integrity during asynchronous operation with multiple reads. Within the current implementation, communications threads protect data only while it is exclusively controlled by the communications thread, any additional protections are outside the scope of the module and instead determined by the originating or target module. As a side effect of this callback procedure, the need for memory protections outside of the addition of callbacks and priority packets is eliminated within the communications class, further reducing possible sources of latency.

4.6 Simulation Module

In addition to hardware related modules, the framework includes a simulation module that can be used with or without a physical robot attached. This highly integrated simulation module provides visualization and state estimation of real and virtual robot platforms. The physics simulation is populated by environmental objects including ground geography and obstacles, and robot platform, which is comprised of modules and other controllers and sensors. Similar to the communications and robot platform module methodology, the simulation module allows connections to multiple environmental objects and robot platforms which share a single simulation environment.

In order to minimize API abstraction overhead and improve development time, the simulation module and all related functionality is currently assumed to be based on the 2.81 branch of the Bullet Physics Engine [29]. As such, Bullet specific data types are used within the simulation, environment, robot platform, and motor core classes, as illustrated in Figure 4.8. The communications class does not share these data types as the functionality is entirely independent of the simulation.

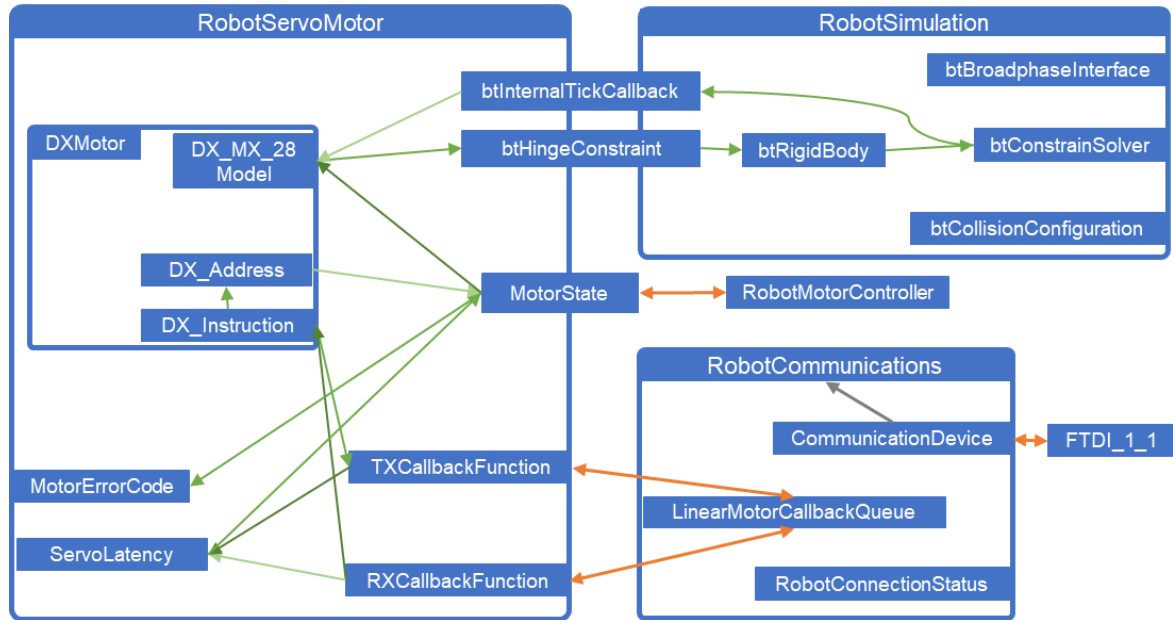


Figure 4.8: Object dependency in the framework and approximate data flow. Classes shared between major components straddle both groups, while classes only referenced between components are connected by orange lines. The green lines represent internal connections and data flow between more complex sub-classes within the main component.

Much of the system functionality is dependent on the simulation specifications, from the definition of the robot to the use of virtual sensors. In depth analysis and implementation of the simulation module is the main subject of Section 5 and Section 6.

4.7 Additional Tools and Utilities

In addition to the primary modules and their implementations, there are a variety of additional features necessary for the simplification of several aspects of the robot identification and control. These include low level systems such as state recording and processing, as well as higher level components for system identification and optimization.

4.7.1 Robot state recording

Data collection and access is necessary for proper operation of closed-loop robot control algorithms and general analysis of the robot performance. In addition to the hardware communication paths discussed in Section 4.5, a method of unified data recording and access is necessary to aggregate data from both the hardware and simulation at the same time, which can then be used during both offline and closed-loop control to validate the robot state. This trace data can include additional information not limited to the hardware response such as robot pose, command targets, and estimated system variables.

A dedicated robot state recording module was developed to improve the functionality of traditional control and optimization methods within an asynchronously updated robot platform. The physical and simulated robot state parameters necessary for optimization methods were recorded at a regular interval dependent on the system stepper. All information of the current robot state, including current latencies, is packaged into a serializable data store that can then be used by other modules. A summary of the class is shown in Figure 4.9.

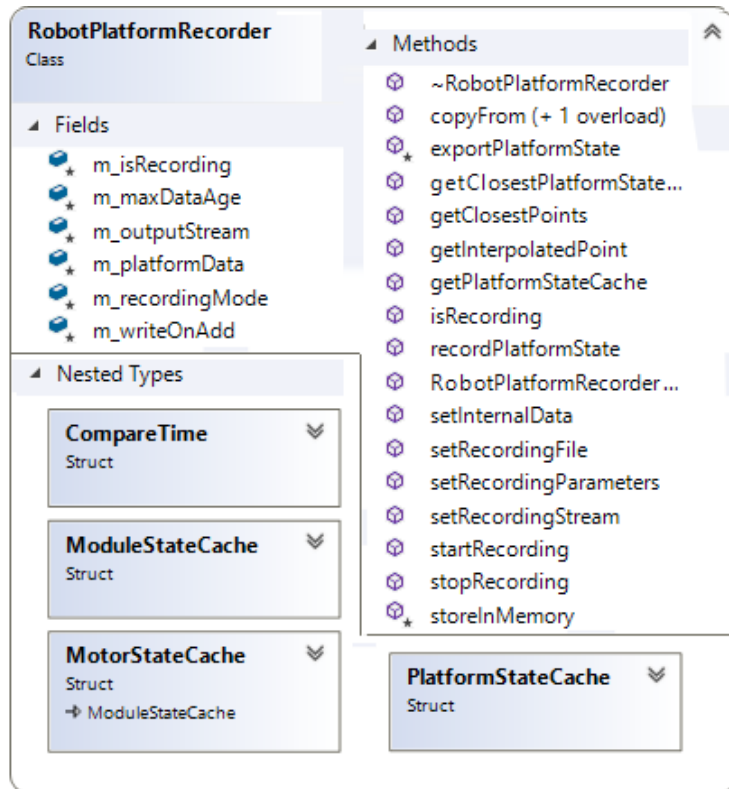


Figure 4.9: Robot state recording module and recorded states API surface. Data storage and interpolation between points are integral to the `RobotPlatformRecorder` class.

4.7.2 Multithreaded cache based comparison

As optimization using parallel simulations require extensive access to the same filtered and resampled time domain baseline data, a compromise of performance or memory must be made. A solution that is trivial to implement is a full scope copy of any data required by the requesting algorithm such that each independent thread owns an exact copy of the data. These copies can be made with language level copy mechanisms for relatively fast duplication of static data. For simple robots and a low number of threads this can often be acceptable, with an average quadrupled requiring less than 400kB of data per second per copy at 240Hz simulation rate and less than 1000Hz hardware sampling rate. An additional requirement for this method is that all data must be static, and all simulations must run at multiples of a base rate to pre-emptively resample and filter hardware data. While this is often the case for the control framework when used for system identification and optimization, real-time use of hardware would produce data in a dynamic fashion and complicate implementations of direct data copy, as each and every copy must be properly updated when new data is added.

For more complex data sets beyond the absolute minimum joint data however, a more sophisticated mechanism for data storage and manipulation can drastically increase performance. This can be achieved by pre-formatting data and taking advantage of multiple concurrent memory read capabilities of modern multiprocessing computers. As such, a trace data memory management system designed for multiple concurrent read and singular write was developed using language level containers and algorithms to provide portability between operating systems and processor platforms. As writing to such structures is non-trivial in multiprocessing systems due to potential race conditions, locking mechanisms derived from the C++11/14 implementations of `std::mutex` are used to provide data boundaries to a cache of formatted and filtered data. All externally accessible data is pulled from this cache if available, or requested from a private pool of raw trace data for processing. The data is therefore only processed once requested, minimizing overhead related to processing of excessively detailed information.

One implementation of this data aggregation system, outlined in Figure 4.10, was developed to provide the data manipulation necessary to build a fitness function to a given set of simulation and hardware data. As simulation and raw data are often sampled at different times due to the nature of the asynchronous communications, this class attempts to filter out latency jitter in raw data and match it to a fixed time-step simulation input through interpolation. Formatting the raw data to the simulation rate and variable type allows for a more accurate error calculation during the computation of a fitness function.

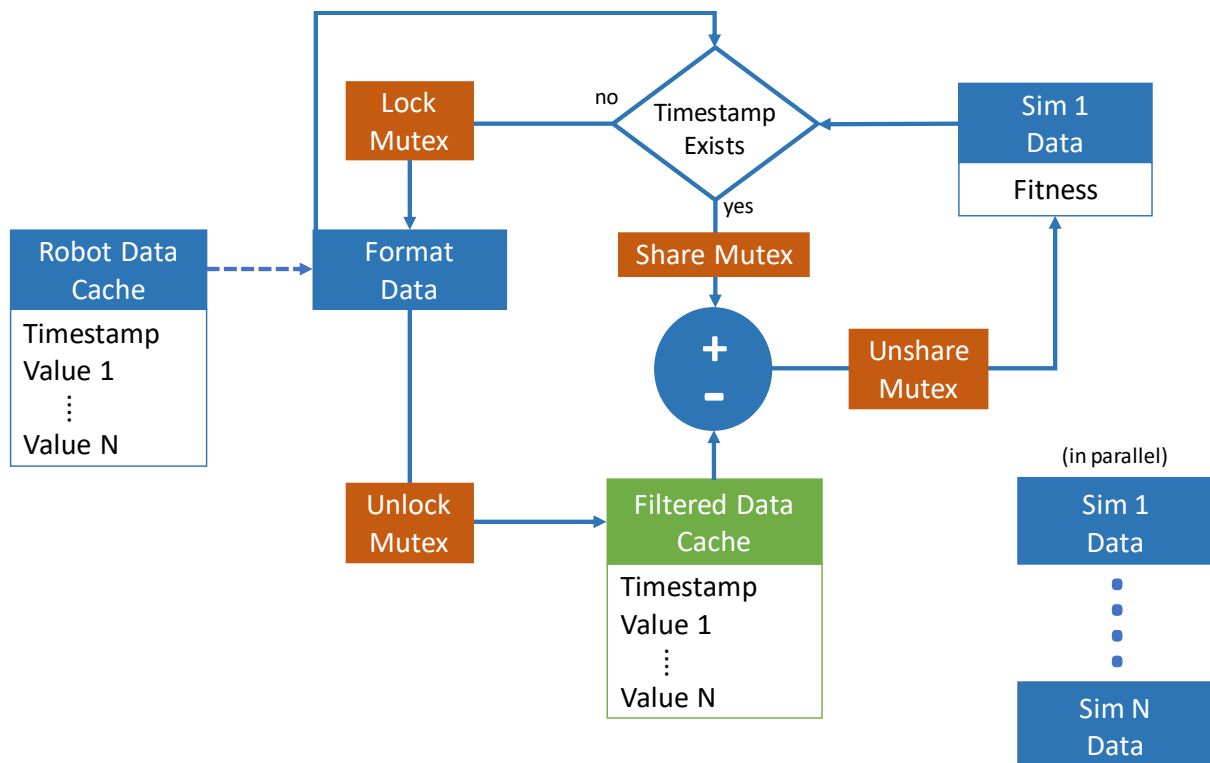


Figure 4.10: A thread safe cached data comparison filter for calculating robot fitness. Read operations share a multiuser mutex while commands that change shared data exclusively lock the mutex to prevent unsafe memory access.

In the base implementation of the management code, data read commands attempt to acquire a `std::shared_lock` to the filtered data cache in the case certain data currently exists, avoiding the overhead of an exclusive locking mechanism while preventing race conditions in the event another thread is adding new data to the cache. In the event necessary data is not present in the cache but can be interpolated from the raw data the cache is built upon, the read command requests an `std::unique_lock` on the cache to add the interpolated data to the cache. As the cache can become unbounded in the case of online use, a circular buffer of sufficient size is used for the cache area, allowing the oldest data to be retired without additional overhead.

Although this method increases time for initial access to not-yet parsed data, subsequent requests for data are significantly faster as there no need to copy the entire original data set. For pre-formatted offline data, this mechanism may be unnecessary as read-only access is guaranteed to be thread safe. However, this is necessary for real-time use as it prevents memory corruption as new data is continually added to the cache and older data is allowed to expire, as well as for systems employing multiple simulations with differing sampling times.

4.7.3 Optimization

Optimization methods are used for a variety of simulation related purposes, however they are not limited to simulations. While the optimization functions are entirely separate to the core control framework, the features added are integral to motor identification and estimative control. The structure of all optimization methods used follow the structure of Algorithm 4.4 and sub-Algorithm 4.5, however the use of threading, step 5 in Algorithm 4.4, is optional and is not used for simple function with a low number of samples per iteration. The tolerance T and continuation function $f(y_{min}, \dots)$ is algorithm dependent and may include output values, input values, and iteration count. Using language level polymorphism, as is the case with the framework core components, allows new techniques to be used with minimal effort.

Algorithm 4.4: MultithreadedOptimization

```

1: Input: Sample Function S, Max thread count C, Tolerance T, Sample Number N
2:  $y_{min} \leftarrow$  Large Number
3: for  $i=1...C$ 
4:   reserve OptimizationThread  $O_i$  with local copy of S
5:   start independent threads for each  $O_i$ 
6:   while  $f(y_{min}, y_{min-1}, \dots) > T$ 
7:     update sample parameters  $x_{k,j}$ 
8:     queue  $y_k \leftarrow S(x_k)$  to  $O_i$  and set  $i$  to next value
9:     wait for OptimizationThread counter = N
10:    reset OptimizationThread counter
11:     $y_{min} \leftarrow \min(y_k)$ 
12:     $x_{min} \leftarrow x_k, k = \text{index}(\min(y_k))$ 
13:  end
14: stop all O

```

Algorithm 4.5: OptimizationThread

```

1: Input: Sample Function S
2: Queues: Sample Parameter  $x_k$ , Sample Function Result  $y_k$ 
3: Shares: Counter c
4: while not stopped
5:   if queue is not empty
6:      $y_k \leftarrow S(x_k)$ 
7:     increment c
8:     remove from queue
9:   if queue is empty
10:    wait
11: end

```

Various optimization implementations can be developed based on the number of variable inputs desired for a given system as shown in Figure 4.11. This hierarchy allows any optimization method to use multithreading in a simple and low overhead fashion, which is required for real-time use. Although any type of optimization method can be developed with this toolset in mind, there was a larger focus on the use of heuristic and other non-gradient methods, which do not require well defined gradients and may work on highly irregular solution surfaces.

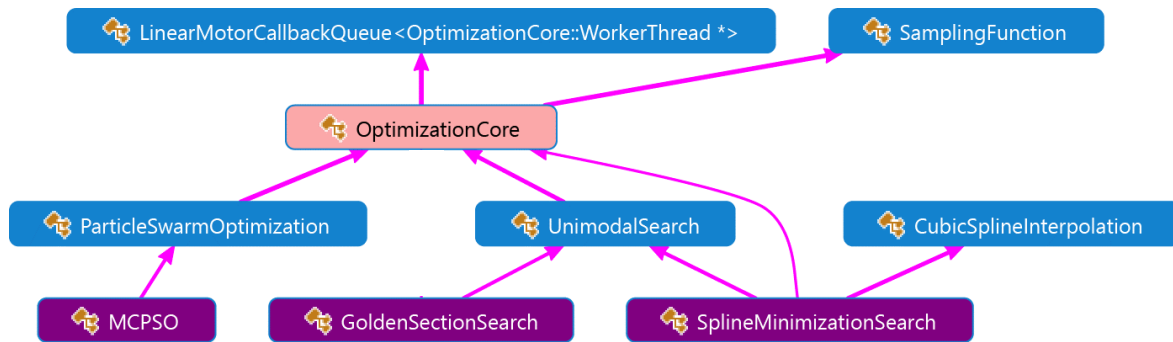


Figure 4.11: Optimization toolset for the control framework. The sampling function can be any subsection of the framework including a full robot simulation or hardware robot.

When a large number of input variables is desired, stochastic and random optimization techniques can be applied, including evolutionary or Monte Carlo methods. As these methods do not require a gradient calculation, they provide simple, if less than optimally efficient, ways of determining the likely global maximum or minimum within a desired region in an n-dimensional space comprising of all possible input variable values. Stochastic methods reduce memory and system fitness shape requirements of at the cost of additional computational time necessary for sampling areas that may provide little additional innovation. Currently implemented particle swarm methods can be used to find global minima on n-dimensional fitness surface. These methods are:

- Particle Swarm Optimization (PSO), uses multiple independent sampling with the next sample parameters determined by a combination of the previous parameters, the particle fitness, and population fitness.
- Multi-colony Particle Swarm Optimization (MCP SO), a modification of the PSO method that uses multiple independent populations, with a single master group and multiple slave groups. The two main implementations are competitive, where the master group provides fitness information to all particles and the master group is the current best group, and cooperation, where the best fitness from the slave groups is added to the master group.

In the case that a single input variable is to be used with a function with unknown but unimodal shape, non-linear search methods can be used. As with the particle methods, no gradient information is required for the function of the search, however individual samples are not independent, as only a single sample is generally used per iteration. The two implemented methods are:

- Golden section search, a minima search method for unimodal functions that separates the function into two unevenly sized sections with a ratio equal to the golden ratio. Three initial samples are required and an additional sample is calculated per iteration.
- Spline minimization search, a multithreaded approach that calculates multiple initial samples simultaneously to produce a cubic spline estimation of the full function. The spline function is then searched for a minimum with a golden section search.

Golden section search can be used with any unimodal function to optimize at a very high rate using just logic statements on function sample. Although the lack of gradient information makes this theoretically slower to converge than other methods, in terms of iterations necessary, the minimal memory and mathematical manipulation are often much faster to compute. The entire function, in the

implemented iterative form, is outlined in Algorithm 4.6. When used in the context of optimization toolset, each call of the sample function within an iteration may be executed simultaneously in parallel, however each iteration must be executed sequentially.

Algorithm 4.6: GoldenSearchOptimization

```

1: Input: Sample Function S, Tolerance T
2:  $\varphi = \frac{(-1+\sqrt{5})}{2}$ 
3:  $x_l \leftarrow$  lower search bound,  $x_r \leftarrow$  upper search bound
4:  $y_l \leftarrow S(x_l)$ 
5:  $y_r \leftarrow S(x_r)$ 
6:  $x_{m1} \leftarrow x_r - \varphi(x_r - x_l)$ 
7:  $x_{m2} \leftarrow x_l + \varphi(x_r - x_l)$ 
8:  $y_{m1} \leftarrow S(x_{m1})$ 
9:  $y_{m2} \leftarrow S(x_{m2})$ 
10: while  $|x_{m1} - x_{m2}| > T$ 
11:   if  $y_{m1} < y_{m2}$ 
12:      $x_r, y_r \leftarrow x_{m2}, y_{m2}$ 
13:      $x_{m2}, y_{m2} \leftarrow x_{m1}, y_{m1}$ 
14:      $y_{m1} \leftarrow S(x_{m1} \leftarrow x_r - \varphi(x_r - x_l))$ 
15:   if  $y_{m1} > y_{m2}$ 
16:      $x_r, y_r \leftarrow x_{m2}, y_{m2}$ 
17:      $x_{m2}, y_{m2} \leftarrow x_{m1}, y_{m1}$ 
18:      $y_{m2} \leftarrow S(x_{m2} \leftarrow x_l + \varphi(x_r - x_l))$ 
19: end
20:  $x_{\min}, y_{\min} \leftarrow \frac{x_{m1} + x_{m2}}{2}, \min(y_{m1}, y_{m2})$ 

```

In order to maximize performance and minimize the impact of optimization based soft sensors on the control rate, the spline minimization search method was developed. In any simulation based optimization technique, the computational time of a sample can often be several orders of magnitude greater than the optimization method itself. By formulating an interpolated function, we can separate the expensive, in terms of computing time, sampling from the inexpensive optimization step, as well as make the samples entirely independent from one-another. As such, we can execute a preset number of samples simultaneously on multiple cores, reducing the necessary time almost linearly with the number of cores assigned to calculation, as shown in Figure 4.12.

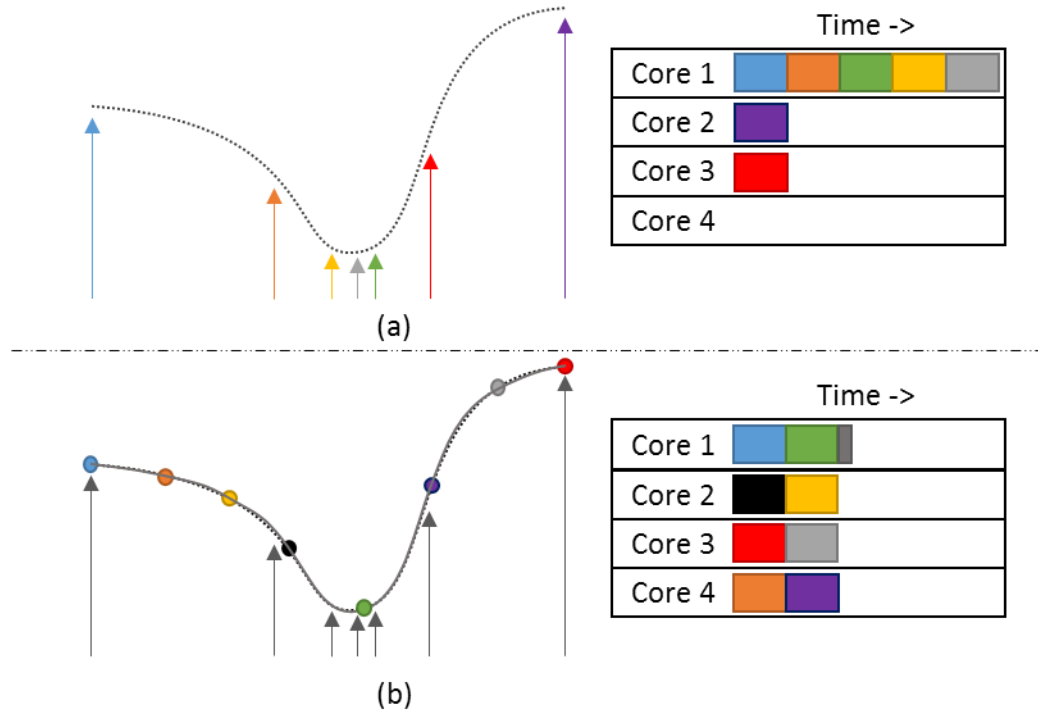


Figure 4.12: Comparison of the standard golden ratio non-linear function search (a) and sampled spline minimization method (b). Although sampling uses more CPU time, the overall execution time is greatly decreased.

This method can be implemented with any number of interpolation methods and secondary optimization. However, the current implementation uses a cubic spline interpolation with a golden search on that produced spline. The method, outlined in Algorithm 4.7 below, works best with a relatively large number of samples that ensure a minimal number of local minima that can occur even with monotonic input points.

Algorithm 4.7: Spline minimization search

- 1: Input: Sample Function S , Point count N , Tolerance T
 - 2: for $i=1\dots N$ do in parallel
 - 3: $y_i \leftarrow S(x_i)$
 - 4: Point $P_i \leftarrow x_i, y_i$
 - 5: Build cubic spline function C from all P
 - 6: $x_{\min}, y_{\min} \leftarrow \text{GoldenSearchOptimization}(C, T)$
-

4.7.4 Interpolation

Several interpolation methods for 1D and 2D data sets are available in the framework to simplify data analysis and improve control smoothing. In clarification, the dimensionality of these methods is in terms of the number of degrees of freedom of the interpolation space rather than the degrees of freedom of the data itself. Traversal along a predefined line through n -dimensional space would be considered to have a single degree of freedom, while a traversal on a predefined surface has two degrees of freedom. In particular, 1D methods are used directly for smoothing of data points for control and analysis. Currently two 1D spline interpolation methods are available, a classical cubic spline and a

monotonic spline. Assuming a two-dimensional input point, the classical cubic spline estimates the value y' of a point x' using a third order polynomial function of the form

$$y' = a_0 x'^3 + a_1 x'^2 + a_2 x' + a_3 \quad 4-1$$

with values a_0, a_1, a_2 , and a_3 calculated from surrounding data points in such a way as to produce a twice differentiable smooth function that passes through known points exactly. The solution for these variables can be calculated using the four closest points, two on each side of the desired x' value, by mapping the points as a parametric curve. Although generalized forms that work on three dimensional points are also available, a publicly available implementation that uses two-dimensional points was chosen for simplicity. The implementation used was originally developed by Tino Kluge and was based on the interpolating spline definition given by Doris Kochanek [51].

In addition to the classical method, a modification of the cubic spline is available for data sets where overshoot is undesirable. When used in non-monotonic sets, a monotonic spline maintains piecewise monotonicity by mandating that all critical points of the interpolation function lie on data points, which eliminates estimation based overshoot [52]. Due to this nature the method it is not well suited for interpolation of functions for minimization, however the method can provide very stable, fast, and smooth interpolation for data in the case linear interpolation produces less than desirable results. This is particularly useful for the smoothing of joint targets in locomotion methods that use low update frequencies, as the motion of motors is expected to be monotonic between updates.

4.7.5 Extensible Synchronized Clock

As many of the modules can be run in an asynchronous mode, a lightweight clock system was implemented for processing dependent and real-time operations in a synchronized manner. A high resolution, low jitter clock object was designed to allow a single master world clock to synchronize multiple independently steppable slave clocks at module specific synchronization barriers. When the master clock is based off the computer system clock, this allows a single real-time continuously stepping clock to drive multiple discrete time asynchronous clocks without requiring special compensation for desynchronization conditions due to potentially different step sizes in each clock. The master clock does not however need to be real-time, simplifying conversion of a system from real-time to simulation time without modification.

A version of this clock can be used internally in any module that requires time stepping, and can be found in both the communications and simulation classes to ensure the proper function of those classes. Although hard real-time timestamping is not possible without a real-time operating system and compatible hardware, the asynchronous nature of this clock system makes it possible to reduce time jitter at each module level and improve overall system performance.

4.7.6 Discrete System Stepper

While the system generally is run with the major modules executed asynchronously of each other in real-time, an optional discrete stepping method was added for use with faster than real-time offline methods and system visualization discussed in Section 5.6. In this method, the simulation, control, and optionally the communications components are synchronously stepped in such a way as to ensure the

control target can be fed into the simulation with the same time delays as the actual hardware, but with a positive phase shift. This allows the virtual robot to in effect predict the motion of the actual robot and enable the use of sensory fusion and virtual sensors discussed in Section 6.3.

4.8 Application to Quadruped Robot and SARM Hardware

4.8.1 FTDI FT232R considerations

The majority of motors and sensors used in the quadruped platform are connected to a FTDI FT232R based USB to TTL serial converter for communications, as only a small number of devices used a physical transport layer other than TTL compatible serial UART communication. This type of device is subject to the USB bus limitations on frame transfers as discussed in Section 4.5.4 above. In order to understand the full performance limitations however, the low-level specifications of the device must be discussed. The FTR232, like many USB communications devices uses device side and driver side buffers to smooth data transfers. The FTDI driver holds an accessible software buffer in the computer's memory for both transmission and data receipt, transmitting data to and from the hardware buffers upon request. The transfers are initiated by accessing software buffers using driver commands, `FT_Write` and `FT_Read` in the D2XX driver interface [53], and block execution until the command is executed. In the case of `FT_Write`, this block is minimal, as additions to the buffer do not depend on the transfer of data to the device. `FT_Read` however, the function will not return until data of a predetermined size is read into the buffer or the connection times out, which makes unbuffered or naive access patterns significantly slower due to USB framing. As such, the access patterns and connection parameters must be tuned to maximize data transfer and minimize independent data read commands [54].

In addition to the USB limitations, the hardware bandwidth may also need to be considered for highly articulated robots such as the quadruped platform. The hardware side includes independent transmit and receive buffers with 128byte and 256byte capacities respectively, which can be limiting factors in some situations. Using sequential write packets described in Section 4.8.1 in the methods described in Section 4.5.4, the robot motor commands for a single target update cycle require just 117bytes, 108bytes for write commands and 9bytes for a single read command, fitting within the hardware limitations. As previously discussed in Section 4.8.1, the standard read and write methods can be simultaneously used within the memory limitations of the hardware buffers, however the latency may be increased beyond the ideal scenario if the transmission rate is low enough that the hardware buffer is not exhausted before the next data update. For the currently used 500kBaud rate, the transfer of twelve motor write commands and one read command requires 2.34ms. As read commands block until a response is retrieved, the minimum control cycle is extended to four USB frames, or 4ms, resulting in a maximum theoretical open-loop control frequency of 250Hz. However, only a single response can be obtained in this time, resulting in an effective closed-loop control frequency of less than 20Hz for methods that use hardware motor data.

4.8.2 Dynamixel 1.0 protocol packet structure and optimizations

The most used packet structure in the current robot platform is that of the Dynamixel 1.0 communications protocol for Robotis Dynamixel servomotors. This protocol is shared by the two motor

types used for quadruped platform testing, the AX-12 and MX-28, as well as several other motor models by Robotis. Where possible, additional devices, including IMUs and other sensors, also use a compatible packet protocol to minimize the risk of errors while communicating over the same hardware channel.

This protocol, visualized in Figure 4.13, uses 8bit aligned binary characters to directly modify a parameter table in the RAM of the target servomotor within the Dynamixel system, or can be used to store binary formatted information of any type if used by non-Dynamixel devices. The packet payload is preceded by a header section comprised of a two-byte preamble followed by the target ID number, non-header packet length, and instruction type. The packet is concluded by a one-byte redundancy check which provides data validity information. The data section of the Dynamixel 1.0 packet is dependent on the requirements of the instruction type used, generally including the target addresses to read or write and the payload.

0xFF	0xFF	ID	LEN.	INST.	Value 1	Value 2	...	Value N	CRC
Header					Data				End

Figure 4.13: Generic Dynamixel communications packet structure. The specific order of data values depends on the instruction type used.

This packet structure is used for both writing to motors and receiving motor state information, with the instruction byte being replaced by error status information in the case of a reply packet. The structure of the packet allows motors to ignore both irrelevant and corrupted data, including data meant for other devices and reply packets from other motors without interference. Due to the target ID requirements however, only 254 devices can be connected to one communications line, with an additional ID reserved for multicasting.

As the primary servomotors, Robotis Dynamixel AX-12A and MX-28T, are both Robotis Dynamixel 1.0 communications protocol based motors, an intermediate implementation class called `DX_Motor` was used to minimize code redundancy and simplify edge cases involving Dynamixel 1.0 specific communications. Motor models and the communications for Dynamixel AX-12A and MX-28T servomotors were fully implemented in the framework, however device communication with additional models is possible due to the intermediate class supporting base communications with all similar motors.

The Dynamixel communications protocol offers several Dynamixel motor specific communications optimizations that use the multi-drop capability, the ability to split control signals through multiple devices without duplication, of the UART communications to update multiple motors with a single command packet. Although this creates unwanted communications dependencies, the performance can be improved by an order of magnitude over un-optimized communications, particularly in the case of read commands. In order to minimize the impact on the remaining system, these optimizations were implemented on the `DX_Motor` abstraction class initialization methods while utilizing the existing communications API and a new master and slave API which combines multiple simplifies packets into a single broadcast packet all robots can utilize.

Several methods were available for command synchronization within the Dynamixel 1.0 protocol [55]. In order to improve motor synchronization, the protocol offers two methods compatible with all Dynamixel 1.0 motors, a “reg write” method that stops commands from being executed until a synchronization packet is broadcast, and a “sync write” method that integrates all of the individual commands into a single packet. Although the “reg write” could easily be implemented with the existing methods by adding a dummy motor that only executes the synchronization packet, there are no guarantees that this packet would arrive before the built-in timeout of the motors, resulting in possible loss of control. Instead, the “sync write” functionality was chosen for its smaller overall packet length and guaranteed instantaneous execution.

As the amount of data sent in the command packets is reduced from 108 bytes to 43bytes, the performance benefit using this method is significant for the quadruped platform, with the transmission time reduced from 2.16ms to 0.86ms. The most important aspect of this change however is that the synchronization is guaranteed to within the interpreter execution time of the motor control loop, providing a hard real-time mechanism a significantly smaller delay than even interrupt based communications.

For this method, the write callbacks are passed through a master motor and formatted into the proper Dynamixel packet format before being set to the communications device. Existing packet building methods in the `DX_Motor` class and derived classes were reused to build the new packet format, however an additional binding step is necessary to bind the additional motors to the master motor.

Like the unified packet for writing, the read response rate may be improved with a mostly undocumented addition to the Dynamixel 1.0 protocol, a “bulk read” method available only on MX series motors. As with the “sync write” method, a single synchronization packet broadcast to all applicable motors can be sent that requests data from multiple motors and requires no additional user input to synchronize the response. The motors respond in the order defined by the read command packet, using packets equivalent to the standard reply packet. As with the “sync write” method, both the read and response are administered by a master motor instance that requests and returns data to the single packet paths as expected.

Using this method, the motor state information can be read from the motors at a rate that is significantly faster in the case of the quadruped robot with an USB to UART adapter. The entire data response can be returned in the same total transmission time of 3.16ms, however the 156 bytes can fit within the receive buffer and processed all at once. The total cycle, including the initial read command and motor responses can be completed in as little as 6 USB frames, compared to a minimum of 30 frames necessary with independent reads. When used in conjunction with the “sync write” method, the necessary commands are capable of being set simultaneously to the transmission buffer, allowing the number of required frames remain at 6 allowing control rates in excess of 150Hz.

4.9 Summary

A high performance low-level control and simulation framework was developed for modularized robot applications using low-cost components. Command rate performance was optimized using a mixture

of queues, callbacks, threading, and driver level optimizations while ensure that robot and simulations matched as closely as possible. Simulation integrated control capability was developed as a first-class citizen of the control framework, with the simulation aspects doubling as both virtual robots for offline experimentation and as a high quality in-phase state estimator. In addition to core framework components required to communicate with the physical robot and simulation, a significant number of features were added to allow for optimization, visualization, and data recording and analysis.

5 Physical Simulation

5.1 Goals and Requirements

The simulation subsystem was one of the core components of the low-cost robot platform design, providing both virtual robots for system testing and value estimation for robot-in-the-loop control. As the simulation was intended to be heavily integrated into the control framework, several features were required for optimal operation of the framework as a whole. The general requirements of a simulation system are:

- High performance, must support real-time simulation on portable computers
- High accuracy, must be able to replicate robot dynamics with minimal error, even with friction and unknown controllers
- Platform independence, should have minimal external library requirements and no dependency on hardware or operating system
- Simplified use, the system should be simple to implement, modify, and tune

In addition to these general requirements, certain technical requirements were necessary to integrate the control and hardware with the simulation. These included:

- Object oriented interface for the system, no non-static global variables
- Direct joint parameter control, joint motion must be externally controllable and must be triggered from the simulation side
- Support for shapes of arbitrary complexity, must be able to use standard mesh definitions

In particular, the object-oriented requirement is necessary for use in optimization methods, as any simulation must be entirely independent from other simulations. In general, physics engines are internally developed in object-oriented languages, primarily C++ for performance reasons, however some revert to C style interfaces with internal global variables for certain parameters for integration with other languages. To simplify integration into the control system, simulation methods that use C style object interfaces would require wrapper classes that abstract the method into an acceptable object form that may be managed more easily. Simulation systems that do not offer an object-oriented approach were not considered for the framework. Additionally, the use of global variables can limit multi-threading substantially or entirely prevent in-process multiprocessing depending on the parameters they control.

The joint parameter control is likewise mandatory in order to integrate the motor control and model with the rest of the simulation. In particular, the simulation side control function trigger is necessary to allow the simulation to be updated at the sub-step level when necessary, as there is no sub-step

synchronization barrier possible otherwise. In practice, most physics libraries implement this as a user modifiable callback function that executes during some phase of the sub-step.

5.2 Discrete-Time Rigid Body Simulation Engine

Two rigid body physics engines were considered for the core of the simulation framework required for the quadruped robot project. The ODE engine had long been used at the Laboratory of Smart Mechanisms [56] and many commercial applications [26] [57] [41] [27]. The engine is written in C++ but accessible only through C style interfaces to allow for high level integrations as a standalone black box library, with each memory struct representing a tangible object or connection between objects. Although it has been partially documented through published books and papers using the engine to simulate robots, the project has few resources outside of the source code to facilitate the integration of this platform into a larger system.

Despite its age, it is still regarded as one of the fastest and most reliable physics libraries for robotics, becoming the de-facto engine for ROS, Gazebo and DARPA robotics challenges [12]. A large part of this is the ability to step the system in two distinct ways, using the fast, sequential iterative stepping approximation dQuickStep function and the high quality full matrix inversion solution dWorldStep method. These system stepping functions can be used as needed to provide higher accuracy or higher performance within the same simulation at different steps, allowing for fast calculation of the system when there are few constraints, as well as more robust calculation when multiple constraints and motors are used.

An alternative engine tested for the simulation framework was the Bullet Physics Engine, which had not received significant academic use at the time but had been incorporated into a variety of 3D graphics and game engine software. Prior work in the development of a biomimetic robot snake locomotion technique used a python wrapped version of this engine for simulation of robotic gaits, resulting in reliable results in certain circumstances. The overall design of Bullet is similar to ODE; however the exposed API is entirely C++ with dependency hierarchies separated by concepts such as collision, constraints, and visualization. Unlike ODE, there are numerous sources of information on the engine available through an active support forum, including the derivations of contact and constraint equations and quirks in the implementation.

Due to more active development of the engine and broader audience, Bullet has a larger set of functionality and internal optimizations, particularly in the realm of constraints solvers and collision detection. Although the engine included only two solvers, an iterative solver similar to dQuickStep in ODE and a soft body solver when originally investigated, by the time an engine had been selected Bullet was expanded to feature over half a dozen solvers including a port of the dWorldStep solver from ODE and an experimental Featherstone solver based on mobilization rather than constraints.

The Bullet Physics Engine was determined to have the structure, performance, and capabilities most suited for the robot control framework integration. Although the engine was mostly used intact, several changes to components and default values were necessary to fully integrate the engine into the robot

control framework. The simplified API view of Bullet is shown in Figure 5.1, showing the core components of the library and the main interdependencies.

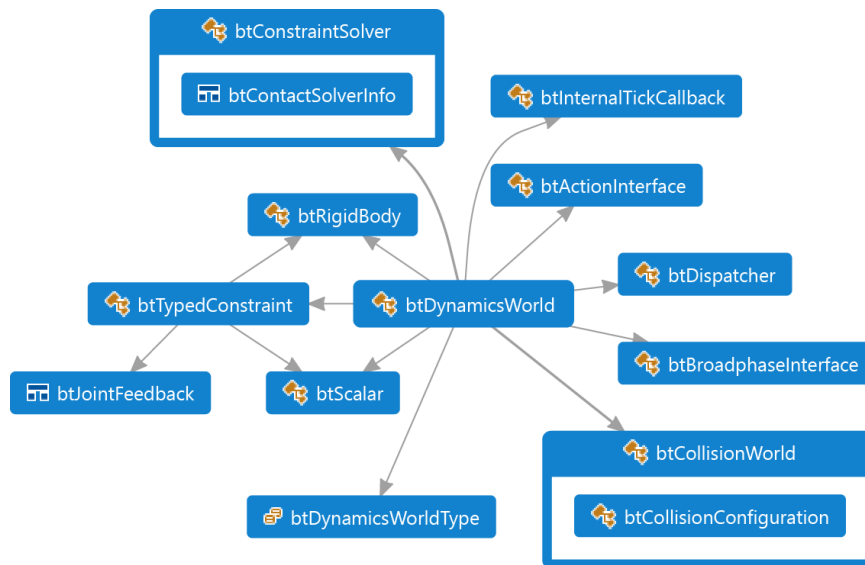


Figure 5.1: Bullet simulation core components. The *btDynamicsWorld* and *btCollisionWorld* are highly integrated, with the dynamics world requiring collision world information to function as more than a specialized MLCP solver.

5.3 Constraints Modifications

5.3.1 Rigid body interactions via Constraints

Within the Bullet Physics Engine, all interactions between objects are modeled as constraints on the bodies involved in an interaction. In simple terms, a constraint is a limit on the possible motion for a degree of freedom in motion between bodies, although the exact formulation of constraints can be considerably more complicated [22] [23] [30]. Simple contact is a one-dimensional constraint along the contact normal, while a hinge joint can be defined as five constraints in every direction, linear and rotational, except for the pivot direction. Although bodies have multiple, competing constraints, each constraint is limited to the body pair for which it was originally defined.

The calculation of the equations of motion for each joint connected body is achieved by formulating relative velocity constraints for each body pair dependent on the constraint type as a set of mixed linear complementary problems (MLCP). External forces are then added as bias factors within the MLCP, improving performance as only the sum of the external forces on each object are considered. These problems are then solved to within a desired threshold for each constraint using a variety of possible iterative and matrix inversion method constraint solvers. Generalized MLCP solvers can be used, however performance and stability gains can be obtained from a rigid body optimized solver with additional considerations for constraint tolerances and dynamics estimation [23].

A majority of physics libraries available, including Bullet, define constraints in the velocity domain, resulting in solution spaces that lie in the lines defined by Figure 5.2. As such, the constraints work on velocity error rather than position or acceleration errors, however the torque limit is computed in an acceleration domain due to the derivatives that comprise the constraint formulation.

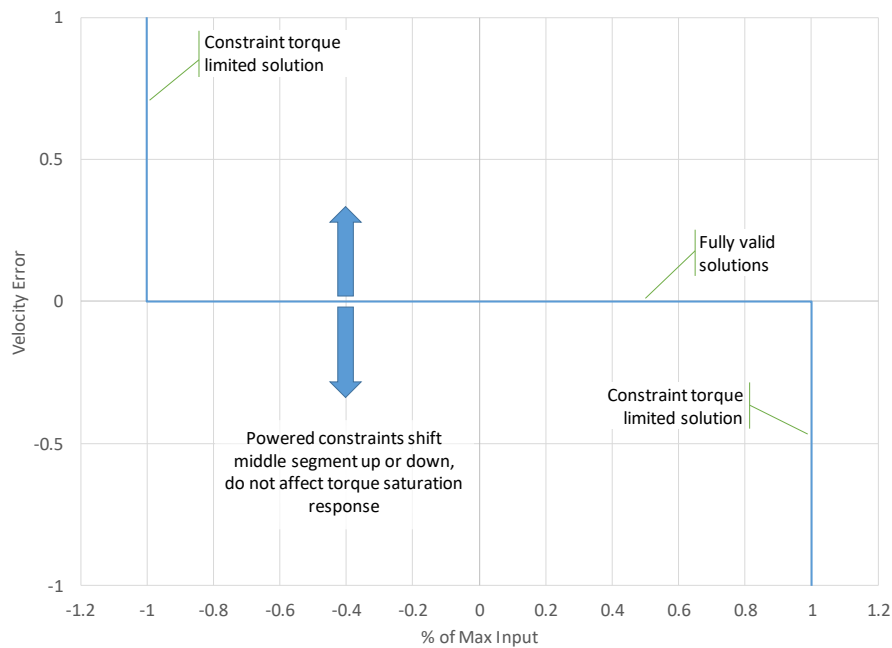


Figure 5.2: Bullet constraints operate on one of three solution spaces when resolved during a simulation step. The inclusion of additional error tolerances can change the result from the theoretical response.

A solution approximately equal to the lowest required input energy for the given constraints on a system is returned, however there is no guarantee that all non-redundant constraints will have valid results or that the overall input energy is the absolute minimum energy. A corollary to the lack of constraint guarantee is that for a finite maximum applied force, over-constrained and over-driven systems fail gracefully with one or more constraints increasing the required force for the remainder of the interrelated constraints. This property is often used to implement friction in several rigid body solvers [23] [22], however the method can be applied to constraints by setting a non-zero relative velocity to achieve motor-like dynamics [29].

Of particular concern is the hinge constraint type, a five or six DoF constraint set which allows only relative rotation about a pivot axis. Five DoF constraints are necessary to limit relative translation in all three axes and relative rotation in two axes. An additional constraint axis can be added to the pivot itself to limit rotation past certain joint angles.

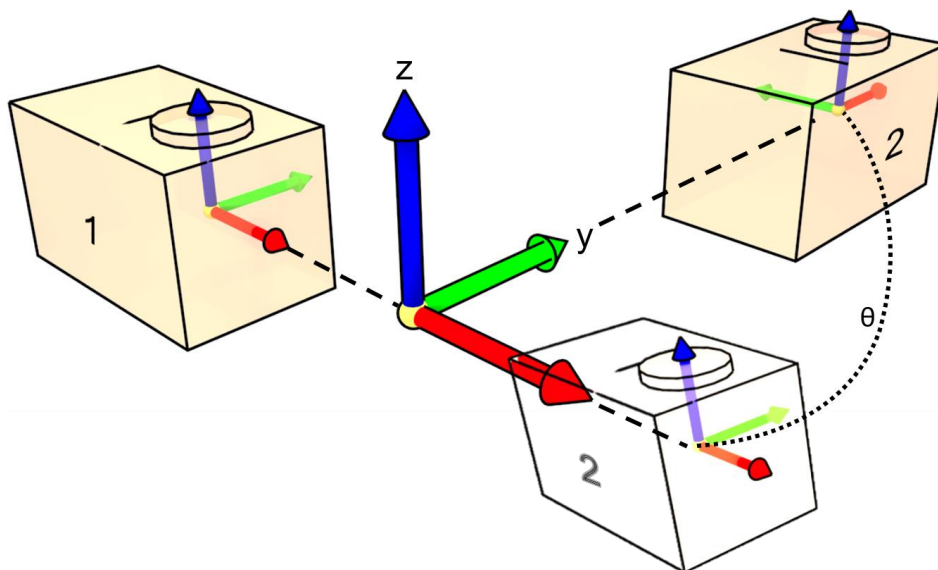


Figure 5.3: Local rigid body orientations and constraint based joint for use in rigid body simulation with a joint rotation about the Z axis by $\theta = 90^\circ$. Body 2 is shown with no joint rotation, the default joint definition, in white.

Although passive constraints can be useful in some situations, such as modeling doors and obstacles that move under external forces, motorized constraints are necessary to actuate robot joints. The formulation of motorized constraints does not vary significantly from the passive constraint with joint angle constraint, using the same generalized equations with non-zero and non-infinite components that represent the desired velocity and allowable torque respectively. As with the passive constraints, the constraints solver will attempt to correct the velocity error within the torque bounds through optimization, returning a new velocity approximately equal to the minimum energy delta to the desired system state.

The optimization method chosen can affect the transient response of the system, particularly when desired velocity changes are large with a low frequency iterative solver. Due to the high stiffness of the full matrix inversion method, the MLCP solver type produces results identical to the theoretical constraint shown in Figure 5.2 within a single time step even with non-infinite bounds on velocity and torque. This produces reliable replication of control signals every simulation step which allows standard control algorithms and motor models to function as intended. Iterative methods however suffer from two issues that affect the ability to implement the intended methods. Joint errors in iterative methods are significantly larger than those on stiffer ones due to the limited number of iterations possible in one step. Although the integration errors at any given motorized constraint are small, the order of constraint resolution may affect the overall robot pose. As the constraint errors affect both motorized and non-motorized constraints of the joints, it is possible for these errors to loop back through the non-motorized constraints in an uncontrolled way, particularly if error reduction energy is non-negligible.

5.3.2 Constraint error allowance and correction with CFM and ERP

A pair of hidden stiffness parameters, used to drive joint constraint errors to zero in the case of divergence, may come into play when computing both passive and motorized constraints. The Constraint Force Mixing parameter is an error allowance within the solver equations that allows for a non-negative tolerance to the velocity constraint. When integrated, this results in slow divergence of

the joint axis in the direction of each joined body. The Error Reduction Parameter (ERP) is a non-linear, non-physical energy adding term which acts as a spring-damper on constraint errors to drive them to zero in discrete time. This parameter is given in terms of percent correction per simulation step, and can be set globally or per constraint, including joints and collisions. These parameters together are defined in such a way that they can be configured as a position error based spring damper for each constraint axis using the definition:

$$\begin{bmatrix} ERP \\ CFM \end{bmatrix} = \frac{1}{T_s k_p + k_d} \begin{bmatrix} T_s k_p \\ 1 \end{bmatrix} \quad 5-1$$

The default values for these parameters are $ERP = 0.2$ and $CFM=0$, which causes stiffness issues when the simulation stepping rate T_s changes significantly from the default of $T_s = \frac{1}{60}s$. Figure 5.4 demonstrates an extreme case of the ERP and CFM scaling issue, where the default values are not sufficiently stiff to prevent contact penetrations or under actuation of joints at the default simulation stepping rate. Likewise, the correction values can increase stiffness, and therefore added system energy, into an unstable range where the simulation can experience unbounded growth in constraint errors due to overcorrection at each simulation step.

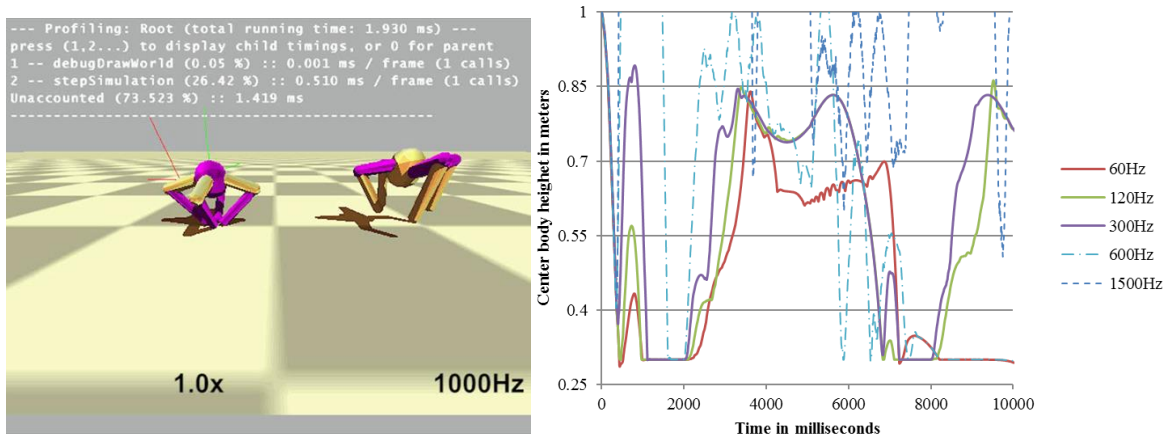


Figure 5.4: Motor simulation experiment for a simple quadruped. The simulation frequency response shows high variation due to ERP effects.

To avoid instabilities as well as under actuation, the ERP and CFM were configured to take in timestep independent stiffness parameters that had been tested on the simulation system to produce stable results with high stiffness. Although this can affect both iterative and matrix inversion solvers in different ways due to changes in overall system stiffness, the parameters found for integration into the simulation framework were capable of working with both solution methods.

5.3.3 Motorized Constraint Control Modes

The constraint definition allows for partial or total control of the target velocity and maximum allowable torque for a motorized constraint. These two potential outputs are sufficient to drive a joint three possible control modes, kinematic mode, torque mode, and lowest energy mode.

The kinematic mode can be achieved by using an infinite maximum torque and a specified velocity, resulting in motion equivalent to a kinematic move. As the torque is unbounded, the desired velocity

will be achieved within the tolerances prescribed by the ERP and CFM. This method can be used to implement kinematic approximations without additional development time, while eliminating orientation issues under external contacts often seen in a floating body approach [9]. When combined with output torque recording, this method can also be used to calculate the inverse dynamics equation for arbitrary configurations.

A torque command mode can be developed by setting the velocity to a large or infinite value in the direction of the torque, and the maximum torque is the desired torque. The velocity target cannot be reached with finite torque even with infinite time, resulting in a directional torque being applied to the joint. Most position error control strategies output a useful torque in the control loop, allowing this method to be used in dynamic models with non-trivial control schemes. The initial motor models for the robot platform in Section 6.3 used this method, however some shortcomings are present due to discrete time stepping and errors during solving.

Using both finite velocity and finite torque, a third mode of operation for motorized joints can be used. As both targets are finite, the constraint solver will attempt to optimize the joint in such a way as to minimize the total required input energy to the velocity limited or torque limited profile. This produces non-intuitive results during control when friction terms and non-trivial systems are used. With proper consideration for these effects, the system can be run with increased stability over the torque only method at lower simulation rates. A novel motor model utilizing this control mode is introduced in Section 6.4.

When multiple constraints on the same axis are used, the torque and lowest energy modes can be simultaneously used to drive the system to independent often conflicting targets. Any constraint with an infinite torque limit cannot be used in conjunction with other redundant constraints without affecting system stability. As the maximum input energy is bounded in this case, the system response remains bounded to a non-linear function of the two input targets. In the case of one of the constraints having zero velocity target and finite torque, the resultant constraint pair is equivalent to an object experiencing internal friction. When the velocity is set to the current velocity, the system is approximately under internal inertia of the joint separate from the inertia of each attached body, which can be used to implement pulleys and gears with non-negligible inertia. An example of this constraint pair is demonstrated in Section 5.5.1 to implement a stabilizing friction term to the joint.

5.3.4 Motor Model Hooks

In order for the internal motor model and control output to be executed properly, a hook into the constraints solver is required. Although precomputation is possible, adding the torque as an external force on the bodies before constraint calculation, a higher quality result is possible using a deeper integration that can run independently of high level calls and execute with every sub-step. Bullet provides such functionality through the constraint callback handle, a generic C-Style function pointer. This function pointer is called each time the constraint is recalculated during a sub-step, allowing for high precision control of the joint even in the case that multiple sub-steps are executed before the main function completes a single major step. Figure 5.5 demonstrates the general system stepping and constraint resolution graph, with the dark blue segments representing the callback hook.

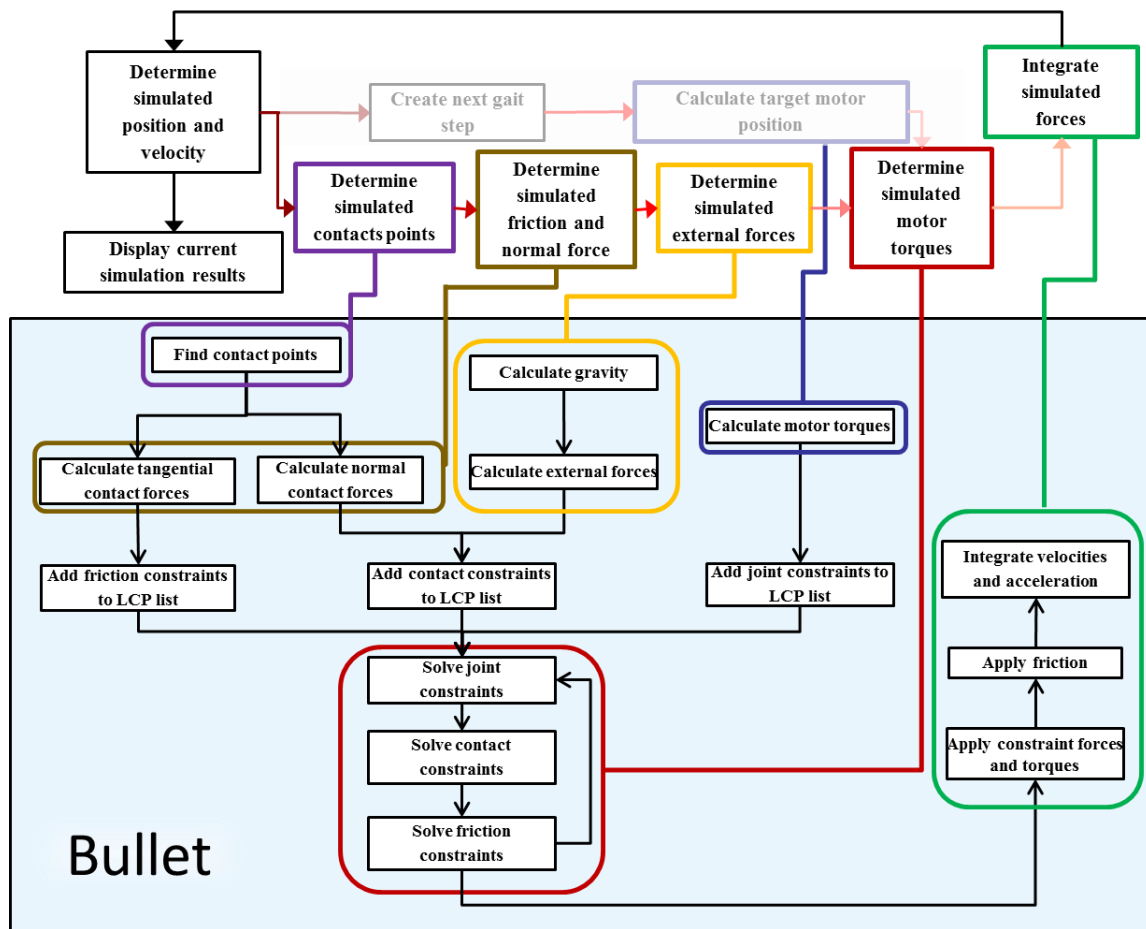


Figure 5.5: The simulation flow graph of a typical Bullet Simulation step with external motor model attached. The general program flow is given on the top, with bullet specific implementations on the bottom.

As Bullet includes a system timer that can be synchronized to external timers, this hook can provide not only the motor model outputs, but can also completely encompass the motor model and trajectory following equations. This allows the system to be easily integrated into the asynchronous robot control framework, as the motor model can request a new trajectory using the available internal clocks. Although this functionality was used in early versions of the simulation framework, more robust main clocks were necessary for the system to function, as described in 4.7.5 and 4.7.6. The internal stepper is however still used within the motor model to approximate time-delayed commands to better simulate real hardware.

5.4 Performance Considerations

5.4.1 Collision Body Simplification

When a robot is simulated with a floating reference frame, a large number of collisions are expected as the only counteracting force to gravity for the system as a whole are the reactive forces at the points where the robot interacts with a surface. These collision locations are constantly shifting as the robot moves, requiring constant recalculation of both the vector and magnitude of restitution forces. This in

turn requires a significant number of calculations, which can often be a limiting factor in simulation performance on robots with many legs and redundant contact points.

Collision detection in the Bullet Physics Engine is a multi-layered collection of objects and methods designed to optimize the algorithms required to check for the possible or actual collision of two or more objects within space, returning a manifold of points in the event of a collision for further processing. Collision shapes can fit into three main categories based on the shape characteristics as seen in Figure 5.6, each having an increasing computational complexity over the last:

- Primitives: Simple shapes with closed form boundary solutions. Includes boxes, cylinders, cones, spheres, capsules.
- Convex Hull: A space defined by a collection of flat planes that completely enclose the object without intersecting it.
- Triangle Mesh: A space defined by a collection of interconnected triangles surfaces with an orientation such that the surface perpendicular is defined with the positive z in the external direction.

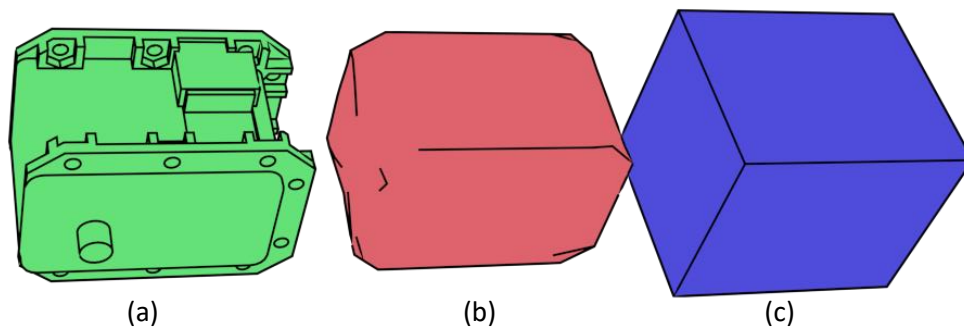


Figure 5.6: Collision shapes for a single motor module. The high resolution triangulated mesh output from CAD files is shown in (a), the convex hull of the triangulated model is shown in (b), and a bounding box approximation is given in (c). There is an order of magnitude of difference in the computation time for collisions at each simplification step.

Although the difference in model quality for the purposes of visualization is significant, the effective quality of the simulation is less dependent on highly accurate collision models over collision site local accurate models. As with many robot types, there is only a small portion of the quadruped robot that is expected to be in collision with objects at any time, generally limited to just the foot contact pad and occasionally the lower leg during inadvertent collision with non-ground obstacles. Rather than providing high vertex count triangle mesh data or high node count convex hulls for the end ball and the servomotor sections, it is possible to use a closed form defined ball and cylinder to not only more accurately describe the actual shape of the foot, but also reduce computational times by orders of magnitude. The compound primitive approach, as shown in Figure 5.7, creates a model of similar quality to the CAD model derived triangle mesh in the expected contact region while significantly reducing the complexity of non-contact areas.

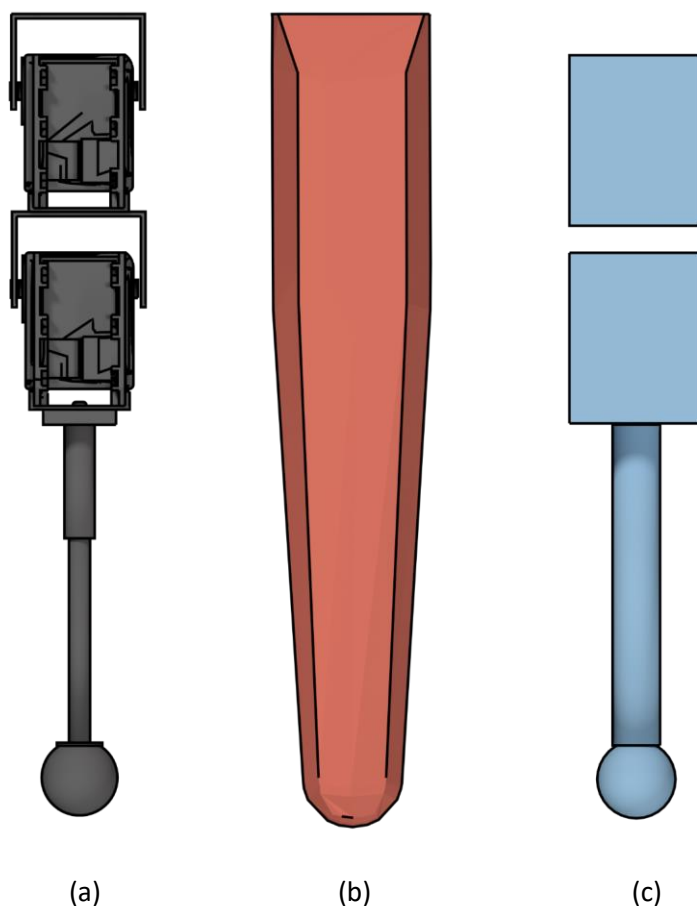


Figure 5.7: Collision shape representations of the lower half of the robot leg. The CAD model converted into a triangle mesh is seen in (a), while a possible convex hull representation is given in (b). The compound primitives shape in (c) shows a compromise between collision accuracy and performance.

Triangle mesh and concave objects in general can be computed in one of two ways. The first option is a dedicated mesh collision solver, in the case of Bullet this is the GIMPACT package, although many options exist [29][58]. These solvers generally execute a locally exhaustive search of mesh faces within a margin of an additional collision object. When comparing a triangle mesh with a less complicated collision object, the method may attempt to optimize the search algorithm in the vicinity of each triangle face, resulting in decreased performance for high quality meshes. In the case of objects colliding or nearly colliding in multiple locations, a single search can take several hundreds or thousands of microseconds to compute.

An improvement on this is the use of convex decomposition techniques and compound collision bodies. This method builds multiple, overlapping convex hulls that estimate the original concave triangle mesh. Although the computational costs in creating this compound collision object are high, amortization rate from the increased performance in collision handling between convex objects are high enough to often justify the mesh precomputation within a single simulation step. Currently this method is used to develop the ground plane for robot motion when a trivial ground model cannot be used.

Although it is difficult to estimate exact performance benefits in all situations due to the varying number of collision points and how algorithms respond to multiple, sometimes redundant collision, the scale of the difference in performance is significant.

5.5 Discrete-time simulation of modular robots using rigid body methods

Rigid body simulation is often formulated as a partially specialized constraints optimization problem, with a fitness function given by the change in system energy required for a solution to a given set of constraints. At any given time-step, a solution approximately equal to the lowest required total input energy for the constraints, including position and friction, on the system is used for integration of the next system state. However, for current solvers there is no guarantee that all constraints will have valid results or that the computed constraint input energy is the absolute minimum for a given constraint system. Additionally, in the case of constraint redundancy, any redundant constraint pairs are guaranteed to have at least one constraint invalidated in the case target velocities for both constraints are not equal.

For a finite maximum applied force, over-constrained and input energy limited solutions fail gracefully without adding an infinite amount of energy, ensuring system [23], with one or more constraints increasing the constraint force required to maintain a valid solution for the remainder of the interrelated constraints. This property is often used to implement friction in several rigid body solvers [22] [23], as well as powered actuation constraints to achieve motor-like dynamics by setting a non-zero relative target velocity [29]. In particular, servo-motor rotation joints can be given as a set of one powered constraint on the Z rotational axis with a prescribed velocity and five zero velocity constraints on the remaining rotation and linear axes.

In many cases, the internal inertia can be considered negligible or approximated by a saturation model in order to simplify the full body dynamics and control models of modular robots. For servo-motors used within modular robotics, the internal rotor and gearing inertia are often several orders of magnitude lower than the inertia of the motor module in its entirety. This assumption is verified by the results of the simulation conducted with small time-steps, as the simulation errors are approximately equal to the experimental data sampling errors.

When simulated with time-steps that are significantly longer than the electrical and mechanical time constants of the internal DC motor, the time necessary to achieve $1-1/e$ of the asymptotic value in a step response, the additional torque due to internal inertia and angular acceleration are negligible in comparison to external values. The angular acceleration of a joint between two steps can therefore be calculated by applying the motor output torque and external torques due to gravity and other sources, τ_m and τ_{ext} respectively, in conjunction with the motor friction τ_f to the overall module inertia J as given in Eq. (1).

$$\ddot{\theta}_m = \frac{\tau_m + \tau_{ext} - \tau_f}{J} \quad (1)$$

When a constraints-based rigid body package is used, an equivalent calculation is done during the constraint solving using the current step external torques and velocity to reduce integration errors. The joint angle is not directly controlled as in Eq. (1), rather it is computed as one of many constraints on the free motion of the lever arm, with the motor torque computed as either a torque or a non-zero

velocity constraint at the pivot for both bodies joined by a hinge joint. For the implementations of the constraints and solvers referenced in this paper, a simplified formulation of inter-body joints as constraints can be found in Section 3 of Catto's paper on iterative dynamics [23]. The angular velocity is solved using the input torques and angular velocity, given as the pair of generic equations Eq. (2).

$$\dot{\theta}_m = G(\tau_m - \tau_f, Z) \quad (2a)$$

$$\theta_m = F(\dot{\theta}_m, Z) \quad (2b)$$

The equation pair depend upon the internal state of the system, Z , which is inherent to the simulation method. The internal state of the bodies comprising a joint can be black boxed for simple control methods and the resultant joint properties can be used independently of this internal state.

Motor friction due to gearing and other sources is approximated as a combination of a velocity dependent dynamic friction component f_d and direction dependent static friction component f_s . We can use the current step velocity to estimate the current motor friction using

$$\tau_f = f_s \text{sign}(\dot{\theta}_m) + f_d \dot{\theta}_m \quad (3)$$

when the expected velocity change over the step is small. This simplified friction estimation is comprised of a static component in the direction opposite to motion f_s and dynamic component dependent on the current average velocity f_d , which are dependent on the internal imperfections of the motor. Although additional parameters to compensate for gear slip and external loadings could be added, their affects are negligible in comparison to the sampling errors.

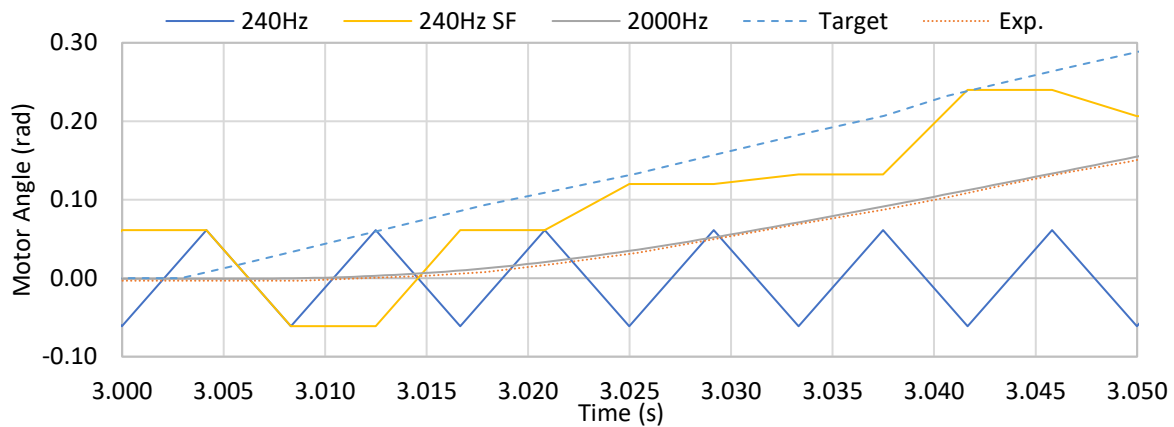
5.5.1 Stability considerations

Due to first-order explicit and symplectic Euler integrators used by real-time simulation packages like ROS, simulation stability is of significant concern in commercially available rigid body physics engines used in robotics. Simplified discrete time motor simulations often require smaller time-steps, at the cost of significant additional computational effort, to prevent controller and simulation instability that can arise from chattering resulting due to high gain controllers and discrete friction approximations [18]. In particular, servo-motors with low body and joint inertias compared to the high static and viscous friction and controller gains, such as those used in modular robots, are sensitive to chattering in response to the various inputs. When the instantaneous velocity is approximately equal to the current step velocity, any countering torque from the counter-electromotive force and friction will dampen the simulated system as expected from the continuous system model. If viscous damping is applied directly using this approximation at lower simulation frequencies however, the delayed damping responses can result in control instability or even overall simulation instability due to errors in the step integrated velocity approximation. Due to this and additional control approximations, the simulation stability of this method depends on the control and friction torques in respect with the inertia of the bodies attached.

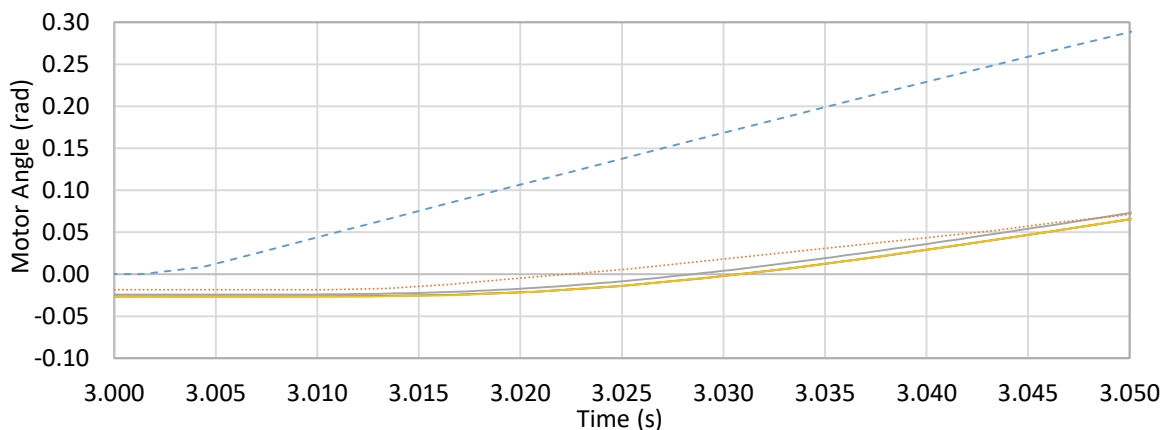
Work has been done in the application of stable viscous damping and mass ratio clamping to reduce constraint errors [41] [30], however the existing methods employ non-physical, solver specific modifications to the solved system state. Solver configuration dependent parameters are not readily

portable even within the same rigid body dynamics framework and may not work with systems that use a mixture of solvers within a single simulated environment. Similarly, stable integrators for rigid body simulations have been developed [41] [59], but the additional computational cost associated with sub-sampling steps can be significant enough to outweigh the performance benefits of larger simulation steps compared to smaller steps in a symplectic Euler method.

In certain cases, it may be possible to compute the components of the motor friction separately (SF) from the input torques as an additional constraint on the system to reduce the effects of non-linear friction that may increase instability [50]. The discrete time approximations with larger step sizes can become unstable when system inertia is low, as seen in Figure 5.8 (a), even in the case that simulations at higher inertias are indistinguishable between various step sizes as in Figure 5.8 (b). As the viscous friction is a property of the system itself, a system experiencing these instabilities would require a smaller step size or the use of an alternative, non-physical damping method to improve the velocity estimations.



(a)



(b)

Figure 5.8: Instability due to low system inertia when using low simulation frequencies shown in (a) can result in significant joint state errors, even in the case that increased system inertia produces stable simulations, seen in (b). System was tested with a $\frac{4}{\pi}$ Hz sine wave starting at 3s using the electro-mechanical model described in Section 6.3 at 240Hz and 2000Hz, and using a separate friction (SF) implementation at 240Hz.

Hsu et al. [41] mitigated this instability due to discrete-time approximations by using implicit joint velocity smoothing that replaces direct friction computations with non-physical damping. The resulting implementation of the joint parameters is highly dependent on the simulation software used and even software internal parameter changes including simulation step size and solver method. As such, any change in simulation parameters may necessitate recalibration of the joint parameters to maintain accuracy. Additionally, this method is limited to viscous damping, and is unable to approximate friction curves with static components that can arise in certain motor types.

In past experiments, we have minimized the instability by separating the driving torque and friction into an applied torque and zero velocity constraint [50], where the system optimizes the constraint pair to ensure friction is always removing energy from the system. This separation enforces that the friction component be applied in a direction counter to instantaneous motion at all times, regardless of the velocity direction in the current and past steps. The stability in control however is limited by the amount of torque applied in comparison to the maximum allowable friction for a given system inertia, with low viscous friction failing to stabilize the simulation, particularly if the controller gain is high. Likewise, high viscous friction can create velocity instabilities, where the velocity alternates between a high value and zero, as can be seen in the 240Hz SF result in Figure 5.8 (a). Although permanent joint control failure is avoided, the accuracy of this method drops substantially at low simulation rates with low inertias. Due to the additional computational effort necessary to process the additional constraint, single step performance is decreased over the direct implementation. However, as the simulation of the system already contains a set of six constraints per joint, the additional of a single redundant constraint is small and often acceptable due to the increased stability.

5.6 Real-time visualization capability

The simulation module may be executed asynchronously to the control and communication, in real time, using either the Bullet internal visualization tools for timing control or an externally timed execution routine. Although a majority of the work completed with the Bullet engine ran too fast to enable real-time visualization, the primary simulation used during robot-in-the-loop control benefits from visualization to ensure that the simulation and robot look identical to the operator.

The original purpose of the Bullet Physics Library was for animation and as such the simulation is by default tied to the graphical rendering rate of the system. Within Bullet, this is done using the OpenGL based GLUT library timer callback methods synchronize simulation steps to the render buffer update to ensure that the graphics components have the most up-to-date information possible. As the number of simulation steps per update is made dependent on the render step size, the ratio between the render frequency and simulation frequency may change according to the rendering complexity with low repeatability. When implemented in this form, several simulation steps may be executed in quick succession followed by a significant delay until the next batch, effectively reducing the closed-loop control frequency of the simulation when a physical robot is also connected.

The motor control loop is designed to be simultaneously executed for both the hardware device and simulation, which is incompatible with the visualization synchronized method and can make the simulation become unstable when the number of control actions exceed the simulation buffer size built

into the motor classes as discussed in Section 4.3.1. In order to mitigate issues due to the irregular execution timings when visualization is used in the loop, the servomotor class internal command buffer, normally used to replicate communications latencies, can also be used to compare commands to a simulation step clock, approximating command delays which would occur in a real-time response. Although this limits the effects of the command and simulation latencies, a sufficiently fast command rate can result in dropped commands and desynchronization from the hardware.

To ensure that the simulation has the most up to date target information, the control system is integrated at the simulation sub-steps, updating control commands based on each discrete time increment in the simulation. This is not however an ideal solution, as the robot control frequency for the physical robot platform is effectively reduced to the visualization refresh rate as the commands are calculated every visualization frame rather than the simulation rate. For early demonstrations of the robot platform this allowed the robot to successfully use feed forward joint commands, however more complicated motion could not be reliably achieved.

An alternative solution to the system stepping issue when visualizations were enabled was developed to avoid the issues due to a visualization oriented timing approach, enforcing that rendering and simulation execution run at optimal timings. Control of the simulation and motor targets was moved to an independent thread, which executes functions related to platform control, simulation, and data recording in sequence at a prescribed time interval. The visualization loop is then run in a separate thread controlled by the OpenGL backend, which takes ownership of the thread and only returns upon program exit.

As the visualization routines require simulation data from a single frame, as opposed to partial data from several frames, to display the robot state without graphical artifacts, a memory synchronization barrier to prevent the use of partial states is required. A mutex barrier is implemented in the simulation by way of a doubly inherited class that has the functionality of both the simulation and visualization subroutines as shown in Figure 4.11. With this method, simulations can be executed at precise timings independent of the platform and execution of visualization. Although priority is not set for a collision case, the visualization access to data is significantly shorter than the simulation, minimizing the impact on the simulation timing. Delays caused by an executing simulation do not affect the usability of the visualization, as short interruptions will only cause the display buffer to retain the previous data rather than return an empty or corrupted image.

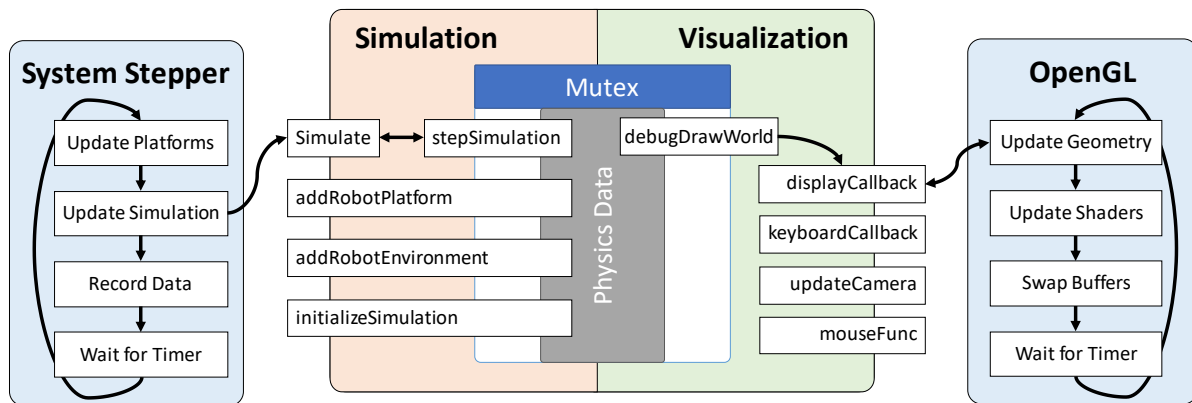


Figure 5.9: A simulation and visualization class with mutex barrier between the simulation and visualization. External triggers call the functions responsible for simulation and visualization independently and may cause a collision.

5.7 Faster than real time methods

In addition to real time simulation mode, the simulation can be run much faster than real time when in a sub-network that does not include a physical robot or visualization. When this is desired, a synchronized discrete time stepper for the simulation and remaining control framework is required, allowing the system to run without a real-time clock. The performance of this type of system is dependent on the robot being simulated and its control algorithm, however the independent nature of such a system allows multiple versions to be simultaneously calculated on individual computer cores. The optimization techniques discussed in 4.7.3 above take advantage of this faster than real-time support on multiple threads to enable computation above 175 times real-time in consumer desktop computers, allowing the use of optimization based observers using fully simulated systems.

Although a majority of Bullet was capable of simultaneous simulations in a single process, several changes were necessary to properly implement the multi-threaded simulation pool. The largest concern was a series of bug fixes regarding the `btDantzig` and `btMLCP` classes due to the use of global static variables during computations of the matrix inversion. In rare cases, there would be collisions where the calculations on two threads overlapped in such a way as to corrupt the constraint matrices and cause soft-failures of simulation where the calculations complete without error, but the values returned were off by minute amounts. These changes were later incorporated into Bullet 2.82+. Additional private functionality was added to bullet to improve the system reset and minimize resource allocation, including constraint reinitialization.

5.8 Summary

A generic rigid body constraints based simulation package can be utilized to provide high quality estimation of physical motions in open loop. In particular, internal representation of joints in such a package lends itself to the integration to modular robots and their control framework. The modularization process forces single joint per actuator conditions on the hardware, and likewise the simulation methodology limits joints to two body connections, a condition that may cause difficulties in the simulation of non-modularized robots. As such, the Bullet Physics Engine, which uses a

constraints based method for simulation, was utilized as the backbone of physical simulation for the framework.

Although there are significant benefits to the rigid body simulation packages discussed, integration into the control framework requires tuning and additional functionality not originally present in the package. Among these are custom solutions to graphical visualization and motor control during simulation, both implementing a multithreaded approach to improve the asynchronous update paradigm of the control framework. Additionally, the applicability of traditional simulation models for motors is of great concern, as the discrete stepping of the system can induce simulation instabilities when steps are too large for a given control parameter set. Much of this instability was minimized through collision object and global constraint parameter tuning and modifications, however model based instabilities, further discussed in Section 6, were required to produce a viable simulation.

6 Motor Response Simulation

6.1 Introduction

The physical simulation of the motors discussed in Section 5.5 is limited by the unknown motor control actions that govern the actual joint accelerations and torques. This chapter will introduce current and novel methods of both simulating and calibrating motor responses within a simulated environment. Motor simulation methods can be designed based on required accuracy, performance, and simulation stability, and each aspect will be discussed in terms of the simulation library outlined in Section 5 and the control framework in Section 4.

6.2 Simplified Motor Approximations

6.2.1 Kinematic Approximations

Kinematic methods have long been applied to robotics to estimate the robot state under the assumption of perfect motor tracking of any input command. A naïve implementation of this method can only properly be used when at least one body within a joint chain is fixed in position, with additional body positions determined entirely from the joint angles. This method can be solved with little effort when closed form kinematics for a specific platform are given, or through linearized approximations. In the case of a linearized approximation, this can be done by propagating equations 3-1 and 3-2 through the fixed body to each leg and body. An example of simulation errors at low load is shown in Figure 6.1, where the position errors are in the same order of magnitude as the input and experimental positions. This naïve method shows much more significant errors at higher dynamic loads for geared or direct drive DC servo-motor based modules, with a motor loading of 700g showing an error of half of the input amplitude.

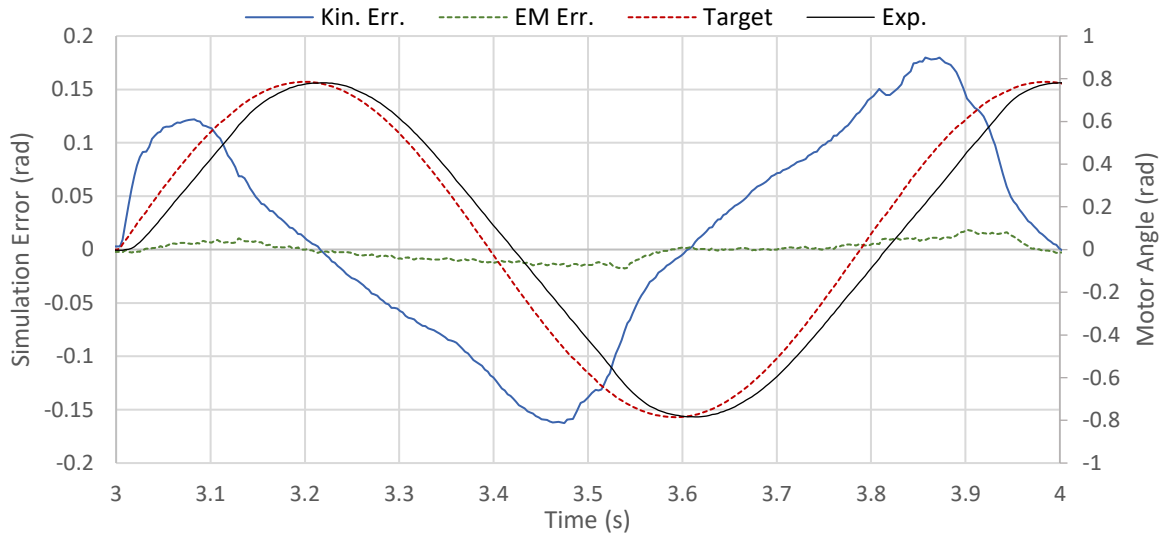


Figure 6.1: Simulation error of system using a latency compensated kinematic approximation. The original target and experimental data are plotted on a secondary axis. A dynamic approximation is shown (EM) for comparison.

The naïve implementation unfortunately does not work well in motor bounded motions, where the performance of the motor is not substantially higher than required for dynamics of a viable motion. A bounded velocity approach is the simplest system model that can be used for simulation with non-trivial control. In this method, a velocity function is developed for each joint to achieve a target joint position change within a reasonable timeframe. By linearizing the system motion with a first order velocity component, imperfect motor responses can be easily added. This avoids the discontinuous actual position in pure kinematic approaches like the “virtual chassis” [60] when the target position is discontinuous.

6.2.2 Saturation limit control

A more physically applicable system that can be trivially implemented in the current physics simulation is torque limited velocity targets for dynamics simulation. This method takes advantage of the properties of rigid body constraints to ensure that the motor output remains in a feasible range limited by its no-load velocity and stall torque. An example control surface for the naïve implementation of the saturation limited control is shown in Figure 6.2. This saturation limited control method provides the basis for the more robust conditional constraint method in Section 6.4.

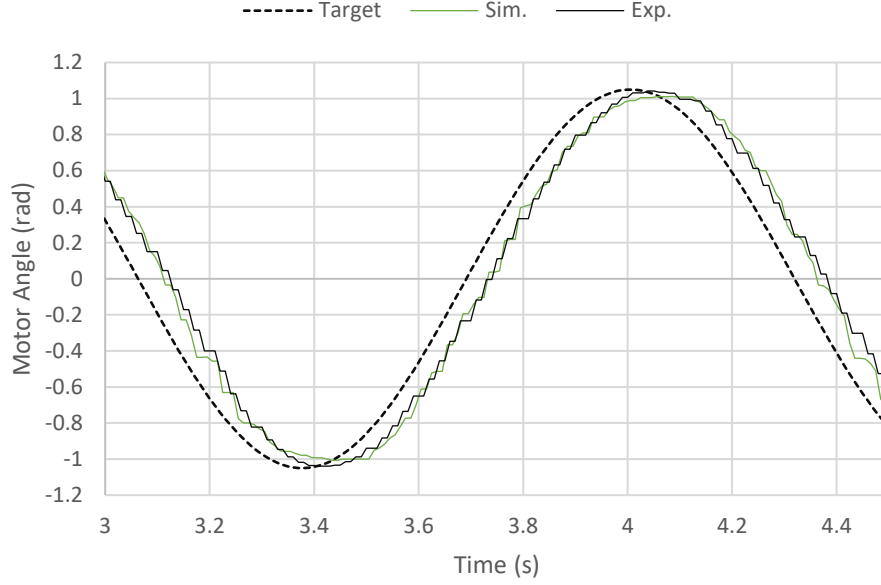


Figure 6.2: Saturation control of simulation during a bounding gait. Sampling limitations in the raw data leads to periods of no data between saturated motion response.

6.3 Electro-mechanical Torque Sub-simulation

For most robotic systems using DC motors, direct calculation of control actions on a voltage level for an equivalent motor subassembly often provides very accurate state estimation that can be used to estimate motor torques and velocity changes for the subassembly. Although complex models are available to compensate for secondary electro-mechanical coupling effects including motor winding characteristics and gearing imperfections, these factors can often be considered negligible even when milliradian precision is required. This simplifies the electro-mechanical state equations to variables that can easily be obtained from specification sheets to within a reasonable margin of the build differences between individual subassemblies. In the case of highly geared motors, this equation can be further simplified by assuming any position dependencies can be averaged out over the given time step, removing the position dependency of torque generation and requiring only a knowledge of the timestep average current and electromotive voltage potentials. When time steps are relatively small in comparison to the maximum motor frequency change rate, motor velocity $\dot{\theta}_m$ and the target voltage V_{target} can be approximated as constant values and the electro-mechanical state equations can be given by Equations 6-1 and 6-2.

$$\frac{\delta i}{\delta t} = \frac{V_{target} - K_s \dot{\theta}_m - R_a i}{L_a} \quad 6-1$$

$$\tau_m = K_t i \quad 6-2$$

The average current through the motor windings is given by i for a system with a counter-electromotive force constant of K_s , motor induction L_a , and terminal resistance R_a . The current produced by the motor electrical state can be converted to the output torque τ_m by the motor torque constant K_t . Integration

of the $\frac{\delta i}{\delta t}$ component can be partially accomplished by a variety of functions including the Runge-Kutta method or a subset of the Euler method, however the integrator time-step is often more crucial to the stability of the system. Additionally, it may be difficult to obtain an accurate $\dot{\theta}_m$ without conducting a full system step, and approximations to the current $\dot{\theta}_m$ must be made without additional external state information. In the case of the simulation framework, options exist for both the Euler integrator and Runge-Kutta fourth order integrator with fixed and separately stepped $\dot{\theta}_m$ estimator.

Servo-motor manufacturers generally do not disclose the physical and electrical properties of the DC motors used by their servo-motors, requiring some level of reverse engineering of the device. Examination of the internal components of the servo-motors used was required to determine the electro-mechanical motor model parameters previously stated in Table 3.3. The controller algorithm used in the Dynamixel MX-28 servo-motor used were provided by Robotis, however additional fitting is required to convert these gains to physical voltages which control the motors. The final value of the motor inductance however was allowed to be modified to account for the slew characteristics of the H-bridge, line driver, and microcontroller, as well as to increase simulation stability in the case the slew rate does not limit the effective velocity gain to physically plausible levels. When properly tuned, a limited slew rate does not significantly affect high load motor responses, however the low load response can be made significantly more stable at an internal and external simulation rate of 2000Hz.

The control action on the current loop, input as voltage V_{target} , may be provided by the servomotor manufacturer as a user modifiable PID controller with gains k_p , k_d , and k_i . When the actual motor angle θ_m and target motor angle θ_{ref} are known, a simple PID controller with voltage output can be given by Equations 6-3 and 6-4. As physical systems have control voltage limitations, this control action is bounded to V_{max} , which is often equal to the line voltage of the system being powered.

$$V_{target} = k_p e + k_d \frac{de}{dt} + k_i \int e dt, \quad -V_{max} \leq V_{target} \leq V_{max} \quad 6-3$$

$$e = (\theta_{ref} - \theta_m) \quad 6-4$$

The high gearing and low-quality motors used have a significant control region that can be dominated by friction. Although there are several models of friction that can be interchangeably used within the motor simulation [61], a Coulomb-Viscous friction model was found to be a reasonable model with fewer variables for identification than a Stribeck model for lubricated components.

$$\tau_f = f_s \text{sign}(\dot{\theta}_m) + f_D \dot{\theta}_m \quad 6-5$$

This model can estimate friction with only a velocity input and two identifiable constants, the maximum static friction f_s and the velocity proportional component f_D . Friction is unfortunately a highly non-linear function due to the sign function, and even small phase errors in calculation can create unstable systems. The electro-mechanical model as implemented in the control framework is particularly

sensitive to friction during dynamic motions due to the limited feedback for the motor velocity component $\dot{\theta}_m$.

Within the simulation package, the full motor model implementation is shown in Figure 6.3 with the actual joint position, velocity, and actual torque computations treated as a black boxed transfer function. The target position of the motor θ_{ref} is used to find the system angle error and error rate change, which are converted to a motor torque through the motor sub-simulation transfer function, which is then combined with a previously computed friction value. This torque is then directly applied to the required bodies within the simulation plant to produce the subsequent angle and angular velocity states. As a result, the effective equivalent integrator for this type of system when friction plays a large role can be considered a first order forward Euler. Due to this, the overall system is less stable despite the increased accuracy within the controllable region as discussed in Section 5.5.1.

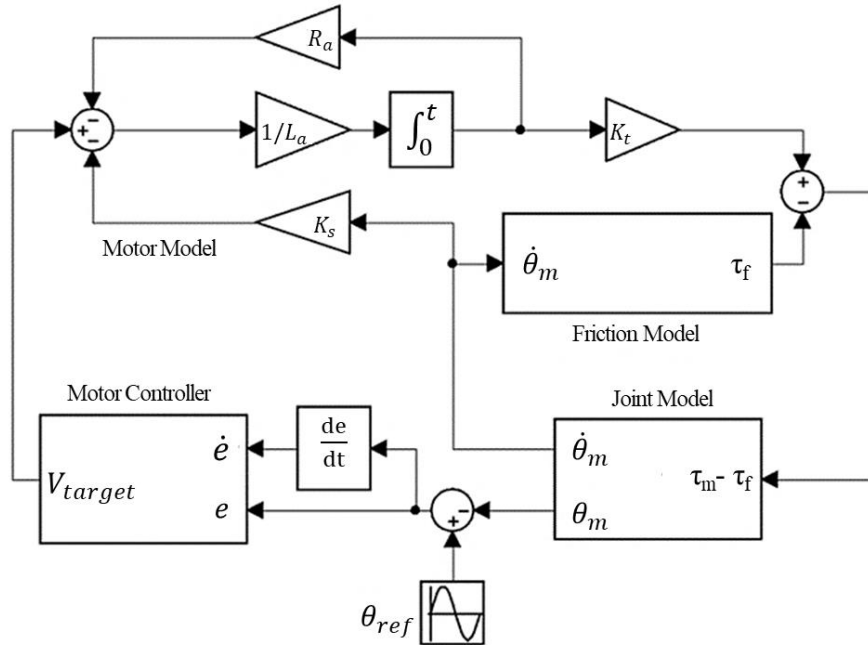


Figure 6.3: Electro-mechanical motor model for a servo-motor sub-simulation within a rigid body simulation package. The friction approximation, motor controller, and motor model can be used with a black boxed joint model.

To avoid estimated velocity based friction errors, super-positioning of forces can be implemented by applying friction directly on the joint as an additional system constraint that shares axis definitions with the primary joint definition. As the velocity and joint torques are calculated concurrently, this additional torque can be matched to the actual friction values more closely, always in a target velocity direction opposite to the next step velocity. This is effectively a semi-implicit integrator for the friction terms, which can improve overall system stability over the direct application of friction in the primary joint.

6.4 Conditional constraint method

Additionally, the traditional electro-mechanical model of servo-motors requires knowledge of the hardware specifications of both the motor and gearing mechanisms for accurate simulation, however this requires extensive reverse engineering of the mechanical and electronic hardware, as well as any software based controllers, to produce a viable model. Although it is possible to estimate values based on system responses in various conditions, several of the variables are redundant and increase the difficulty of finding unique and accurate solutions that match the module dynamics. Our constraint based method treats the module as a grey-box, with a known control target but only a generalized internal model, in order to allow new components to be identified without knowledge of the exact specifications of the internal controller, gearing, and motor characteristics. The methods for calculating the model parameters in such a system will be discussed in Section 7.2.

6.4.1 Constraint based stability improvement

Although using a separate friction component is often an acceptable solution, further improvement to simulation stability and accuracy can be achieved by applying lessons learned from the saturated limit control and separated friction models. When the system is modeled within a constraint based physics environment, the properties of rigid body constraint resolution can be used to produce a simplified approximation with higher stability when time-steps are large. As discussed in Section 5.3, constraints in rigid body simulation can be violated without significantly affecting the stability of other constraints with finite torque limits. In this superimposed state, constraints group will tend to apply only the minimum input energy required to achieve the highest energy differential target velocity so long as the constraint is valid. Both velocity and maximum energy bounds can be configured at each time stepping to ensure that the simulated motor does not exceed physically plausible velocities due to simplified approximations of the internal motor dynamics and friction during large time-steps, while retaining the effects of motor loading and overdriving from large externally applied torques. This model can be separated into two main components, a method of utilizing non-linear constraints utilizing a sub-simulation as a state predictor and a simplified motor model which can more easily be identified by optimization.

6.4.2 Conditional Constraint Method

Typical use of the constraint either controls the velocity, as in pure kinematic methods, or the joint torque, as in standard electromechanical models. The constraints discussed in Section 5.3 can be configured per simulation step with both maximum allowable torque and velocity targets for the joint defined by the constraint. In order to use both targets effectively, a redefinition of the motor model is required to separate internal effects from the external ones. This new model defines an ideal internal state for the motor, ignoring external torque sources, which can provide a maximum upper bound for the motor velocity that can be achieved through control actions in the motor.

$$\ddot{\theta}_{ideal} \cong \frac{\tau_m - \tau_f}{J_{int}} \quad 6-6$$

$$\dot{\theta}_{ideal} = \dot{\theta}_m + \int_t^{t+1} \ddot{\theta}_{ideal} \quad 6-7$$

The ideal acceleration $\ddot{\theta}_{ideal}$ can be considered a no-load approximation of the servomotor motion profile, comprised of the drive torque τ_m , internal friction torque τ_f , and internal inertia J_{int} . This internal inertia includes both rotor inertia terms and the gearing inertia, but does not include external inertias. Concurrently, the ideal velocity $\dot{\theta}_{ideal}$ can also be computed as the integral of the ideal acceleration with the currently known velocity of the motor $\dot{\theta}_m$. When the time steps are small and errors due to external sources are large, this integral can be achieved to reasonable accuracy with a first order forward Euler approximation.

The motor velocity actually achieved is the optimized result of the system solver minimizing the difference between actual acceleration $\ddot{\theta}_m$ and ideal internal acceleration $\ddot{\theta}_{ideal}$ given a maximum input torque τ_m . As the motor velocity is bounded to this target velocity, numerical errors due to low simulation rates are reduced in the case of low link inertia, such as the foot link state during the non-contact portion of a leg motion.

An added benefit to this velocity limitation is the ability to reject overdriving of the motor due to a large external force in the same direction as the estimated velocity. In a standard torque method, the external and internal torques would add a significant acceleration to the object which would often be corrected by the motor controller and motor model with smaller time steps. As the final torque direction for the motor joint is calculated during the constraint solving step, a value up to the desired torque can be added in a direction opposite to the external torque, limiting the instabilities in simulation that can arise from large instantaneous changes in external loadings often seen during foot placement. For small time steps this can be a significant source of simulation errors, however the simulation stability is greatly increased.

Although a constraint target limited to a velocity given by Eq. 6-7 provides stability improvements over the direct torque application, the system can still experience bounded oscillations, due to non-linear friction and friction-like velocity dependent terms which may cause sign changes between the current and predicted velocity states. This chatter proportional the controller's first order error term, the controller K_p , and inversely proportional to the inertia. Low system inertia can show significant oscillations as the inertia approaches zero and the system experiences an impact. As the robots serviced by this framework were expected to operate in motions that may produce such impacts, a method of reducing the strength and occurrence of the oscillations was necessary.

Chatter can be further reduced by applying a sliding mode like controller to the motor model with a sliding plane dependent on the expected velocity and type of input torques. By conditionally modifying the target velocity between two states, a friction dominant case when the friction exceeds motor torque and resultant target velocity changes the sign compared to the actual velocity, and a general case when the motor control laws do not expect significant impact to velocity from friction alone. The friction dominant target velocity in a constraints based rigid body method [22] can be considered a zero target-velocity constraint rather than just a force counteracting the instantaneous velocity. By switching constraint target velocity between the ideal no-load velocity and zero based on the composed torque and expected velocity changes as given in Eq. 6-8, a sliding mode controller like damper for systems can be designed.

$$\dot{\theta}_{\text{target}} = \begin{cases} 0.0 & \text{sign}(\dot{\theta}_{\text{ideal}}) \neq \text{sign}(\dot{\theta}_m) \text{ and } |\tau_m - \tau_f| < |\tau_f| \\ \dot{\theta}_m + \Delta t \left(\frac{\tau_m - \tau_f}{J_{\text{int}}} \right) & \text{else} \end{cases} \quad 6-8$$

The motor model independence of the conditional constraint method is maintained by stabilizing the simulation without the use of implementation specific correction factors typically used in robotic applications [41]. The conditional constraint can be implemented as shown in Figure 6.4 with minimal additional impact to the code base outlined in Section 5.3.4.

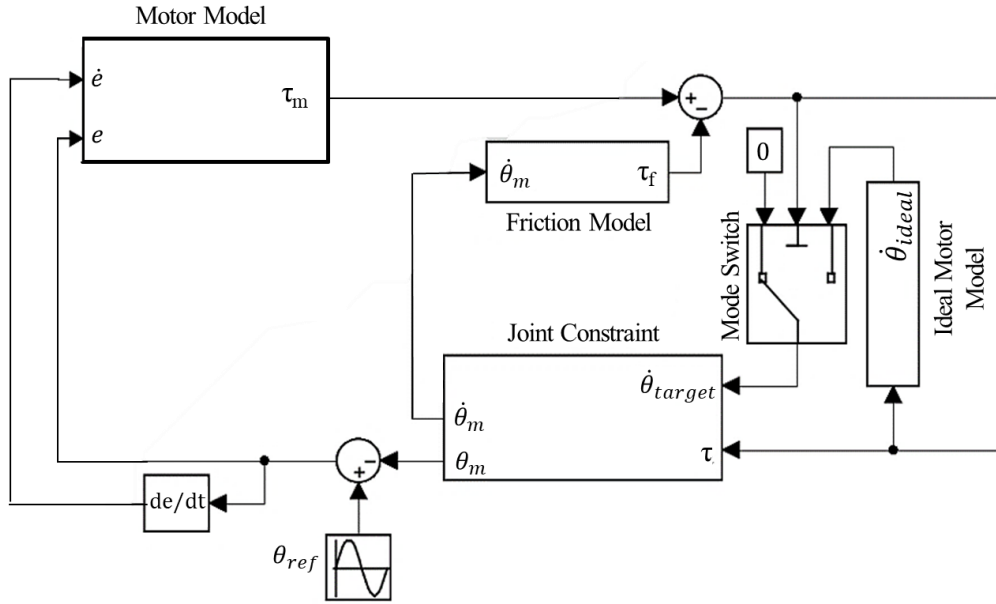


Figure 6.4: Simplified conditional constraint method for servomotor simulation. The ideal motor model and mode switch can be implemented independently of the motor and friction models as a secondary state predictor.

This model bounding and stabilization method is effective at preventing simulation stepping errors due to large time-steps in discrete time simulations when external inertias are low in highly geared motors with non-negligible friction. Underdamping of velocity for quickly increasing external loads can be minimized when external loads are within reasonable bounds for the driving motors. If the external vibration frequency is below the Nyquist frequency of the simulation, the system has a finite time impulse decay to near-zero velocity, ensuring simulation stability while the control action of the motor model is itself stable. However, as the system is controlled by a discrete sliding mode operation, decay to exactly zero velocity is not guaranteed, especially at higher loads where control actions may not counteract external forces.

6.4.3 Simplified controller for optimization

To more easily estimate motor parameters when the motor internals are not known, a simplified motor model is beneficial. A generic linearized motor model can be used in place of the electro-mechanically derived motor model when the controller output is linear with the motor model output. For this simplification, a servomotor torque τ_m is assumed to be a dynamic system that has a torque response with a sigmoid shape, where the inputs are given by a known ideal torque response $\tau_{\text{controller}}$ producing an equation of the form:

$$\dot{\tau}_m = \alpha (\tau_{controller} - B_m \dot{\theta}_m - \tau_m \gamma) \quad 6-9$$

As most motors have a steady state torque limited by their velocity, a damping $B_m \dot{\theta}$ is introduced to limit the torque based on a negative feedback of velocity due to various viscous forces. The overall shape of the sigmoid can be controlled by parameters α and γ , which control the asymptotic limit and settling time respectively. This equation can be considered a modified form of the Richards' equation where $v = \frac{1}{2}$ [62]. In the case the current control loop of the servomotor sampling rate is equal to or greater than the voltage control loop sampling rate, the new model is also exactly equal to a torque normalized version of the standard electro-mechanical model. In practice, Eq. 6-9 can be further simplified, as B_m related damping effects can often be integrated into the external friction component in the overall joint dynamics rather than in the motor model level. The equation for a PID controller in the torque domain is equivalent to the voltage domain Eq. 6-3.

$$\tau_{controller} = k_p e + k_d \frac{de}{dt} + k_i \int e dt, \quad -\tau_{max} \leq \tau_{controller} \leq \tau_{max} \quad 6-10$$

The gains k_p , k_d , and k_i are scaled versions of the gains in Eq. 6-3, however due to their interaction with the damping terms B_m and γ it is often beneficial to treat the controller as a target for optimization to improve convergence to a simplified equation. The complete motor model with the simplified controller is shown in Figure 6.5.

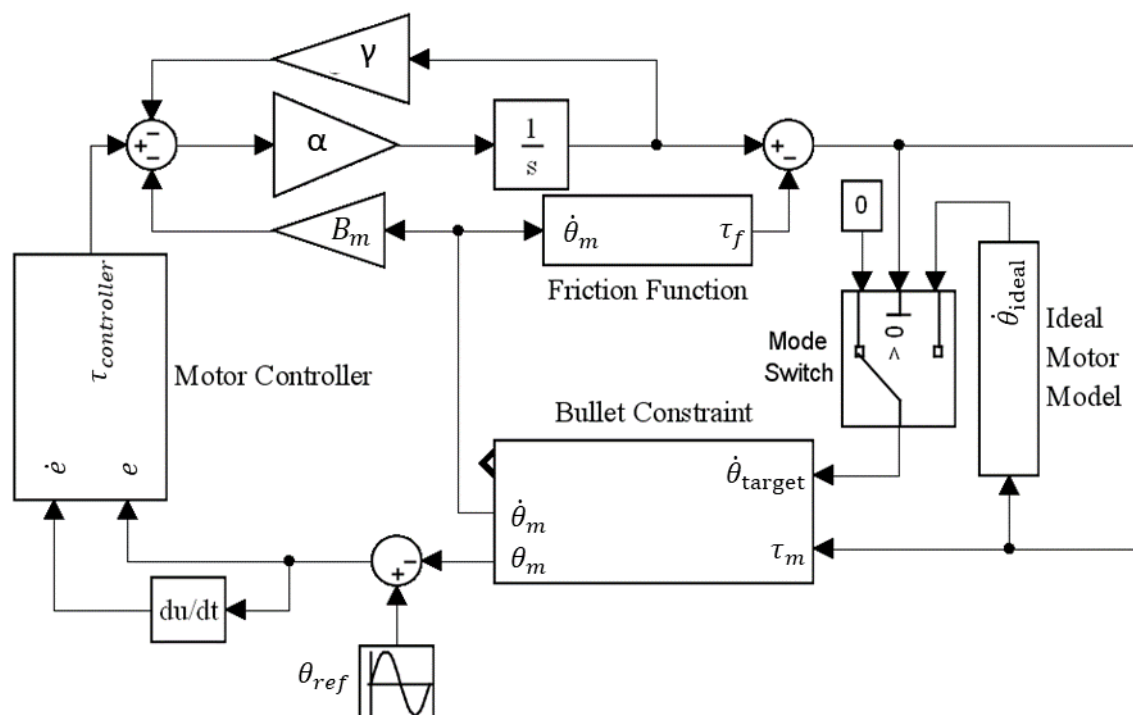


Figure 6.5: Simplified conditional constraint motor model for a servo-motor sub-simulation within a rigid body simulation package. The friction approximation and motor controller are device dependent.

6.4.4 Conditional Constraint Experimental Validation

In order to validate the proposed conditional constraint as a model of a partially black boxed servomotor, several experimental trials were conducted using the SARM system discussed in Section 3.7.1. All tests were conducted with the simulation and experiment running in open-loop, eliminating controller based corrections that may hide errors through different control responses with similar position responses.

Simulated models were implemented in a customized version of Bullet Physics Engine 2.83 with improved multithreading features and tuned base parameters. A Dantzig–Wolfe decomposition based solver was used to avoid joint drift under high loads that the default sequential impulse solver experiences even with a high number of iterations. The system was internally modeled as a hinge joint with the conditional constraint controller, with the load mass body connected to a second static body to represent the motor module and the connecting framework. Due to an inability to accurately replicate the flexible cable used to attach the load mass to the SARM system, the load was estimated as a point mass at the connection point with manually calculated inertias for the test conditions. This was expected to affect the simulation at dynamic inflexion points, where the sign of the acceleration changes during acceleration or deceleration, especially in high energy motions under significant load.

The results of the conditional constraint were accurate to the original experimental data as shown in Figure 6.6. The sine wave response in Figure 6.6 (a) is nearly indistinguishable from the experimental data for both load conditions, without sign of instability induced oscillations or unmodeled dynamics. Due to the higher uncertainties and motion energy during the step response in Figure 6.6 (b) however, errors during deceleration stages can be seen during both an up motion, and more significantly during the down motion. This result was expected due to the modeling inaccuracies, with a more rigid test platform required to improve the simulated response without increasing the simulation complexity significantly. Further test conditions and analysis are described in Section 6.5 and 6.6 below.

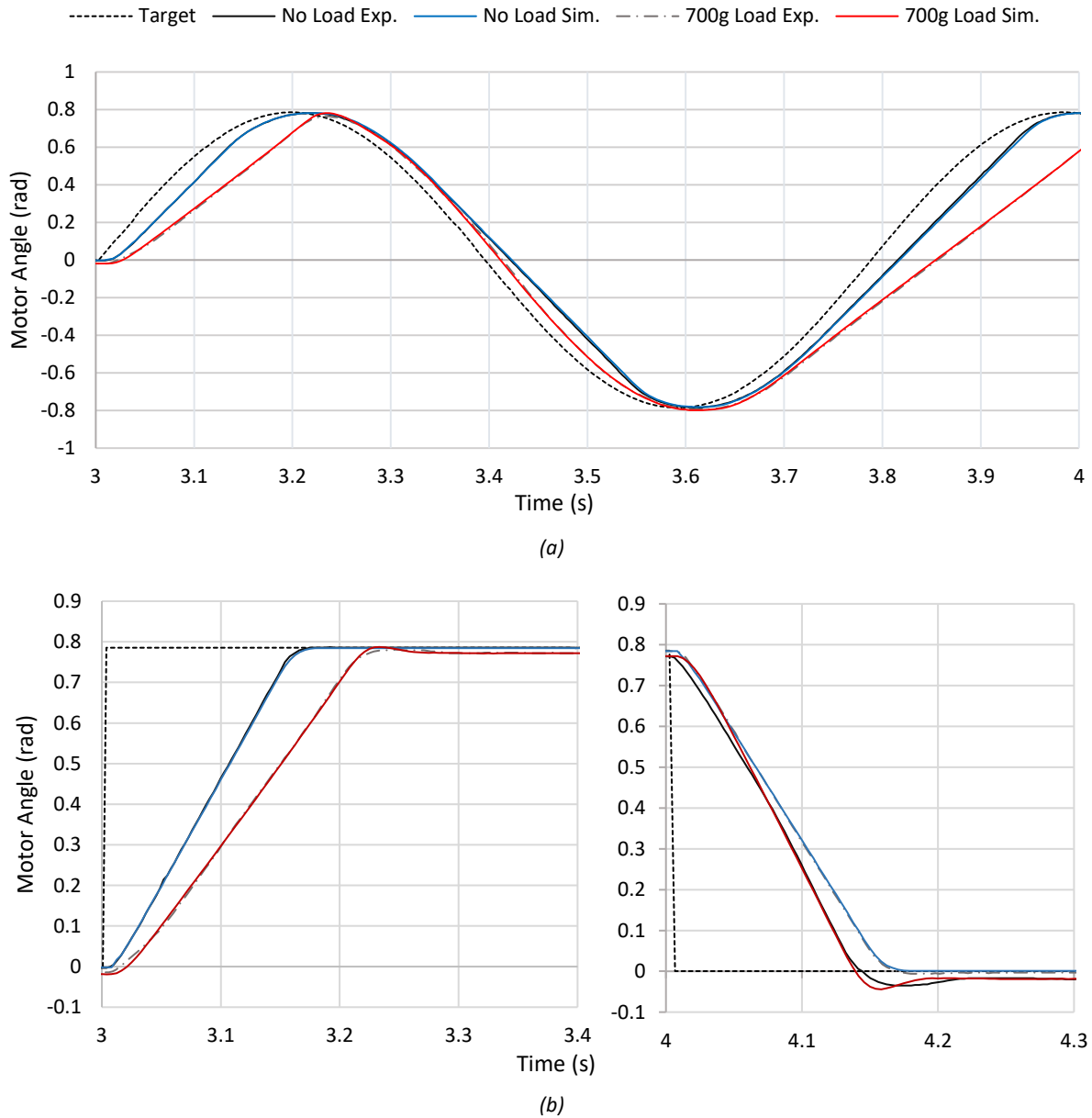


Figure 6.6: Motor response to a $\frac{4}{\pi}$ Hz sine wave (a) and step response (b) under unloaded and 700g load conditions for experimental and conditional constraint method operating at 240Hz simulation rate. Errors are orders of magnitude smaller than the input commands

6.5 Simulation accuracy

The simulation methods described in the above sections were compared using the SARM system discussed in Section 3.7.1 to validate and compare the motor simulation methods using multiple external load conditions. The test platform with an earlier version of the modular robot control and simulation framework [63] described in Section 4. The platform, seen in Figure 3.19 and Figure 3.20, was controlled and monitored by a computer via a USB to TTL serial device using a proprietary Dynamixel communications protocol and a custom coded low latency control data recording framework. The servomotor joint angles were recorded from the module internal 12bit (0.0015 radian resolution) magnetic rotary encoders, which are attached directly to the output shaft to eliminate gearing related

position error. As the communications latency can cause fitting errors twice the value of simulation errors in some of our tests, average latency of the full control and data collection cycle was recorded and applied to the simulation and kinematic models. Linear interpolation of the data was used to approximate the motor condition at the time points necessary to calculate the simulation fitness in all tests, as the data capture rate was both variable and inconsistent with the sampling rate of the simulation trials. Although more accurate interpolators were available, the difference between linear interpolation and a number of spline methods were an order of magnitude smaller than the data acquisition accuracy.

In order to ensure a large position control response error difference between loadings, two main disturbance input patterns were tested, a $\frac{4}{\pi}$ -Hz sine wave and a step function alternating between zero and the positive or negative amplitude bounds at a frequency of 1Hz. All tests were conducted with amplitude bounds of $\pm \frac{4}{\pi}$ due to hardware limitations preventing a larger amplitude. With large enough time steps, the simulations exhibit position and velocity instability at low loads, while the SARM apparatus does not experience similar control issues in experimental tests. This behavior was used to determine a lower bound for simulation frequency discussed in Section 6.6. As such, the accuracy of the electro-mechanical and conditional constraint models was tested at 2000Hz ($\Delta t = 0.5\text{ms}$) and 240Hz ($\Delta t = 4.167\text{ms}$) respectively to avoid the unstable region of control.

The no-load step response for both simulation models, shown in Figure 6.7 (a), demonstrate high accuracy throughout the motion, with steady state error less than the sensor resolution. Errors during motion are largest during the acceleration and deceleration phases where dynamic and kinematic friction are largest. Likewise, the 700g load condition in Figure 6.7 (b) shows vastly increased errors during the acceleration and deceleration, particularly on the descending phase. Although friction related items contribute to the large error, the primary issue is likely due to the mechanical difference between the actual test platform and the simulated one, namely the use of a non-rigid cable to connect the mass to the system rather than a rigid fixture. As such, the inertia and gravity force acting upon the motor is overestimated at the start of a downward motion and can lead to a deviation in the estimated velocity, and therefore position.

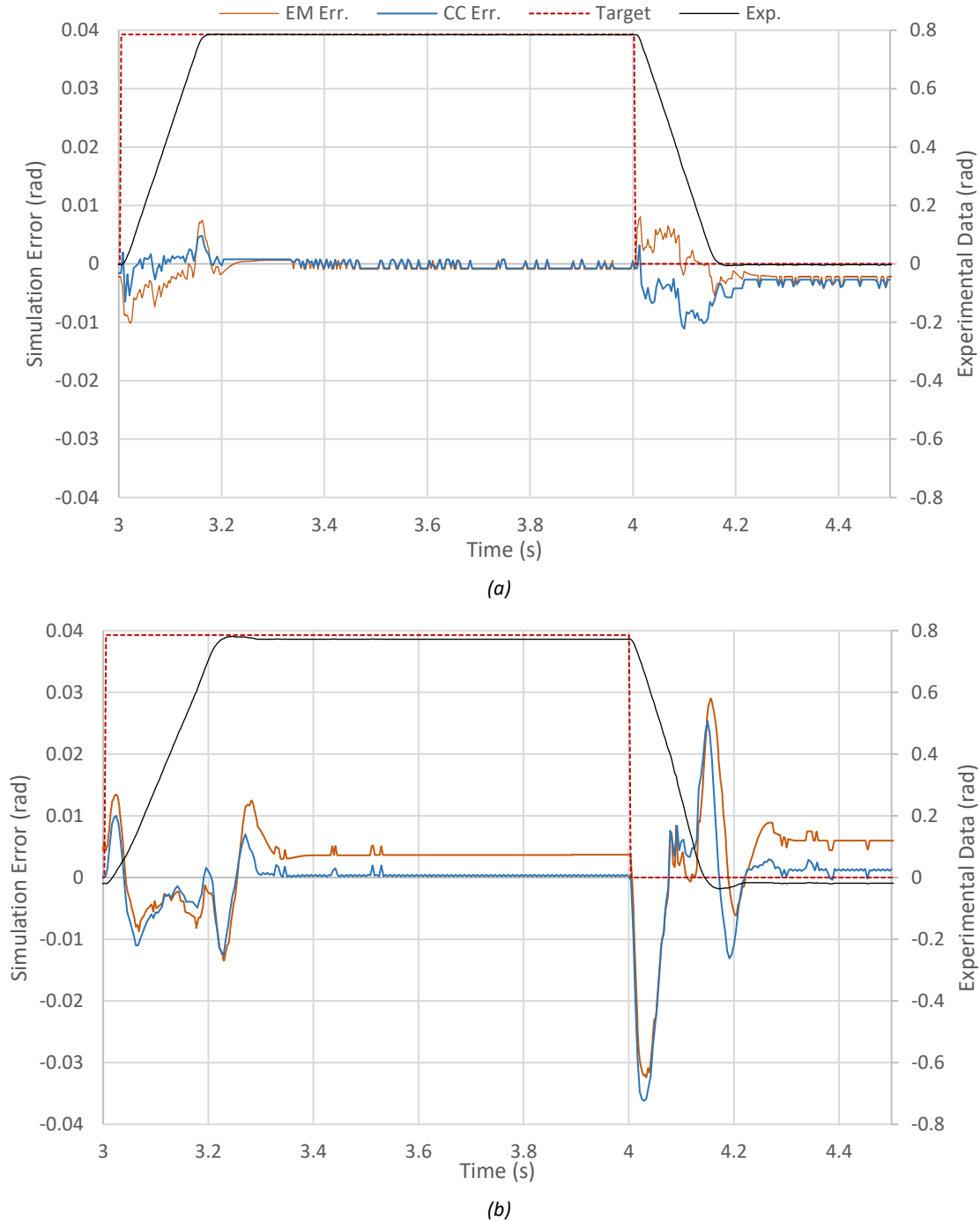


Figure 6.7: Simulation accuracy of the robot platform with unloaded (a) and 700g (b) attached load under a step function for the electro-mechanical (EM) and conditional constraint (CC) methods. The target function and experimental data are shown on the secondary axis, with a position resolution of 0.00153 radian.

The difference in loading condition response for the sine wave input case is less pronounced, as shown in Figure 6.8 (a) and (b). This is due to a lower target velocity at the beginning and end of a downward motion, during which the differences between a cable system and a rigid body one is significantly smaller. Errors during this motion are within one degree during the full motion, regardless of the load condition. The general shape of the error curves show a velocity dependent error, which may be attributed to incorrectly identified friction parameters or an unmodeled secondary motor latency.

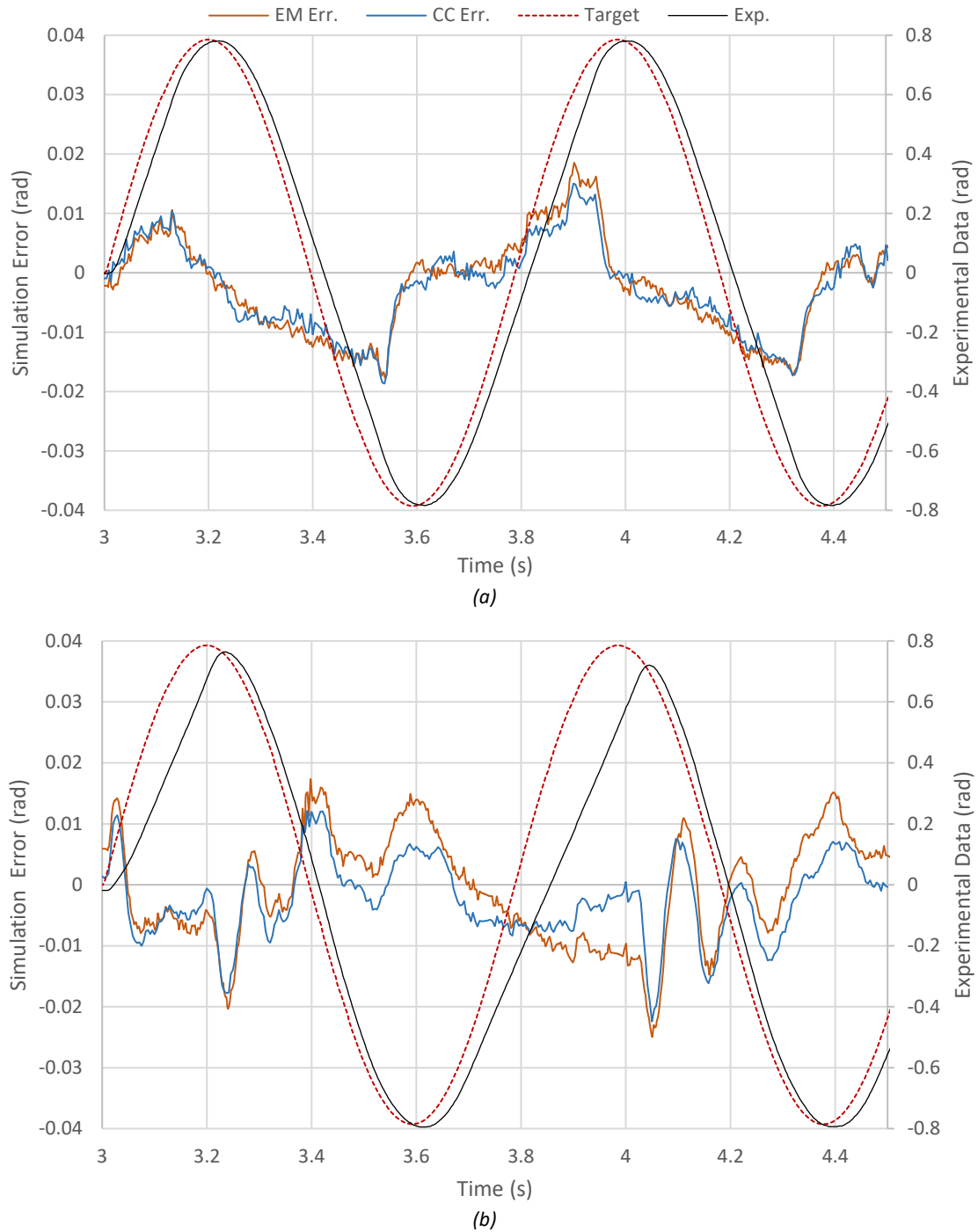


Figure 6.8: Simulation errors of the robot platform without additional load (a) and with a 700g load (b) during a $4/\pi$ Hz sine wave input for the electro-mechanical (EM) and conditional constraint (CC) methods. The raw target function and experimental data are shown on the right-hand axis.

The overall accuracy of the simulation can be determined by the simulated response error to the actual hardware response, using only an identical disturbance input and a pre-identified model set. This open-loop response measures the full scope of the simulation environment, including modeled latencies, to ensure that the simulation is a drop-in replacement for the hardware.

A common pitfall of simulation systems is the limited range of usable system conditions where simulation response is linear with the experimental results. In conditions where external loads are higher or lower than the load the system is optimized for, a reduction of simulation quality may occur due to an increase in un-modeled secondary characteristics or due to numerical issues in the simulation itself. Condition invariance in the overall error map is essential for highly dynamic robots, as the input force and target position may have higher frequency components and vary significantly over the course of one motion.

The overall error of each individual condition trial was calculated by the root-mean-square value due to the oscillatory nature of the input, using Eq. 6-11, while the maximum error was calculated as the absolute maximum error, shown in Eq. 6-12, between the experimental position data θ_{exp} and simulated position data θ_{sim} at every time m .

$$RMS\ Err. = \sum_{n=1} \sqrt{\frac{1}{m} \sum_{m=1} (\theta_{exp\ m} - \theta_{sim\ m})^2} \quad 6-11$$

$$Max\ err. = \max(|\theta_{exp\ m} - \theta_{sim\ m}|) \quad 6-12$$

Average errors in the electro-mechanical and the conditional constraint models shown in Figure 6.7 and Figure 6.8 show significantly more load independence compared to a kinematic model in Figure 6.1, however the electro-mechanical method demonstrates unimodal behavior with a minimum of 300g, the mean load during the identification tests. This unimodal behavior indicates the presence of unmodeled behavior or over-fitting of the initial test data. In contrast, the conditional constraint method shows a flatter response across the tested loading range. Additional tests using additional, non-pivot rigid body solvers showed no significant change in accuracy profile for either the electro-mechanical or conditional constraint methods.

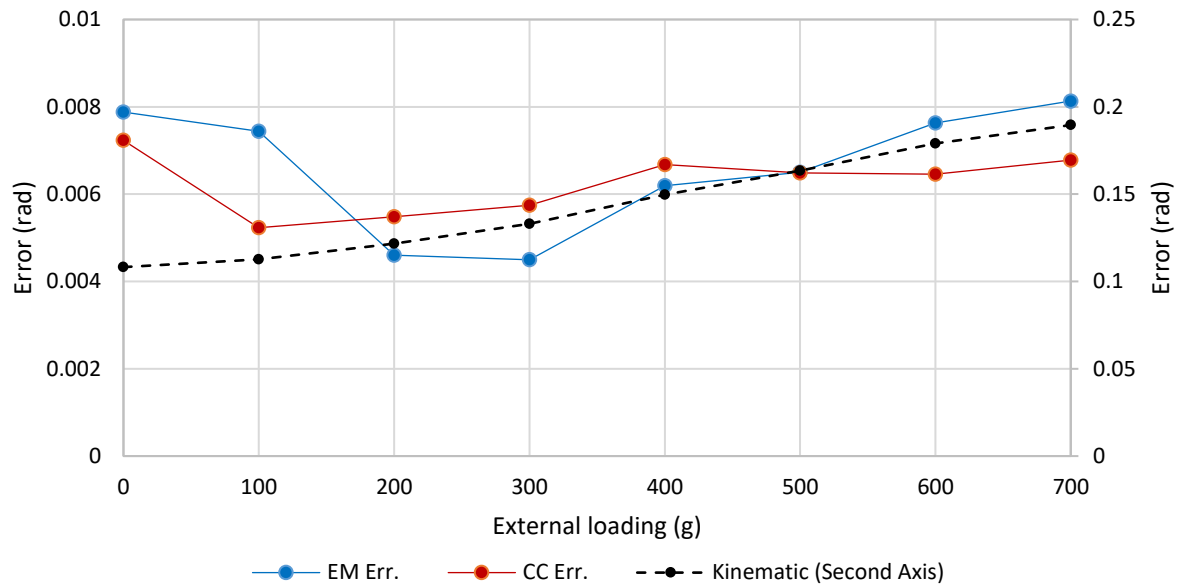


Figure 6.9: Quadratic mean error of the electro-mechanical and conditional constraint models during a $\frac{4}{\pi}$ Hz sine wave target angle input. As the error is over a magnitude of order larger, the result of the latency corrected kinematic model is provided on the second axis.

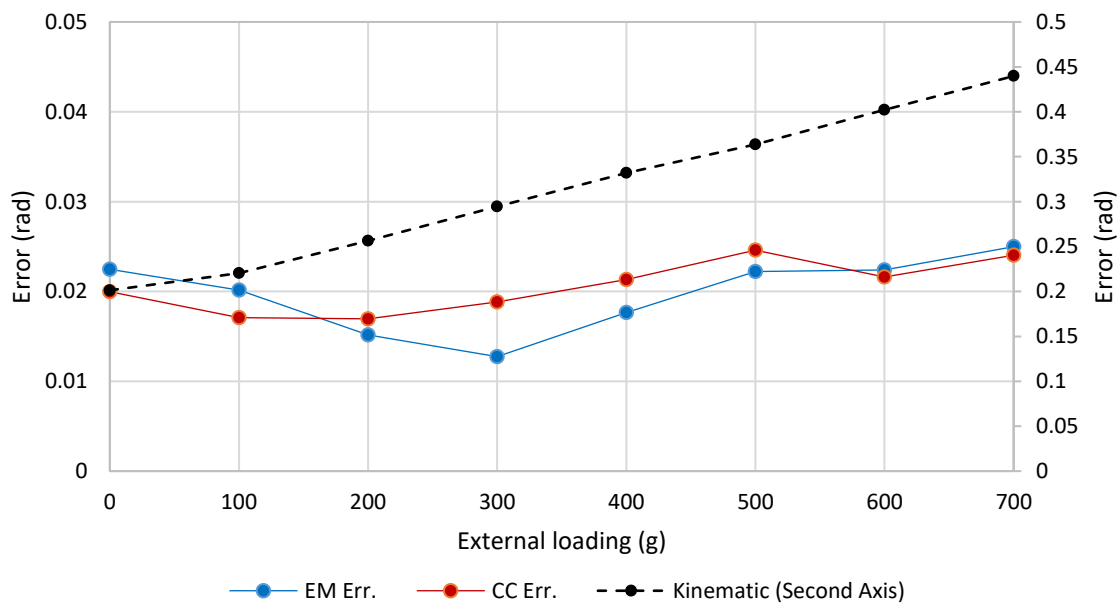


Figure 6.10: Maximum error of the electro-mechanical and conditional constraint models during a $\frac{4}{\pi}$ Hz sine wave target angle input. As the error is over a magnitude of order larger, the result of the latency corrected kinematic model is provided on the second axis.

6.6 Simulation performance

The performance of a simulation is very important in robotics due to real-time constraints that are often associated with their control and use. In general, performance can be separated into two concepts, computational performance and physical accuracy, which often are inversely related. Not all mechanical systems require complex models to be physically accurate, systems which have low friction and high stiffness require only minimal models to produce low errors against empirical data in systems

with small and slow changes in state. Meanwhile robot types with low stiffness, such as 3D printed robots with cheap servomotors, may require advanced models taking friction, strain, and hysteresis in order to be reasonably accurate in highly dynamic motions. Likewise, the simulation methodology can be computationally faster or slower due to a variety of factors including integration type, step size, and underlying optimizations in both the algorithms and implementation.

Overall simulation system performance is a function of the maximum computational efficiency for a given set of accuracy and stability constraints for the simulation. The empirical simulation properties for the $\frac{4}{\pi}$ Hz sine wave experiments are outlined in Table 6.1.

Table 6.1: The framework simulation accuracy and performance for a $\frac{4}{\pi}$ Hz sine wave input to the SARM platform. Performance calculations are for single threaded operations on an Intel Core i5 3570 processor on Windows 10 operating system.

	Kinematic *	Electro-mechanical	Conditional Constraint
Avg. Error (all load conditions)	1.45×10^{-1} rad	6.60×10^{-3} rad	6.30×10^{-3} rad
Max. Error (all load conditions)	4.40×10^{-1} rad	2.50×10^{-2} rad	2.46×10^{-2} rad
Sim. Step computation time	<0.01 ms	0.108 ms	0.091 ms
One Sec. Sim. Computation time	<1ms	215.35 ms	21.89 ms
Minimum stable simulation frequency	N/A	2000 Hz	240 Hz
* Only the unloaded condition used for comparison			

As a kinematic model provides a low computational cost pose estimation for comparison purposes, the two dynamic methods described in Sections 6.3 and 6.4 were compared to a purely kinematic approximation that included feed forward data latency compensation that can be easily implemented in the framework described in Section 4.3.1. The dynamic simulation methods are multiple orders of magnitude more accurate, showing RMS shaft position errors in the range of 6 milliradians and maximum errors in the range of 25 milliradian. In terms of the motor resolution, that is the smallest angle difference that can be recorded, this is approximately equal to two lowest bits of resolution. Furthermore, as the data noise floor at steady state holding is $\sigma_s = 7.82 \times 10^{-4}$ rad, rendering a 3-sigma value of 2.35×10^{-3} rad, for a total $X+3\sigma$ of greater than half of the average error. For the quadruped robot discussed in Section 0, this average shaft error equates to an average endpoint simulation position error of approximately 1mm and 4mm maximum position error.

The performance of the simulation, for a given simulation system, dependent on the stability of the simulation, assuming the system state changes between steps are small enough to be estimated with linearized methods. Table 6.2 shows the simulation accuracy of a $\frac{4}{\pi}$ Hz sine function input for an unloaded platform simulated at various step sizes.

Table 6.2: The quadratic mean error of each simulation method for a $\frac{4}{\pi}$ Hz sine function input in the no-load condition. Unstable results that do not cause simulation failure result in excessively high average error.

Frequency (Hz)	Electro-mechanical	Electro-mechanical with Separate Friction	Conditional Constraint
2000 Hz	5.57×10^{-3} rad	2.44×10^{-3} rad	5.75×10^{-3} rad
1000 Hz	3.60×10^{-1} rad *	4.55×10^{-2} rad	5.80×10^{-3} rad
500 Hz	3.53×10^{-1} rad *	5.48×10^{-2} rad	4.22×10^{-3} rad
240 Hz	3.54×10^{-1} rad *	5.92×10^{-2} rad	4.73×10^{-3} rad
120 Hz	3.54×10^{-1} rad *	5.35×10^{-2} rad	1.07×10^{-2} rad

* Simulation failure, no motion possible

For this particular test, errors less than 10 milliradians indicate high accuracy, while values above 50 milliradians generally indicate strong instabilities. In the case position values oscillate between points without following input trajectory, the simulation is considered to be in a failed state. The separate friction method for the electro-mechanical model shows significantly improved stability at and below 1000Hz, however the errors at these rates is an order of magnitude larger than for the 2000Hz case. Only the conditional constraint maintains high simulation accuracy, and therefore stability, with lower simulation frequencies. Functionally equivalent results at all simulation rates at and above 240Hz are possible through this method, and although the average error is doubled at 120Hz, it is significantly lower than the other simulation methods.

The computational performance of the motor models discussed above was found to be mostly proportional to the step size and number of constraints necessary to implement the model. The electro-mechanical model and conditional constraint model use an equal number of constraints internally and therefore have approximately equal solver performance at any given simulation frequency. The electro-mechanical with separate friction constraints requires significantly more computation time for a given step size due to additional constraints on the system; in the current implementation this method requires twice the number of constraints, however it is possible to reformulate the constraints equations to require a single additional constraint. All three methods ran with computation times that could be considered faster than real-time for this test platform, however the direct electro-mechanical model with additional friction constraint has difficulty running more complex models at real-time in non-trivial environments due to inefficiencies in the current implementation. As the separate friction method does not provide significant improvement over the conditional constraint and standard electromechanical model, the method was not considered despite increase in overall simulation stability.

6.7 Summary

A conditionally active constraint acting as both a motor and friction actuator, developed for use on existing rigid body solver packages without requiring modification of the underlying constraint solving methods, shows promise in simulating low cost black boxed servomotors within discrete time rigid body physics engines. This proposed conditional constraint method produces stable simulation results

with low error throughout a wide range of loading conditions. The conditional constraint model starts experiencing simulation instabilities with frequencies lower than 240Hz, eight times lower than a traditional electromechanical model with friction. A modification to the motor controller estimation may be able to further improve stability at low simulation frequencies, however the increase in simulation errors due to other factors may limit accuracy improvements.

By utilizing larger time-steps without increasing the simulation error or lowering per-step computational performance, significant performance increases can be achieved. This faster simulation opens up several possibilities for a full dynamic representation of the robot where prior methods would have been computationally too intensive, including full body inverse kinematics with dynamics compensation. Additional optimizations in the constraint switching conditions may improve the simulation frequency stability range for this method, further increasing performance.

As part of this new model, approximations to the torque produced by the motor can be used in place of the full electro-mechanical torque model without a loss of accuracy, facilitating identification due to the removal of variable redundancy. This new method can be used in the identification of the motor properties required for simulation using only minimal observational data during regular use, with identification and simulation results within an order of magnitude of the motor sensor resolution. This accuracy and speed in simulation can additionally be used to implement state estimators for various highly abstracted states with minimal input data, the topic of 7.4.

7 Model Identification and Soft Sensing

7.1 Introduction

Two concepts rooted in model identification of robotic components are useful for the development of low cost platforms which the prior chapters describe. Directly, the identification of motor and control parameters is an important step in the calibration of a motor model for use in control and simulation. Although parameters for a variety of components can be assumed through reverse engineering the devices and searching manufacturer specifications, many parameters may be undisclosed due to trade secrets or simply unknown to the manufacturer. Even in the case parameters are available, variances due to manufacturing tolerances may be significant for the model. Figure 7.1 shows the significant improvement in simulation quality when a model is properly calibrated. The uncalibrated system showed was developed using motor controller and gearing parameters from the Dynamixel MX 28T specifications [55] and motor electrical properties from Maxon motors. However, when the parameters are applied directly to the electromechanical model in Section 6.3, the response bandwidth is significantly lower than expected. Calibration of the model properties, especially controller bandwidth and friction components, can reduce errors by an order of magnitude compared to an uncalibrated model.

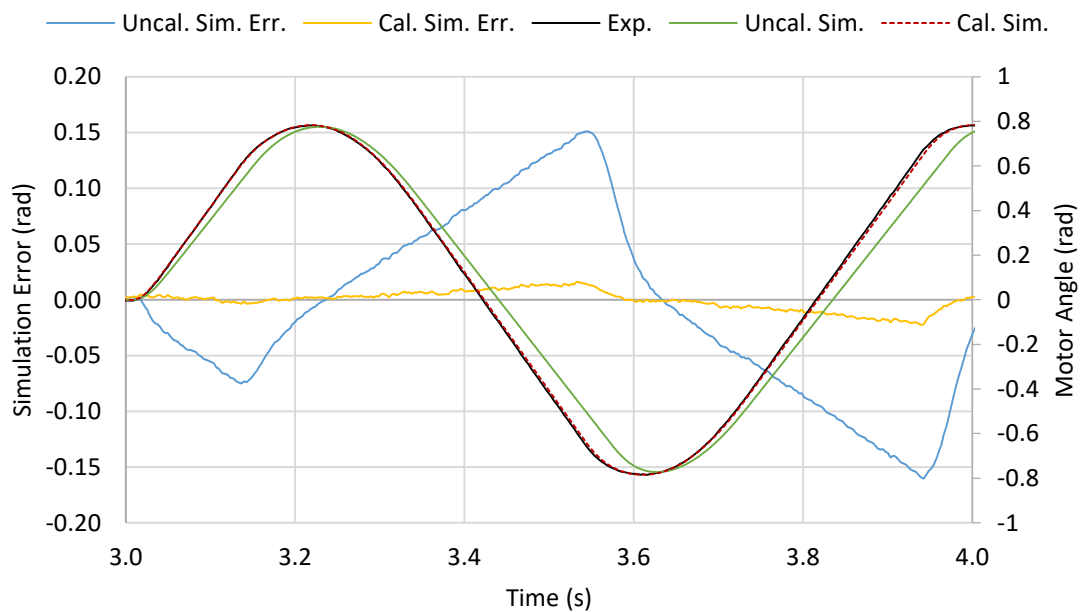


Figure 7.1: A sine wave input disturbance simulation for an uncalibrated and calibrated model of the SARM platform. Uncalibrated errors are only marginally better than a pure kinematic response.

In addition to calibration, model identification can be used as a measure of model accuracy to a changing robot plant response, including signs of health issues. The most important use however is in the development of sensor estimating functions. These “soft-sensors” can be used to indirectly estimate system properties by a variety of methods that use both physical hardware sensors and their software counterparts in tandem. The state estimators can be corrected with sensory information when available, forming an observer-corrector pair. Observer based soft-sensing has long been used in low-dimension, well understood systems to estimate directly and indirectly observable state variables. Several estimation methods can be used within these virtual sensors.

The traditional approach to soft-sensing is the development of an observer to a known system, most often in the form of a state observer or a linear or non-linear filter such as the Extended Kalman Filter. For well formulated, linear problems, these methods can effectively reduce steady state errors between their approximation and the actual values to zero in finite time. State observers use a full or reduced order model of a known plant to formulate an estimated system state that can be used in control feedback loops. The error between the actual response and the estimated response is used to converge the estimated system to the real system by means of a fixed observer gain. In a control system with an observer, all observed feedback variables are estimated, with the raw data only used for convergence of the estimated values. For a state observer to be developed for a particular system the system must be observable, which implies a knowledge of the various models comprising a system must be known in full.

An alternative formulation of the state observer is a Kalman filter, which acts as a self-correcting probabilistic state observer. The extended Kalman filter in particular is often used in robots due to its ability to observe some non-linear systems where state observers may fail. In general, a Kalman filter works by alternating between states of propagation, where assumed system dynamics are used to provide an estimated system state, and innovation, where new data is integrated into the system state via a Kalman gain on the estimate to better track the actual state. As the propagation state does not provide new information to the system, the state uncertainty is continually increased. In contrast, the innovation state reduces uncertainty by decreasing the covariance of observations through the addition of new observations. By utilizing the likelihood of values for a particular target variable the Kalman filter reduces the sensitivity to noise and negligible errors in the control model used while providing additional information on the covariance of data for different system components.

The error based soft sensing implementation can be done in real time using the error between the main simulation and the hardware results to estimate properties that cause these errors, however no such methods have been implemented in the current version of the framework. Instead, soft sensing through more generic, optimization based methods, was implemented due to the similarity to motor identification methods described in Section 7.2. Using the control framework discussed in Chapter 4 using the simulation methods described in Chapters 5 and 6 applied to a representation of the robot platforms in Chapter 2, a soft-sensor can be implemented that allows for estimation of the robot state using minimal hardware sensing.

7.2 Motor parameter identification by particle swarm optimization

System identification is a complex field of research with a wide assortment of specialties including speed optimized, robust, and generalized identification techniques. A typical solution to identification is the reformulation of equations to produce a base parameter set, the minimum number of dynamic parameters necessary for a model, and subsequent excitation of each base parameter [64]. One common issue with this approach is sensitivity to input excitation, where redundant or low relevance parameters during a specified excitation may not converge to meaningful values or be overfitted to a specified input excitation [65]. As such, a more robust and generalized identification methodology was required for the purposes of the simulation framework.

A heuristic search using particle swarm optimization (PSO) was chosen for simplicity in scaling with additional parameters with low additional sensitivity to input excitations used. In PSO, simulation parameters are mapped to an n -dimensional space that contains all possible particle positions, with one dimension per parameter being optimized simultaneously with all other parameters. Each particle updated to a new value through a function of the distance between its current position and a set of attractors. This attractor set is comprised of the position particle's best fitness for any prior position, the best fitness entire collection of particles at the current time, the best fitness entire collection of particles at any time, or additional attractors. Although convergence to a global minimum cannot be guaranteed, this method can provide reasonable solutions even with highly non-linear or disjointed solution spaces. Although there are many flavors of PSO available, two types were implemented and used for optimization in the software framework, namely the weighted particle swarm optimization (PSO) and a derivative of the competitive multi-swarm cooperative particle swarm optimization (COM-MCPSO) [66] methods. As the weighted PSO can be considered a special case of the COM-MCPSO, the method described in Table 7.1 and Algorithm 4.1 is applicable to the weighted PSO case as well. As this method is a component of the optimization system described in Section 4.7.3, Algorithm 4.1 describes the unique update step of the method.

Table 7.1: Parameter definitions for the Multi-Swarm PSO method.

Parameter	Primary function
ρ	Random variable unique in each instance
C_n	Weight constant for attractor n
B_n	Best fitness parameter set for attractor n
v	Time step normalized particle velocity
w	Inertia parameter of particle
n	Attractor list, includes group bests

Algorithm 7.1: Multi-Swarm PSO Update Loop

```

1: Input: Fitness Result and position  $x$  for all particles
2: Persistent: Global, Local fitness position and value
3:  $n = [\text{Global} \quad \text{Local} \quad \text{CurrentGlobal} \quad \text{CurrentGroup}]$ 
4: for each particle
5:   for each group  $n$ 
6:      $B_n = \min(B_n, x)$ 
7:   end
8:    $v \leftarrow wv + \sum C_n \rho(B_n - x)$ 
9:    $x \leftarrow x + v$ 
10: end
11: return  $x$  positions

```

Several starting and boundary conditions for PSO particles, based on the work of Niu et al. [66], were tested independently to verify solution performance and consistency. The unknown properties were bounded to reasonable approximations to improve convergence speed. PSO of 256 particles, with random initial position and velocity within the bounds, was run for a maximum of 10000 PSO steps. In order to improve system performance during optimization, each test was run independently in a worker thread, with as many active worker threads as CPU cores. Results were then verified with additional test data and inputs not used in the optimization step. The additional error due to the inertia approximation during PSO optimization did not result in solutions that were not physically viable for the motors used in the experiment.

7.2.1 Application to SARM platform

To validate the quality of identification results using this optimization technique, the motor parameters of the SARM platform with MX-28 motor was tested using the conditional-constraint. As no redundant variables exist in this representation, standard optimization techniques can be used to calculate reasonable values for all system variables in the case of the conditional constraint method, which can be found in Table 7.2. For all methods however, the friction and controller parameters were unknown but internally independent and were set as additional target variables for optimization.

Table 7.2: Optimization parameters for the simplified torque equation conditional constraint model

Parameter	Primary function	Search Bounds
α	Torque rate	0.0, 10.0
γ	Torque feedback	0.0, 10.0
B_m	Torque-velocity curve	0.0, 10.0
J_{int}	Internal inertia	0.0 kg m ² , 1.0 kg m ²
k_p, k_d, k_i *	PID parameters	0.0, 120.0
f_s, f_d *	Friction parameters	0.0 Nm, 2.5 Nm
τ_{max}	Maximum static torque	0.5 Nm, 5.0 Nm

* parameters also independently optimized for the electro-mechanical model

The friction properties, controller gain, and torque scaling were set as unknowns for optimization and were used to minimize the fitness error of the simulation for a set of independent trials, given by

$$fitness\ error = \sum_{n=1} \sqrt{\frac{1}{m} \sum_{m=1} (\theta_{exp} - \theta_{sim})^2} \quad (12)$$

for n trials and m simulation steps. Sixteen experimental trials, two input functions with eight loadings each, were grouped as sub-samples of a single sample and the total error across the trials was used as the fitness variable for comparison between individual particles.

For model simplicity and performance purposes, the load was modeled as a small mass attached at the endpoint of the arm rather than the cable mechanism used to provide a localized force. This approximation has very low error when dynamics are limited by motor electro-mechanical properties, however the step response may show increased error at the start and stop of motion due to secondary system dynamics from the mass system used. The friction components and controller gains were set as optimization variables, and the induction parameter was also allowed to be modified within a small range of its expected value to offset integration errors in the electro-mechanical model. Experimental data from the $\frac{4}{\pi}$ Hz sine wave and 1Hz step function command inputs using external weights at 100g loading intervals between no load and 700g load at a distance of 56mm from the rotation axis were compared against the simulated systems to ensure a fit across a broad range of conditions.

The results of one such trial can be seen in Figure 7.2, where the model errors are successfully reduced to a sixteen-condition sum of RMS values of under 0.07 radian, an average error of just 0.25 degrees. An additional data set using a slower $\frac{1}{2\pi}$ Hz sine wave with amplitude bounds of $\pm \frac{4}{\pi}$ was used as a control test, without being used within the optimization, to verify model accuracy and ensure models were portable after optimization steps. Additional results of the identification for both this model and the fine-tuning of the electro-mechanical model were presented in Section 6.5.

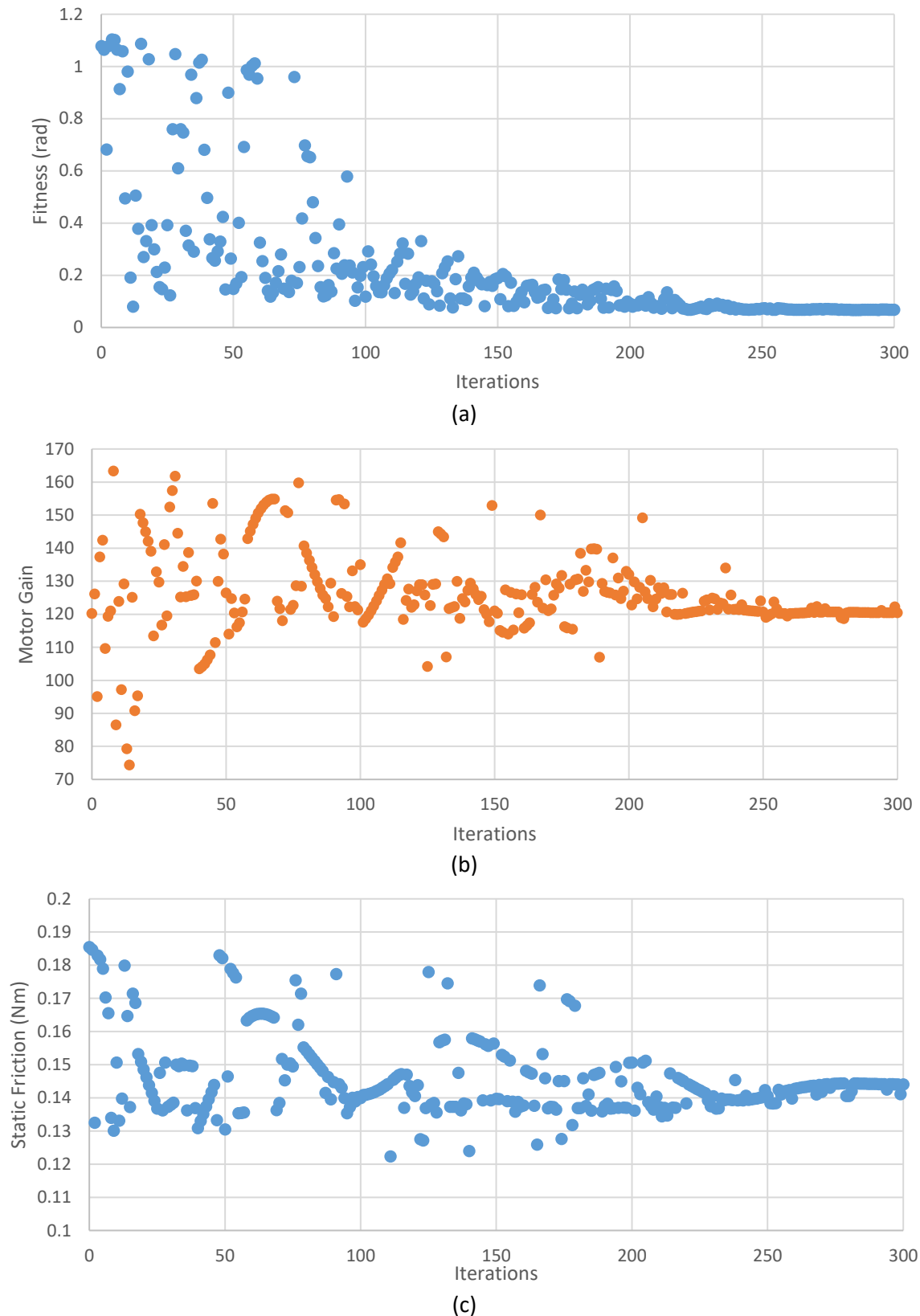


Figure 7.2: The results of a single optimization run for identification of nine variables using MCP SO, using values near the known solution to improve convergence time. The best fitness (a) during each iteration varies significantly until particles settle to a minimum solution. The outline of the trajectory of one particle can be seen in the gain (b) and static friction (c) as a sinusoid with decreasing amplitude.

7.3 Real-time Soft Sensing

The control framework is capable of implementing a wide variety of soft-sensors in real-time using any available data including the joint positions and inertial measurement units of the platform. Of particular interest is the application of torque sensorless estimation to the detection of payloads and unmodeled external forces. To demonstrate the effectiveness of this system in real-world use, the SARM system was used in conjunction with a calibrated motor model outlined in Section 6.4. Direct load sensing by using the position difference between a no-load condition of SARM and a loaded condition is possible using the steady state condition that can be seen in Figure 7.3, particularly in the steady state response at the end of a motion.

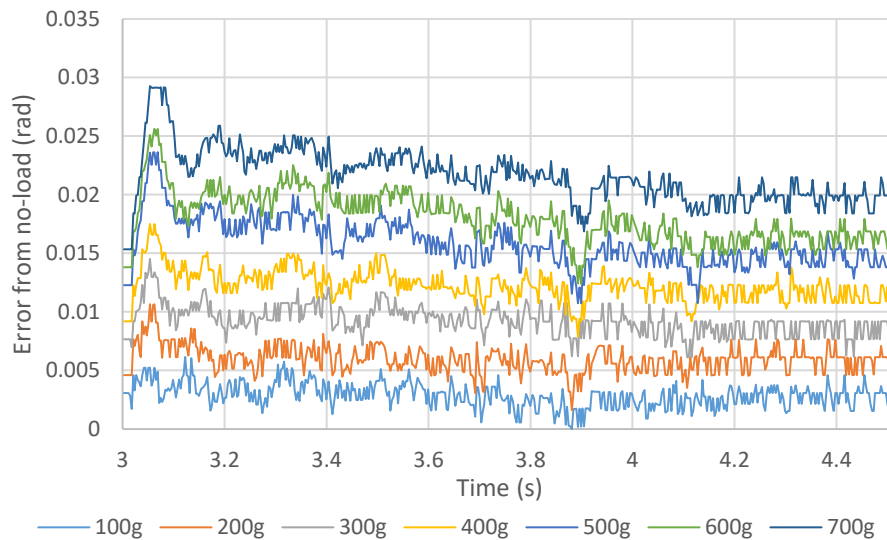


Figure 7.3: The SARM responses as a function of the no-load condition response to a $\frac{\pi}{4}$ amplitude step function shows strong correlation to load conditions.

These distinct changes in motor response to load conditions can be used to develop algorithms which can detect these changes given a certain response value from the motors and a model of the no-load condition for that system. Any additional position error can be considered as an unexpected load at the end effector of the SARM, and when a further assumption that this load is passive is made, a point mass pendulum model can be applied to an observer to compute the equivalent mass. The noise in the error value can be reduced through tuning of the observer bandwidth, and the observer result can be further averaged to obtain additional resolution. While an ideal case such as the step function on a single motor can produce relatively good quality results considering the low signal to noise ratio of the original data, less trivial systems and inputs reduce the effectiveness of simplified estimation systems.

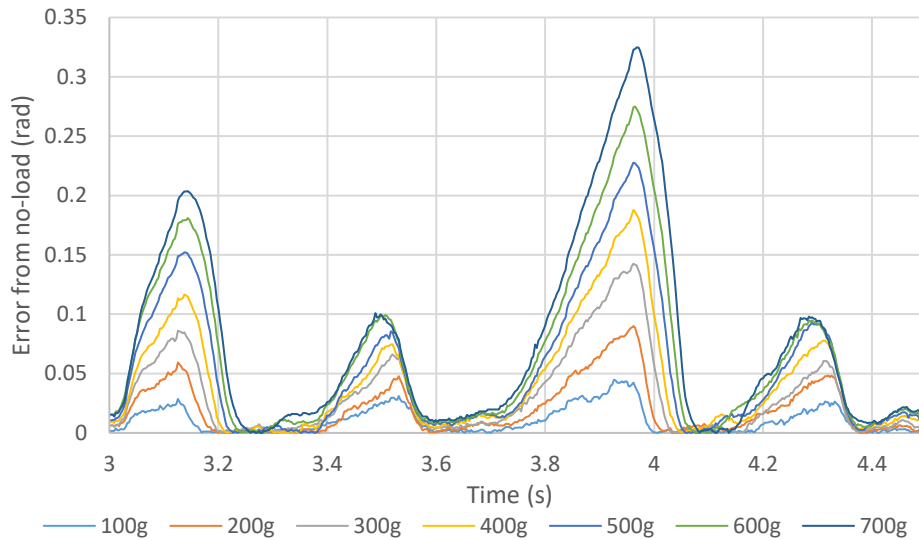


Figure 7.4: The SARM responses as a function of the no-load condition response to a $\frac{4}{\pi}$ Hz sine wave shows areas of strong correlation and weak correlation to load conditions.

In the case of a highly non-linear system such as the mobile articulated robot discussed in Section 2 and as seen in Figure 7.4 above, the basic requirements necessary for the proper function of traditional state observer and Kalman filter soft sensors may not be able to be met. In particular, the following assumptions for each are invalidated:

State Observer

- System is non-linear in many subsystems, not all subsystems are observable
- Linearized approximations may be unstable

Kalman Filter

- Information update rates are relatively low compared to control rate
- Time uncertainty in joint data is very high [67]

7.4 Optimization based soft-sensing

In order to avoid issues present in direct state observer and Kalman filter designs, optimization based soft-sensing uses the metaheuristic optimization methods described in Section 4.7.3 above to find a virtual system state that matches known sensor data for a given set of commands. A full system representation, including hardware latency compensation and controller modeling using the framework outlined in Section 4 as well as the motor models discussed in Section 6.4, can be subtracted from hardware responses throughout a motion to produce a fitness function in terms of overall or average errors. Each successive simulation trial reverts the simulated system back to a known state and computes an open-loop estimation of successive states independently of the robot, paralleling the system identification process for the motors themselves. Given a reasonably accurate calibration of the simulated system, arbitrary system states can be estimated even when they are not directly measurable.

For a single variable which has high physical correlation to a data trend observed in available trace variables, the resultant fitness function is often near-unimodal when only expected external forces are present. The verification of the unimodal shape is shown in Figure 7.5, with average simulation errors reaching minimum model errors of approximately 2 millirad within 10% of the actual applied load. This takes advantage of the high model accuracy and highly unimodal nature of the expected fitness function to quickly find a system minimum.

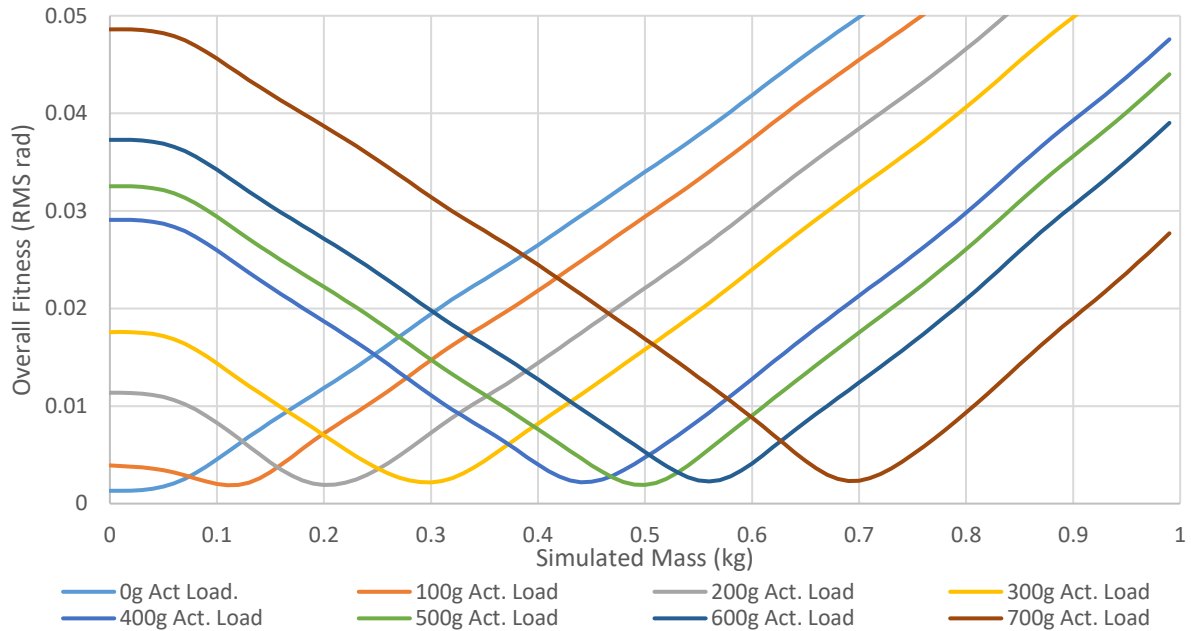


Figure 7.5: The fitness function results for multiple actual and simulation loading combinations is highly unimodal, with function minimums approaching the actual load in a high degree of loadings.

The input trajectory can be any known function from a simple sinusoidal target angle applied on a single motor, to full body gait control with additional deterministic per-motor cascading controllers. This uses the high simulation performance in conjunction with the multi-threaded capabilities of the optimization techniques to provide thorough investigation of a large search space for the best fit to existing data. In practice, single variable estimation can be conducted in real-time at rates above 10Hz using the structure outlined in Figure 7.6.

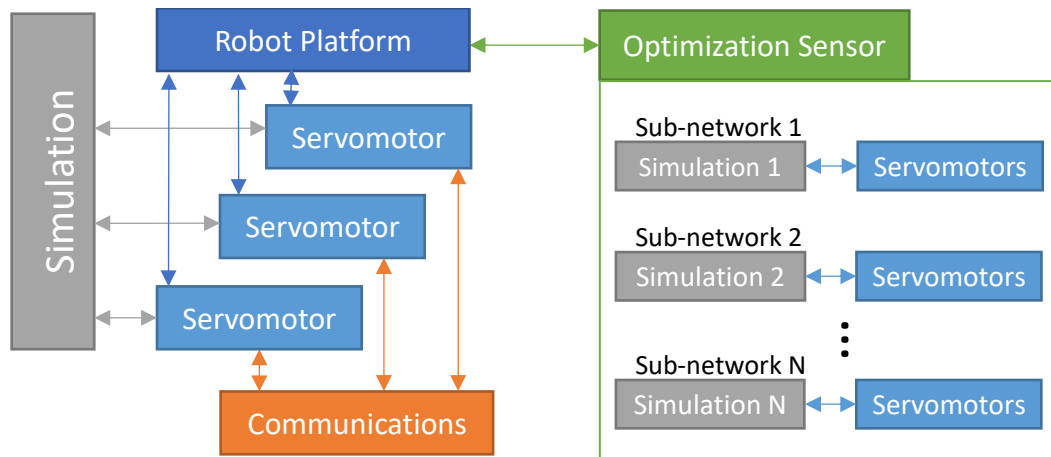


Figure 7.6: The connection graph for an optimization based soft sensor. The "sensor" employs sub-networks based on the actual control network. Sub-networks are independent of the full network and can be run in parallel.

Optimization over the possible loading masses was conducted for several input functions, loadings, and sampling times to verify the effectiveness of this proposed methodology. Module identification parameters previously calculated using the identification methods discussed in Section 7.2 above were used for all experiments. In general, the resulting fitness functions for each experimental load was generally unimodal with sampling windows above 250ms. The performance of smaller window sizes depends greatly on the dynamics of a given joint target function, and may decrease in accuracy for certain dynamics including the down-stroke of a sine wave shown in Figure 7.7 (a) in the 4.00 to 4.25 window and Figure 7.7 (b) in the 3.25 to 3.5 window. Close analysis of these points, shown in Figure 7.4, reveals loading singularities, where the payload is essentially decoupled from the test platform, however even in this case the methodology is able to produce relatively accurate results from the minimal non-singular data within the sampling window.

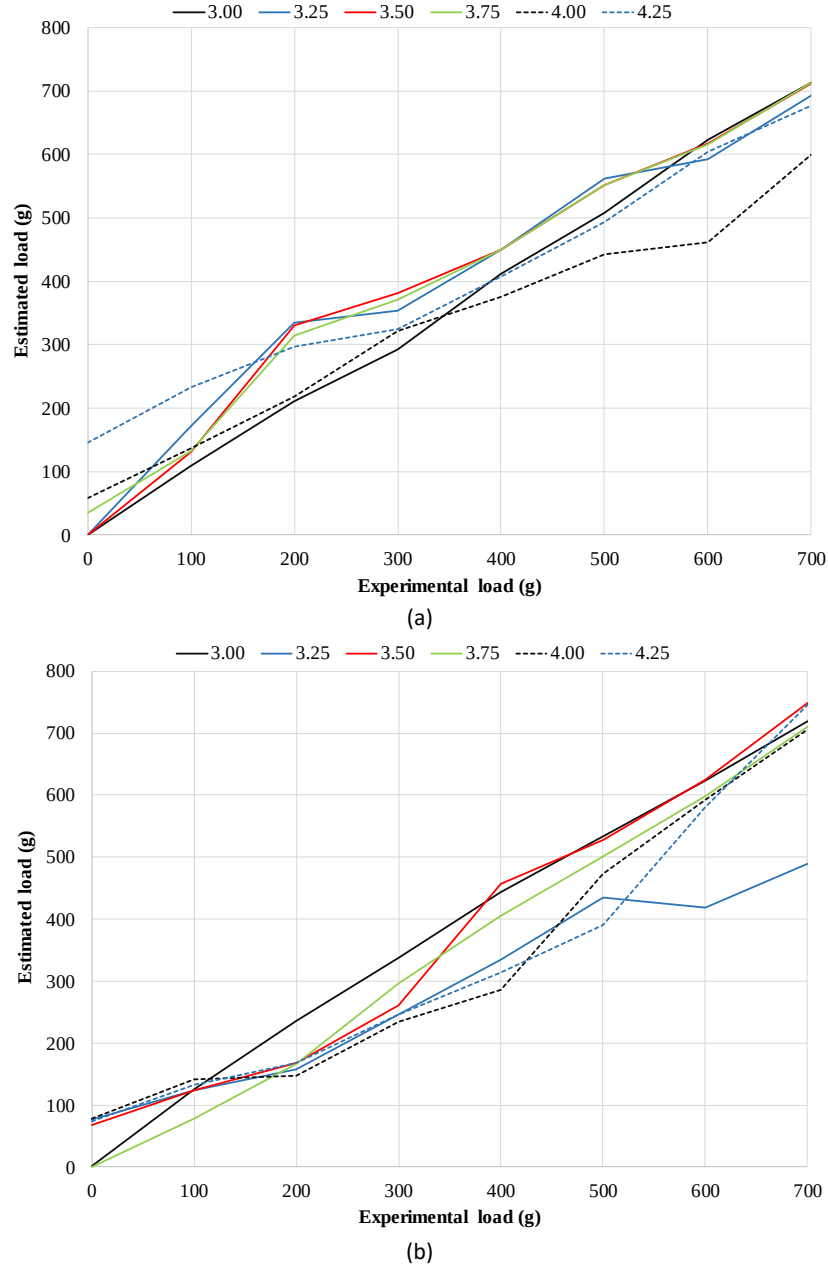


Figure 7.7: Soft sensing results for a step function (a) and $4/\pi$ Hz sine function (b) for each 250ms long sensing window starting at 3.0s, 3.25s, 3.5s, 3.75s, 4.0s, and 4.25s. The largest errors coincide with dynamic response singularities shown in Figure 7.4.

The results of a one second window, shown in Figure 7.8 (a) are accurate to within 10% even with relatively light loads that could otherwise be masked by noise and unmodeled dynamics. When compared to a traditional approach utilizing a torque observer and the low-quality torque sensing available in the motor modules, this method is significantly more stable and can achieve higher accuracy for the same experimental conditions.

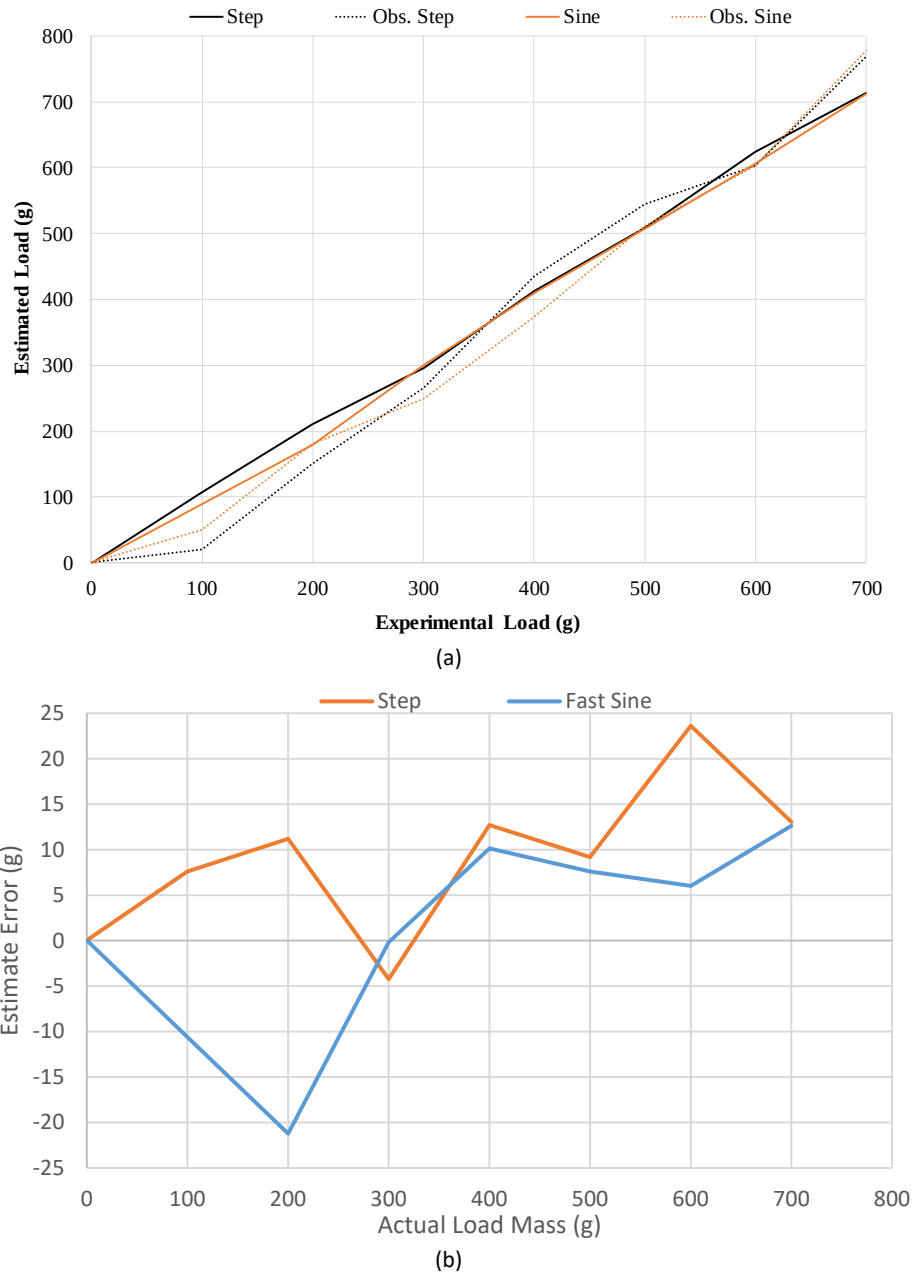


Figure 7.8: The 1 second long sensing window results for both step and 8Hz sine functions (a), using the framework based state estimator method and a traditional torque observer (Obs.). The errors from the tested conditions (b) are within 20g of the measured loading for all trials, estimating the actual mass to within 10%.

Given a sufficiently fast optimization routine, a good approximation of the payload mass can be calculated and applied to the main system simulation to improve overall simulation and control quality. Using a golden section search over a uniformly sampled spline as the unimodal optimization method, the current system can reliably achieve real-time performance with a single test parameter at a rate of approximately 12Hz when using independent one second sample windows on an Intel Core i5 3570. As payload changes are rare, this sampling frequency is sufficient for a majority of control schemes when the results are fed back into the main simulation, particularly if existing simulation data is cached between sensor updates.

7.5 Virtual Sensors

As opposed to the soft sensors discussed in Section 7.2, virtual sensors are purely simulation based and do not directly employ hardware data for estimation. Although they can be encompassed in the observer theories of Section 6.3 as internal states in the case of state observers or projections with high uncertainty in the case of Kalman filters, the virtual sensors discussed here have significant distance from the inputs and outputs that make the categorization of the variables difficult. In many cases the variables involved are likely unobservable through the traditional methods, having states indistinguishable from other related states due to internal aliasing, but nevertheless stable and therefore detectable. The level of trust in the validity of the virtual sensor data can be estimated with statistical methods, however this is outside the scope of this research due to the complexity of the system. Given an accurate enough simulation of the environment and robot components, it is possible to calculate information about a robot system that cannot be measured directly on the physical system. Reaction torques and object distances can be measured through the simulated environment for use in robot pose control and foot path planning algorithms. Although SLAM information is necessary in the case of object distance measurement, the simulated environment provides an opportunity to test possible planned results ahead of motions to detect potential problems.

7.6 Summary

The robust generalized model identification system built into the framework was validated through accurate models which are not overfitted to the input data set. This same methodology was applied in a proof of concept phase set of software based sensors for a robot-in-the-loop sensing and control method were developed and tested with accurate results even with difficult to control input actions. This novel optimization based approach produced significantly better results than a traditional torque observer in a reasonable time frame using only open-loop simulation of a robot trajectory which could be precomputed. The success of this sensor concept validates the use of the high-quality simulation techniques of Sections 5 and 6 for entirely virtual sensing applications.

8 Conclusions

Modularized robots and their control software are of great benefit to robotic system prototyping and research thanks to their low cost and relatively high performance. This thesis covers work in several areas of quadruped robot platform research, particularly those dealing with hardware design and control architectures. Although mobile robots, such as the quadruped introduced here, may benefit from modularization the most, the methodology that drives the modularized robot design process can be applied to most robotic systems. The main contributions of this thesis include:

Chapter 1: The overall objectives and outline of the thesis were introduced.

Chapter 2: Background information on topics of robot design and simulation were discussed.

Chapter 3: A cost-conscious hardware platform designed to improve simulation stability and accuracy was developed through modularized designs. This tackles the problem of cost efficient robotic platforms for academic use, providing sufficient performance at an affordable cost.

Chapter 4: To minimize control and sensor quality loss due to the elimination of most sensors and high-performance controllers, a simulation integrated control framework was designed to accurately estimate the robot state even in situations limited by the hardware protocols and servomotor performance.

Chapter 5: The development of a rigid body physics based simulation subsystem sets the groundwork for the estimation of system properties. A focus on physical accuracy and high performance required custom solutions to various implementation problems.

Chapter 6: Novel motor models were developed for improved simulation through a tuned rigid body simulation system to provide state information without phase delays associated with the low communications bandwidth and reduced sensing capabilities of the platform. Open-loop testing of the resulting models validates their accuracy to several milliradians.

Chapter 7: Model parameter identification techniques using particle swarm optimization were used to calibrate motor models using a reduced set of data compared to competing techniques. The components of the prior chapters come together to allow for advances in soft-sensing, including the use of simulation based optimization to detect environment and robot payload changes during regular robot trajectories.

8.1 Future works

The research presented in this thesis furthers research capabilities in both hardware and software fields, delivering a modularized mobile robot platform and a robust and high performing framework to assist in the control of such a platform. Due to the complexity of the low-level implementations of the software framework and development time for hardware, several topics were left as proposed problems which can be solved with the presented techniques and deliverables. Among those are:

- Additional identification and verification of motor dynamics needs to be done to validate that the soft-sensing strategies are still effective for a full robot platform. A positive validation would eliminate the need for additional force and even IMU sensors from the robot for most control purposes, reducing the hardware platform cost and complexity.
- Non-trivial implementations of the soft-sensor technique could allow for more refined robot health tracking through minimal sensory input, including broken or underperforming motors and IMU drift. The applications of this extend well beyond the quadruped platform and could be extended to non-modular robots as well.
- Implementation of preliminary motion and SLAM methods are required for the robot control framework to achieve the desired initial release state. The addition of these components would improve the use case for this platform and facilitate research in those areas.

References

- [1] Sony Corporation, "<Press Release> Sony Launches Four-Legged Entertainment Robot," Sony Corporation, Tokyo, 1999.
- [2] S. Aizawa, Artist, *Aibo ERS-220 - Robosquare*. [Art]. 2009.
- [3] Boston Dynamics, LittleDog Robot 1.0 User Guide, Cambridge: Boston Dynamics, 2005.
- [4] M. P. Murphy, A. Saunders, C. Moreira, A. A. Rizzi and M. Raibert, "The LittleDog robot," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 145-149, 1 February 2011.
- [5] United States Defense Advance Research Projects Agency, "Legged Squad Support System (LS3) (Archived)," 2013. [Online]. Available: <https://www.darpa.mil/program/legged-squad-support-system>. [Accessed 10 1 2017].
- [6] C. Semini, "HyQ - Design and Development of a Hydraulically Actuated Quadruped Robot," Istituto Italiano di Tecnologia, Genova, 2010.
- [7] H. Khan, "MiniHyQ- Development of a Lightweight Highly-Dynamic Hydraulic Quadruped Robot," Istituto Italiano di Tecnologia (IIT), Genoa, 2015.
- [8] L. Sobehart, "Biomimetic Climbing Gaits for a Snake Robot," Hokkaido University, Sapporo, 2013.
- [9] D. Rollinson and H. Choset, "Virtual Chassis for Snake Robots," in *International Conference on Intelligent Robots and Systems*, San Francisco, 2011.
- [10] D. Rollinson, A. Buchan and H. Choset, "Virtual Chassis for Snake Robots: Definition and Applications," *Advanced Robotics*, vol. 26, no. 17, pp. 2043-2064, 2012.
- [11] M. Akdag, H. Karagulle and L. Malgaca, "An integrated approach for simulation of mechatronic systems applied to a hexapod robot," *Mathematics and Computers in Simulation*, vol. 82, pp. 818-835, 2012.
- [12] C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov and G. Pratt, "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 494-506, 2015.
- [13] A. Boeing and T. Braunl, "Evaluation of real-time physics simulation systems," in *GRAPHITE '07 Proceedings of the 5th international conference on Computer graphics and interactive techniques*, Perth, 2007.

- [14] E. Etien, "Modeling and simulation of soft sensor design for real-time speed estimation, measurement and control of induction motor," *ISA Transactions*, vol. 52, pp. 358-364, 2012.
- [15] P. Pillay and R. Krishnan, "Modeling, Simulation, and Analysis of Permanent-Magnet Motor Drives , Part I: The Permanent-Magnet Synchronous Motor Drive," *IEEE Transactions on Industry Applications*, vol. 25, no. 2, pp. 265-273, 1989.
- [16] P. Pillay and R. Krishnan, "Modeling, Simulation, and Analysis of Permanent-Magnet Motor Drives, Part 11: The Brushless DC Motor Drive," *IEEE Transactions on Industry Application*, vol. 25, no. 2, pp. 274-279, 1989.
- [17] P. Sobanski and T. Orłowska-Kowalska, "Faults diagnosis and control in a low-cost fault-tolerant induction motor drive system," *Mathematics and Computers in Simulation*, 2015.
- [18] J. Lima, J. Gonçalves, P. Costa and A. Moreira, "Humanoid realistic simulator: the servomotor joint modeling," in *6th International Conference on Informatics in Control, Automation and Robotics*, Milan, 2009a.
- [19] A. Mensink, "Characterization and Modeling of a Dynamixel Servo," University of Twente, Enschede, 2008.
- [20] J. M. Esposito and R. V. Kumar, "Efficient dynamic simulation of robotic systems with hierarchy," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, Seoul, 2001.
- [21] M. Gor, P. Pathak, A. Samantaray, J.-M. Yang and S. Kwak, "Control oriented model-based simulation and experimental studies on a compliant legged quadruped robot," *Robotics and Autonomous Systems*, vol. 72, pp. 217-234, October 2015.
- [22] D. Baraff, "Fast Contact Force Computation for Nonpenetrating Rigid Bodies," in *SIGGRAPH 94 21st International ACM Conference on Computer Graphics and Interactive Techniques*, Orlando, 1994.
- [23] E. Catto, "Iterative Dynamics with Temporal Coherence," in *Game Developer Conference*, Menlo Park, 2005.
- [24] B. V. Mirtich, "Impulse Based Dynamic Simulation of Rigid Body Systems," University of California, Berkeley, 1996.
- [25] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, "USARSim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*, Rome, 2007.
- [26] O. Michel, "WebotsTM: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 40-43, 2004.

- [27] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, 2004.
- [28] . Sprowitz, A. Tuleu, M. Vespignani, M. Ajallooeian, . Badri and A. Jan Ijspeert, "Towards Dynamic Trot Gait Locomotion: Design, Control, and Experiments with Cheetah-cub, a Compliant Quadruped Robot," *International Journal of Robotics Research*, vol. 32, no. 8, pp. 933-951, July 2013.
- [29] E. Coumans, "Bullet 2.83 Physics SDK Manual," 4 May 2015. [Online]. Available: https://github.com/erwincoumans/bullet3/blob/master/docs/Bullet_User_Manual.pdf. [Accessed 4 May 2015].
- [30] R. L. Smith, "Manual - ODE Wiki," 15 December 2011. [Online]. Available: http://ode-wiki.org/wiki/index.php?title=Manual:_All. [Accessed 8 November 2012].
- [31] A. Maciel, T. Halic, Z. Lu, L. P. Nedel and S. De, "Using the PhysX engine for Physics-based Virtual Surgery with Force Feedback," *International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 5, no. 3, pp. 341-353, 2009.
- [32] S. Ghosh and S. Moorthy, "Elastic-plastic analysis of arbitrary heterogeneous materials with the Voronoi Cell finite element method," *Computational Methods in Applied Mechanics and Engineering*, vol. 121, no. 1-4, pp. 373-409, 1995.
- [33] E. G. Parker and J. F. O'Brien, "Real-Time Deformation and Fracture in a Game Environment," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, New Orleans, 2009.
- [34] C. V. Nielsen, W. Zhang, L. M. Alves, N. Bay and P. Martins, "Finite Element Formulations," in *Modeling of Thermo-Electro-Mechanical Manufacturing Processes*, London, Springer-Verlag London, 2013, pp. 3-9.
- [35] G. Debunne, M. Desbrun, M.-P. Cani and A. H. Barr, "Adaptive Simulation of Soft Bodies in Real-Time," in *Computer Animation 2000. Proceedings*, Philadelphia, 2000.
- [36] L. Sobehart and H. Harada, "Distance based gait control for a pole climbing robotic snake," in *Proceedings of the 2013 JSME Conference on Robotics and Mechatronics*, Tsukuba, 2013.
- [37] A. Crespi, A. Badertscher, A. Guignard and A. Jan Ijspeert, "AmphiBot I: an amphibious snake-like robot," *Robotics and Autonomous Systems*, vol. 50, no. 4, pp. 163-175, 2005.
- [38] M. Friedmann and K. v. S. O. Petersen, "Adequate motion simulation and collision detection for soccer playing humanoid robots," *Robotics and Autonomous Systems*, vol. 57, no. 8, pp. 786-795, 2009.

- [39] B. Armstrong, O. Khatib and J. Burdick, "The explicit dynamic model and inertial parameters of the PUMA 560 arm," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on.*, San Francisco, 1986.
- [40] C. Wright, A. Buchan, B. Brown, J. Geist, M. Schwerin, D. Rollinson, M. Tesch and H. Choset, "Design and Architecture of the Unified Modular Snake Robot," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, St. Paul, 2012.
- [41] J. M. Hsu and C. S. Peters, "Extending Open Dynamics Engine for the DARPA Virtual Robotics Challenge," in *4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings*, Bergamo, 2014.
- [42] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella and D. G. Caldwell, "Design of HyQ – a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831-849, 2011.
- [43] J. Z. Kolter, M. P. Rodgers and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, Pasadena, 2008.
- [44] J. Craig, *Introduction to Robotics*, Upper Saddle River: Pearson Prentice Hall, 2005.
- [45] M. Quigley, K. Conley, B. Gerkey, F. Josh, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, Kobe, 2009.
- [46] E. Einhorn, T. Langner, R. Stricker, C. Martin and H.-M. Gross, "MIRA - Middleware for Robotic Applications," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2012)*, Vilamoura, 2012.
- [47] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425-432, 1980.
- [48] R. Lisovy, M. Sojka and Z. Hanzalek, "PCI Express as a Killer of Software-based Real-Time Ethernet," Czech Technical University in Prague, Prague, 2013.
- [49] D. C. Schmidt, M. Deshpande and C. O'Ryan, "Operating System Performance in Support of Real-time Middleware," in *Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems, 2002 (WORDS 2002)*, San Diego, 2002.
- [50] L. Sobehart and H. Harada, "Smart-Servo Control by Rigid Body Physics Based Methods," in *Proceedings of The 12th International Conference on Motion and Vibration MOVIC 2014*, Sapporo, 2014.

- [51] D. H. U. Kochanek and R. H. Barteles, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *Computer Graphics*, vol. 18, no. 3, pp. 33-41, 1984.
- [52] R. L. Dougherty, A. Edelman and J. M. Hyman, "Nonnegativity-, Monotonicity-, or Convexity-Preserving Cubic and Quintic Hermite Interpolation," *Mathematics of Computation*, vol. 52, no. 186, pp. 471-494, 1989.
- [53] Future Technology Devices International Ltd., "Software Application Development D2XX Programmer's Guide," Future Technology Devices International Ltd., Glasgow, 2012.
- [54] Future Technology Devices International Ltd., "AN232B-04 Data Throughput, Latency and Handshaking," Future Technology Devices International Ltd., Glasgow, 2006.
- [55] Robotis Inc., "MX-28T / MX-28R," Robotis Inc., Irvine, 2010.
- [56] T. Saito, T. Kagiwada and H. Harada, "Development of an Earthworm Robot with a Shape Memory Alloy and Braided Tube," *Advanced Robotics*, vol. 23, no. 12-13, pp. 1743-1760, 2009.
- [57] ROS.org, "ODE Solvers," Willow Garage, 16 10 2010. [Online]. Available: http://ros.org/wiki/physics_ode/ODE. [Accessed 30 1 2013].
- [58] R. P. Sathe and A. T. Lake, "Collision detection of concave bodies using art gallery problem and cube maps". United States of America Patent US20080158225 A1, 28 December 2006.
- [59] S. Zapolsky and E. M. Drumwright, "Adaptive Integration for Controlling Speed vs. Accuracy in Multi-Rigid Body Simulation," George Washington University, Washington DC, 2014.
- [60] F. Enner, D. Rollinson and H. Choset, "Simplified Motion Modeling for Snake Robots," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, Saint Paul, 2012.
- [61] I. Virgala and M. Kelemen, "Experimental Friction Identification of a DC Motor," *International Journal of Mechanics and Applications*, vol. 3, no. 1, pp. 26-30, 2013.
- [62] A. S. Martinez, R. S. González and C. A. S. Terçariol, "Continuous growth models in terms of generalized logarithm and exponential functions," *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 23, pp. 5679-5687, 2008.
- [63] L. Sobehart and H. Harada, "Modular Quadruped Control and Simulation Platform," in *The Robotics and Mechatronics Conference 2015*, Kyoto, 2015.
- [64] M. Gautier and S. Briot, "Dynamic parameter identification of a 6 DOF industrial robot using power model," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, Karlsruhe, 2013.

- [65] M. Mistry, S. Schaal and K. Yamane, "Inertial parameter estimation of floating base humanoid systems using partial force sensing," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, Paris, 2009.
- [66] B. Niu, Y. Zhu, X. He and H. Wu, "MCP SO: A multi-swarm cooperative particle swarm optimizer," *Applied Mathematics and Computation*, vol. 185, pp. 1050-1062, 2007.
- [67] T. Dall Larsen, N. A. Andersen and O. Ravn, "Incorporation of Time Delayed Measurements in a Discrete-time Kalman Filter," *Proceedings of the 37th IEEE Conference on Decision & Control*, vol. 1, no. 1, pp. 3972-3977, 1998.
- [68] J. A. Smith and J. Jivraj, "Analysis of Robotis Dynamixel AX-12+ Actuator," in *Symposium on Brain, Body and Machine*, Montreal, 2010.
- [69] ROBOTIS, "AX-12/ AX-12+/ AX-12A," 2010.
- [70] M. Raibert, K. Blankespoor, G. Nelson and R. Playter, "BigDog, the Rough-Terrain Quadruped Robot," in *Proceedings of the 17th IFAC World Congress*, COEX, South Korea, 2008.
- [71] D. Mansolino, "Mobile Robot modeling, Simulation and Programming," Ecole Polytechnique Federale De Lausanne, Lausanne, 2013.
- [72] Boston Dynamics, "Bigdog Overview," 22 November 2008. [Online]. Available: http://www.bostondynamics.com/img/BigDog_Overview.pdf. [Accessed 10 January 2013].
- [73] E. Coumans, "Scaling The World," Game Physics Simulation, 22 February 2009. [Online]. Available: <http://bulletphysics.org/mediawiki-1.5.8/index.php>. [Accessed 10 2 2012].
- [74] E. Weitnauer and R. R. H. Haschke, "Evaluating a Physics Engine as an Ingredient for Physical Reasoning," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, Darmstadt, 2010.
- [75] E. Kokkevis, "Practical Physics for Articulated Characters," in *Game Developers Conference*, San Jose, 2004.
- [76] J. Hummel, R. Wolff, T. Stein, A. Gerndt and T. Kuhlen, "An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations," *Lecture Notes in Computer Science*, vol. 7432, pp. 346-357, 2012.

Acknowledgements

This work would also not have been possible without the assistance from my supervisor, Dr. Harada, and everyone else in the Laboratory of Smart Mechanisms. Their support at a technical and creative level helped inspire many additional projects that were instrumental in the development and improvement of the robot hardware and software. I would also like to thank Dr. Kajiwara, Dr. Kobayashi, and Dr. Konno for reviewing and providing insight into areas of improvement for this thesis and research.

I would like to thank my wife, Yurie, for helping me through this years long project, always refusing to let me achieve anything less than my best. I would also like to thank my parents, without whom this thesis would not have been possible. Their encouragement, support, and at times helpful criticism over the years allowed me to focus on learning and practicing new fields of study that would otherwise not been open to me.

Appendix A. Hardware Models and Specifications

Proper system dimensioning is required to properly control and simulate the primary quadruped robot, especially the distances which define the kinematic representation of the robot as given by the location of each motor rotational axis and end effector, in this case leg endpoint, location. The following figures provide the exact dimensions used for the manufacture of components and their subsequent modeling.

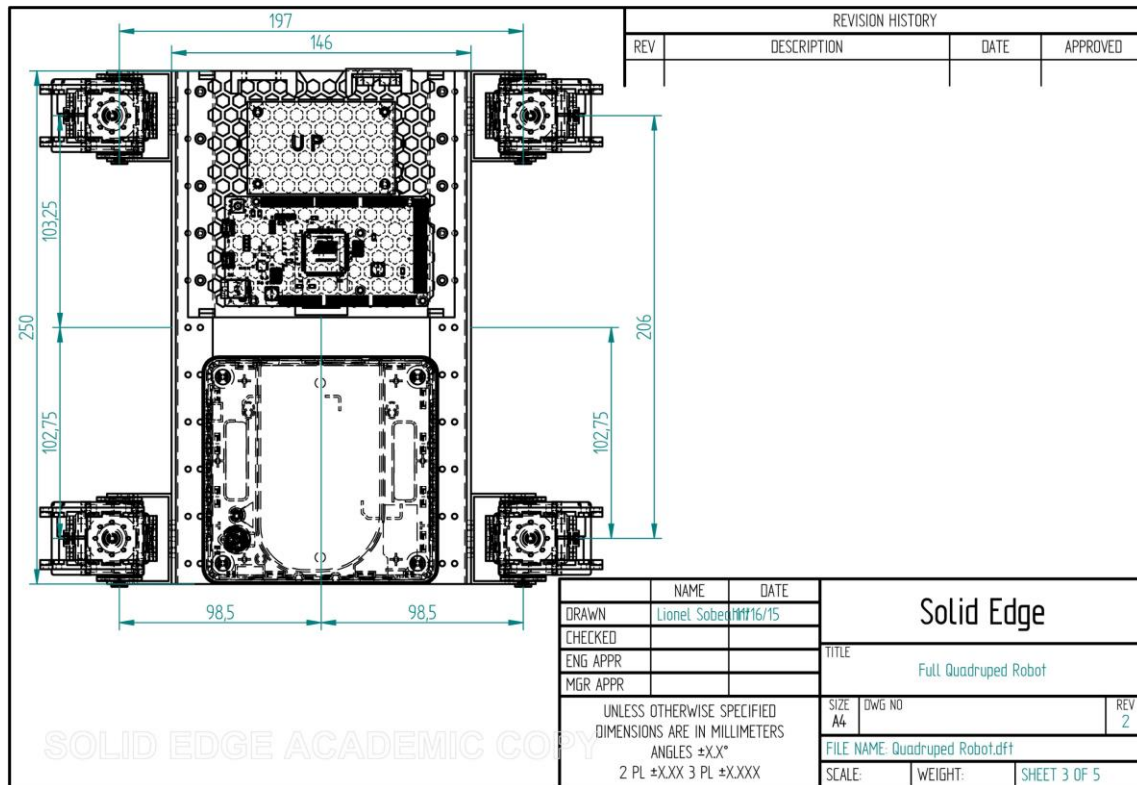


Figure A.1: CAD design specifications for the quadruped robot, top view.

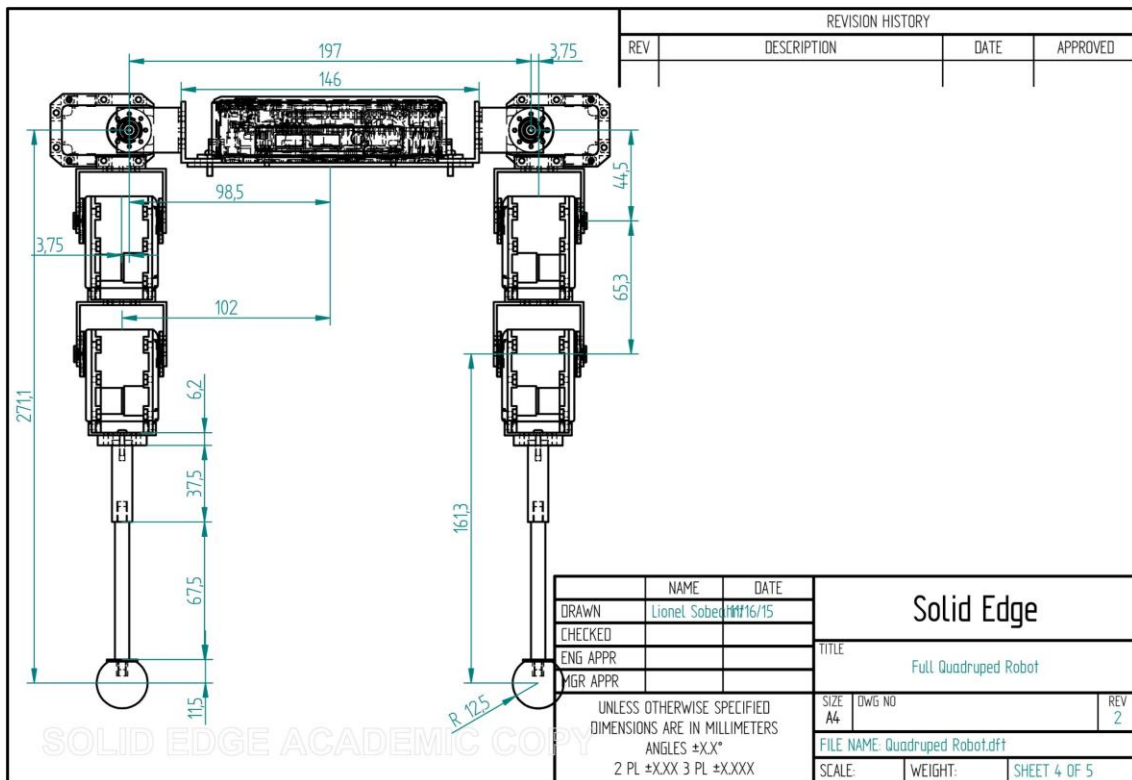


Figure A.2: CAD design specifications for the quadruped robot, front view.

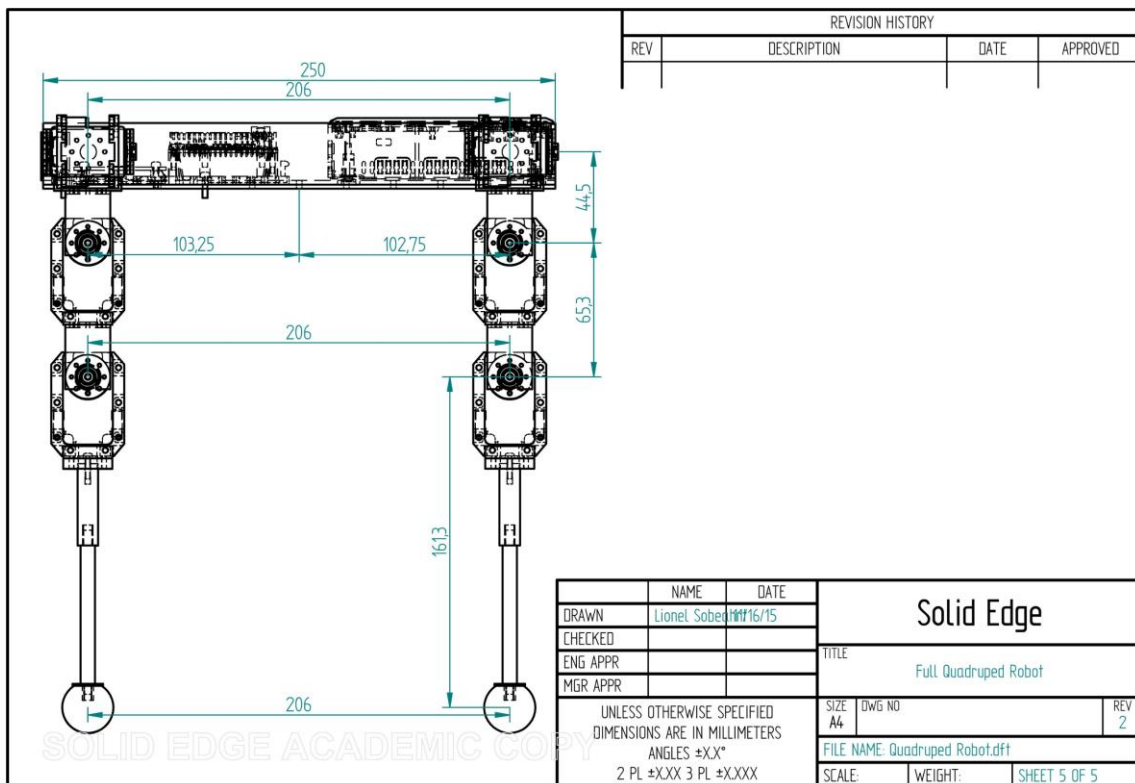


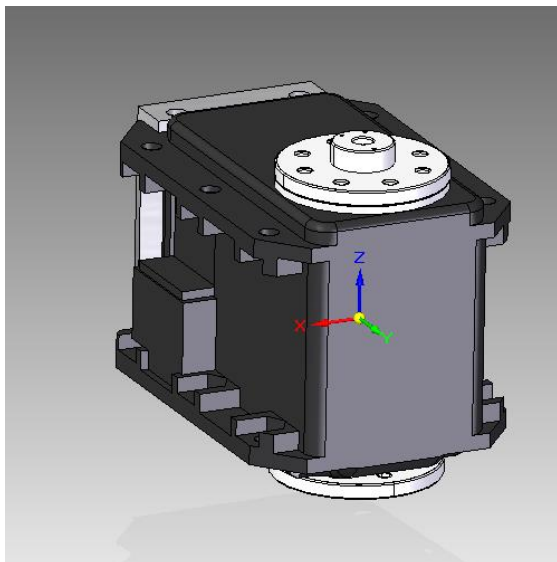
Figure A.3: CAD design specifications for the quadruped robot, side view.

Model specifications are important to simulation accuracy. The following are CAD model based mass and inertial estimations used within the software framework and simulation. This is only an excerpt of

the components and inertial properties, with additional combinations of components may produce different per-joint overall inertias.

Table A.1: Module component inertias for key parts. Inertias in the final system may be different due to additional hardware such as cables and screws.

Regular Module



Physical Properties for Selected Parts
 volume= 44.77125 cm³
 mass= 79.41090 g
 User Quantity Mass= 79.41090 g
 With respect to the Global Coordinate System

Center of Mass:

X= 0.00000 cm

Y= -1.25773 cm

Z= 0.00624 cm

Center of Volume:

X= 0.00000 cm

Y= -1.21415 cm

Z= 0.01040 cm

Mass Moments of Inertia:

Ixx= 368.63881 g-cm²

Iyy= 138.65571 g-cm²

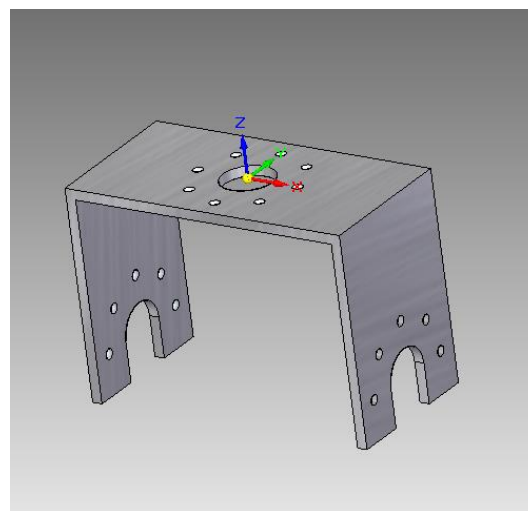
Izz= 331.94459 g-cm²

Ixy= -0.00042 g-cm²

Ixz= -0.00053 g-cm²

Iyz= 3.47246 g-cm²

Horn Connector



Physical Properties for Selected Parts
 volume= 3.085133 cm³
 mass= 8.622947 g
 density= 2.795000 g/cm³
 With respect to the Global Coordinate System

Center of Mass:

X= -0.000000 cm

Y= 0.000000 cm

Z= -0.880368 cm

Center of Volume:

X= -0.000000 cm

Y= 0.000000 cm

Z= -0.880368 cm

Mass Moments of Inertia:

Ixx= 18.159383 g-cm²

Iyy= 42.763781 g-cm²

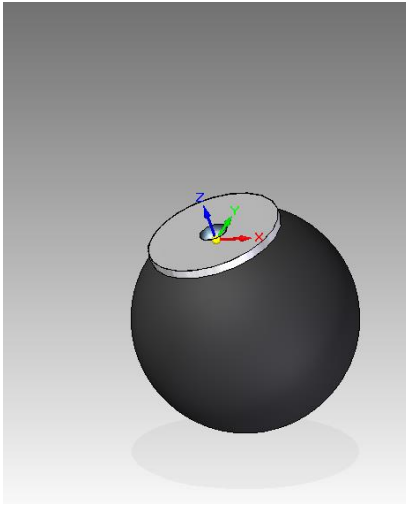
Izz= 32.161711 g-cm²

Ixy= 0.000000 g-cm²

Ixz= 0.000000 g-cm²

Iyz= -0.000000 g-cm²

Foot Ball



Physical Properties Report

volume= 8.143 cm³

mass= 9.48259 g

User Quantity Mass= 9.48259 g

With respect to the Global Coordinate System.

Center of Mass:

X= 0.00000 cm

Y= 0.00000 cm

Z= -1.00837 cm

Center of Volume:

X= 0.00000 cm

Y= 0.00000 cm

Z= -1.05263 cm

Mass Moments of Inertia:

Ixx= 15.765 g-cm²

Iyy= 15.764 g-cm²

Izz= 5.793 g-cm²

Ixy= 0.000 g-cm²

Ixz= 0.000 g-cm²

Iyz= 0.000 g-cm²

Foot Extender (medium)



Physical Properties for Selected Parts

volume= 8.361 cm³

mass= 22.67587 g

User Quantity Mass= 22.67587 g

With respect to the Global Coordinate System

Center of Mass:

X= 0.02019 cm

Y= 0.00000 cm

Z= 3.37675 cm

Center of Volume:

X= 0.02019 cm

Y= 0.00000 cm

Z= 3.37675 cm

Mass Moments of Inertia:

Ixx= 518.096 g-cm²

Iyy= 518.106 g-cm²

Izz= 8.681 g-cm²

Ixy= 0.000 g-cm²

Ixz= 1.546 g-cm²

Iyz= 0.000 g-cm²

Appendix B. Control Framework Implementation

The way in which the control framework fits together and operates during typical operation is determined more by the end use than the framework design. As the system is highly modular, it is possible to configure the framework in any number of ways, however due to performance considerations there are two primary configurations. The first of these is the fully asynchronous option, which allows for unique clocks to be used for each time-dependent module. Each module which updates based on external events is operated in a separate thread with its own clock and execution loop as shown in Figure B.1.

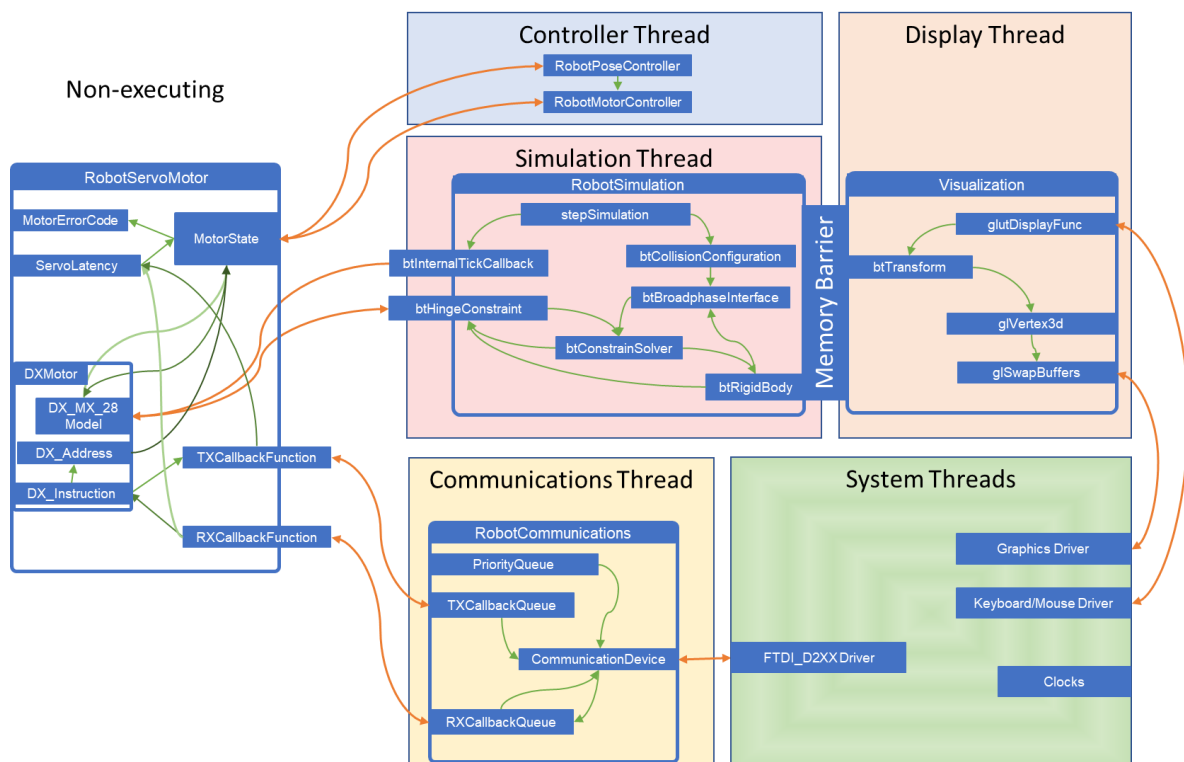


Figure B.1: Major control framework modules and their respective execution threads in a fully asynchronous configuration. Subroutine flows start at the top of each executing section and follow the green arrow, with orange arrow delineating data transfer outside the thread.

This configuration allows for the greatest flexibility in update cycles for all modules even in hardware limited situations where fully synchronous systems would not be able to execute reliably. The core software components were designed with this type of operation in mind, with the passive `RobotServoMotor` class acting as a synchronization mechanism which translates between the execution rates of the various modules.

In practice, it is often easier to work with data sets and states which are more synchronous in nature, particularly when dealing with multiple simulations which use identical internal clocks. At the same time, certain subroutines must be kept independent to avoid hardware and software limitations which can be made exponentially worse through a fully synchronized system. Due to the modularity of the control framework, it is possible to selectively add modules to a synchronous control system as shown in Figure B.2

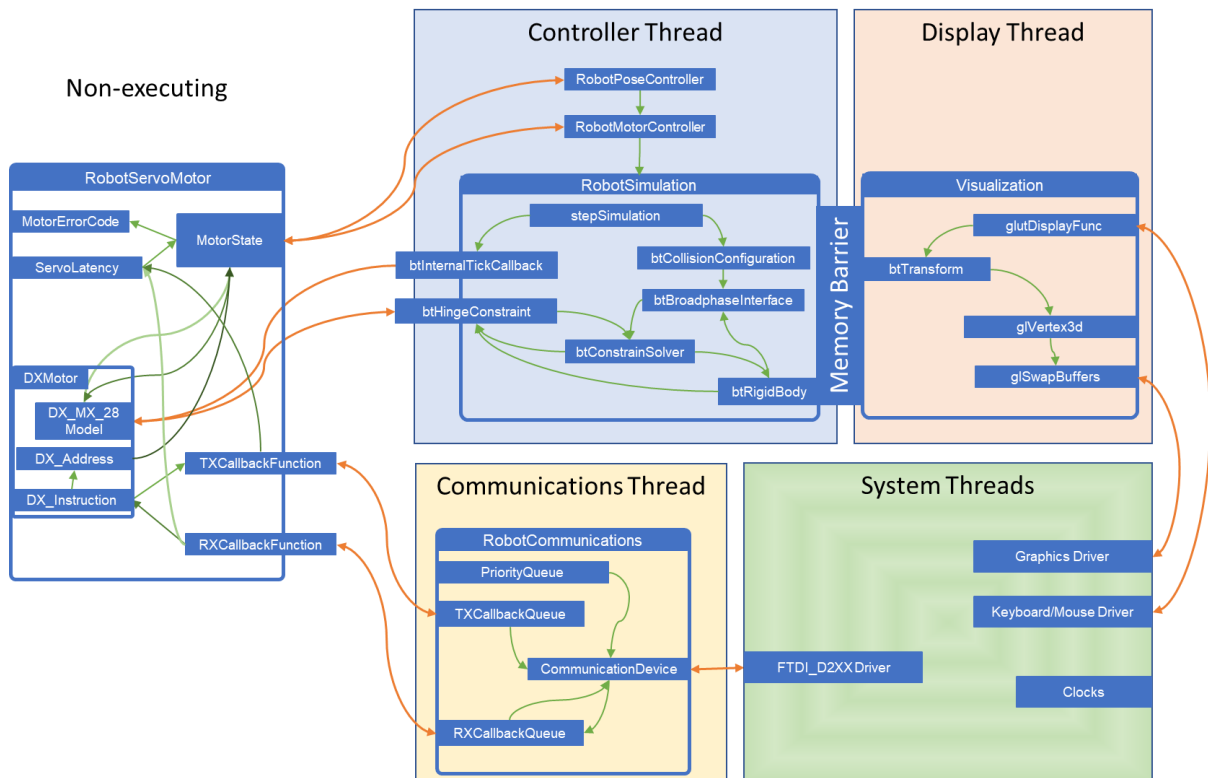


Figure B.2: Major control framework modules and their respective execution threads in a controller and simulation synchronous configuration. Subroutine flows start at the top of each executing section and follow the green arrow, with orange arrow delineating data transfer outside the thread.

With this structure, the hardware dependent modules, communications and visualization, can be executed separately at their optimal rates, while the controller and simulation components operate in a synchronous method to better optimize data collection. When operating with a display thread there can be some additional overhead due to the memory barrier approach implemented in the current revision, however the simulation itself is often orders of magnitude more computationally expensive and collisions do not affect performance in any significant manner.

Framework hierarchy

The software framework utilizes language level polymorphism, templates, and anonymous functions to modularize the system while retaining very high execution performance. Figure B.3 shows a partial code map with hierarchy structuring from highly specified to highly abstracted classes. The core functionality of each module requires interface information from other core classes, however the implementation of the interface is abstracted from each derived class. This maintains modularity on the software level to a high degree, while avoiding performance issues that can arise from fully anonymous methodologies such as ROS.

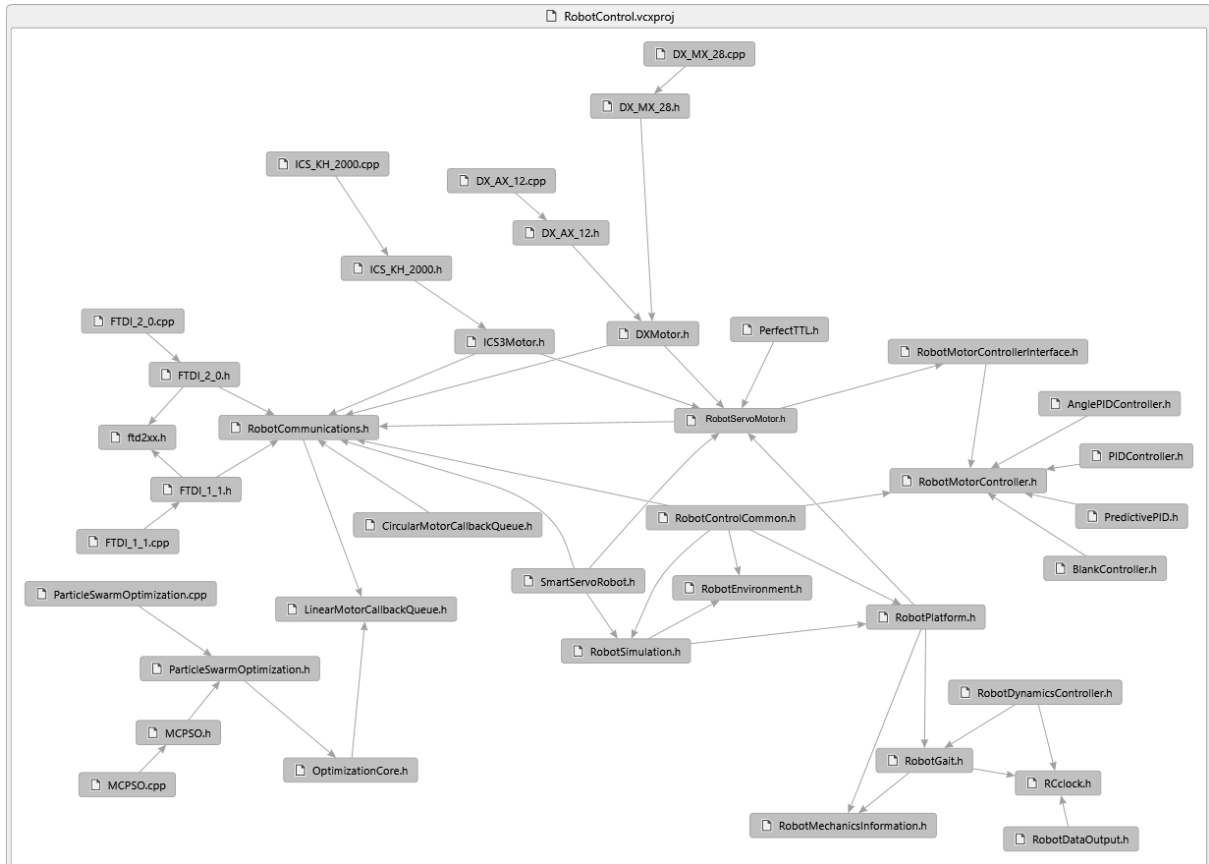


Figure B.3: Major control framework modules and their components. Classes are centered about the *RobotServoMotor* and *RobotCommunications* super-classes.

Although this concept was implemented throughout the framework, an exception was required for integration of the simulation components due to the widely varying nature of physical simulation software APIs. As Bullet Physics Engine was chosen early on for the primary physics simulation, motor models and internal data structures were based in the Bullet Physics Engine implementation and used in the *RobotServoMotor* and *RobotPlatform* classes as shown in . In particular, motor models were directly implemented in derived classes as *btHingeConstraint* types, and *btScalar*, *btVector*, and *btTransform* classes were used throughout the system for their simplicity in computing vectorized computations with SIMD optimizations for high performance.

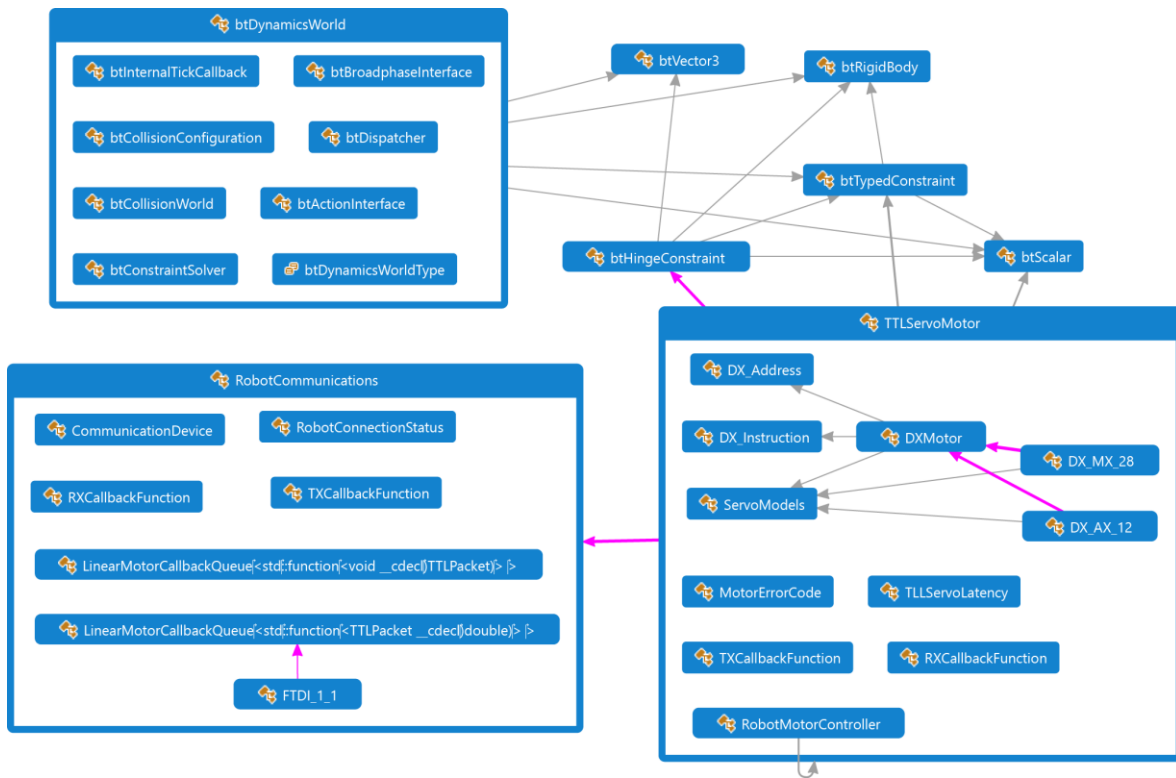


Figure B.4: Core components of the integrated simulation and control framework showing the integration between the simulation and other components.

Although this breaks modularity with the simulation implementation, the overall construction and efficiency of the code is significantly increased. This implementation does not require the simulation to be included for operation, maintaining the modularity in a functional aspect even with the code dependency in derived classes. Future work could isolate and integrate custom functions with applicable plugins for simulation as necessary, albeit at a significant development time cost.

Publications

International Peer Reviewed Journals and Conferences

1. Sobehart Lionel, Harada Hiroyuki,” High performance rigid body simulation of modularized robots using constraint-based models”, Mathematics and Computers in Simulation, Vol.144, pp.91-107, (2018)
2. Sobehart Lionel, Harada Hiroyuki, “Optimization Based Soft-sensing Framework for Modularized Robots”, Proceedings of 2016 IEEE/SICE International Symposium on System Integration, pp.391-396, WeA1A.6, (2016)
3. Sobehart Lionel, Harada Hiroyuki, “Smart-Servo Control by Rigid Body Physics Based Methods”, Proceedings of 12th International Conference on Motion and Vibration Control (USB), 1D31, (2014)

Other Conferences

1. Sobehart Lionel, Harada Hiroyuki, “Modular Quadruped Control and Simulation Platform”, The Robotics and Mechatronics Conference 2015 (2015)
2. Sobehart Lionel, Harada Hiroyuki, “Physical Simulation of Low-cost “Smart” Servomotors”, The Robotics and Mechatronics Conference 2014 (2014)
3. Sobehart Lionel, Harada Hiroyuki, “Distance based gait control for a pole climbing robotic snake”, The Robotics and Mechatronics Conference 2013 (2013)