

Copyright  
by  
Joshua Alexander James  
2016

The thesis committee for Joshua Alexander James  
certifies that this is the approved version of the following thesis:

## **Additive Robot Systems**

APPROVED BY

SUPERVISING COMMITTEE:

---

Luis Sentis, Supervisor

---

Chien-Liang Fok

# **Additive Robot Systems**

**by**

**Joshua Alexander James, B.S.E.E.**

## **THESIS**

Presented to the Faculty of the Graduate School of  
The University of Texas at Austin  
in Partial Fulfillment  
of the Requirements  
for the Degree of

**Master of Science in Engineering**

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2016

# Additive Robot Systems

Joshua Alexander James, M.S.E.

The University of Texas at Austin, 2016

Supervisor: Luis Sentis

Additive robot systems are created by “adding” together heterogeneous robot modules at clearly defined interfaces. Each robot module is aware of its geometry, dynamics, primary function, and capabilities. Communication between modules allows the system as a whole to perform complex human-centric tasks, even when the system is assembled with arbitrary structure. Additive robot systems have the potential to revolutionize the robotics industry. Reusable mass-produced modules and automated system integration allow complex robots to be created at a fraction of the cost and effort of existing solutions. The ability to quickly optimize additive robot system hardware for a specific task is particularly well suited to real-world scenarios in which the tasks to be performed are not known beforehand, including disaster relief, space exploration, and flexible manufacturing scenarios. This work presents a framework for module interfacing, modeling, and control in additive robot systems.

# Table of Contents

<b>Abstract</b>	<b>iv</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>viii</b>
<b>Chapter 1. Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Contributions . . . . .	3
1.2.1 Scalable Dock Design . . . . .	3
1.2.2 Architecture . . . . .	4
1.2.3 Distributed Module Dynamics Decoupling . . . . .	5
1.2.4 Whole-body Controller for Modular Robots . . . . .	6
1.3 Outline . . . . .	6
<b>Chapter 2. Architecture</b>	<b>8</b>
2.1 Overview . . . . .	8
2.2 Physical Interfaces . . . . .	10
2.3 Self-Identity . . . . .	17
2.3.1 Functionality Primitives . . . . .	18
2.3.2 Physical Model . . . . .	24
2.3.3 Relationships and Status . . . . .	27
2.3.4 Static Self-Identity Specification . . . . .	28
2.4 Architectural Algorithms . . . . .	30
2.4.1 Module Connection . . . . .	30
2.4.2 Module Failure Detection . . . . .	32
2.4.3 System Self-Identity Generation . . . . .	32
2.4.4 Security . . . . .	34

<b>Chapter 3. Module Decoupling</b>	<b>36</b>
3.1 Overview . . . . .	36
3.2 Model Compression . . . . .	39
3.3 Centralized Module Decoupling . . . . .	41
3.4 Distributed Module Decoupling . . . . .	42
<b>Chapter 4. Whole-Body Control</b>	<b>51</b>
4.1 Overview . . . . .	51
4.2 Task Prioritization . . . . .	52
4.3 Joint-space Tasks . . . . .	53
4.4 Goal-space Tasks . . . . .	54
4.5 CoM Control . . . . .	57
4.6 Centralized Optimization . . . . .	61
<b>Chapter 5. Future Work</b>	<b>63</b>
<b>Bibliography</b>	<b>66</b>

## List of Tables

2.1	Docking Logic Table	15
2.2	Functionality Primitives	22

# List of Figures

1.1	Atlas and Valkyrie	1
1.2	Example additive robot system	2
2.1	Mobile Manipulator Assembly	9
2.2	Modular Hierarchy for Mobile Manipulator	9
2.3	Radially-symmetric interlocks	11
2.4	Possible dock design	12
2.5	Docking procedure	14
2.6	Four components of self-identity	17
2.7	Functionality primitives	18
2.8	Industrial Robot Functionality Tree	22
2.9	Cleaning Robot Functionality Tree	23
2.10	Mobile Manipulator Functionality Tree	23
2.11	Quadcopter Functionality Tree	24
2.12	Humanoid Robot Functionality Tree	24
2.13	Physical models	25
2.14	Relationships and status	27
2.15	Example module	29
2.16	Module Connection MSC Diagram	31
2.17	Module Removal MSC Diagram	33
2.18	Flipping Due To Dock Change	35
3.1	Module decoupling layer	36
3.2	Centralized Module Decoupling MSC Diagram	41
3.3	Synchronous Module Decoupling MSC Diagram	45
3.4	Asynchronous Module Decoupling MSC Diagram	45
4.1	Whole-body Control	51

4.2 Single (a) vs. alternatively valid (b) targets and (c) a 2D visualization of a goal region (green). . . . .	54
4.3 Illustration of one-dimensional goal error . . . . .	56

## 1.1 Problem Statement

Robot systems today are created by teams of skilled engineers over the course of many years, with the development of complex, monolithic robots like Atlas (Boston Dynamics) or Valkyrie (NASA-JSC) costing millions. The integration of different robot components (actuators, sensors, hardware, etc.) represents a large amount of the time and cost of such robots. Furthermore, the effort spent developing and integrating these systems is difficult to directly transfer to new robots; The developed subsystems are often interdependent and cannot be easily separated.

Additive robot systems are created by “adding” together robot modules at clearly defined interfaces (Figure 1.2). A diversity of robot systems can be created from a small set of hardware modules. Thus, development efforts can be trivially transferred to new robots by simply reusing modules. Each module



Figure 1.1: Atlas and Valkyrie

is less complex than a full robot as well, and can be tested more thoroughly, resulting in robots that exhibit more reliable and predictable behavior than the current state of the art.

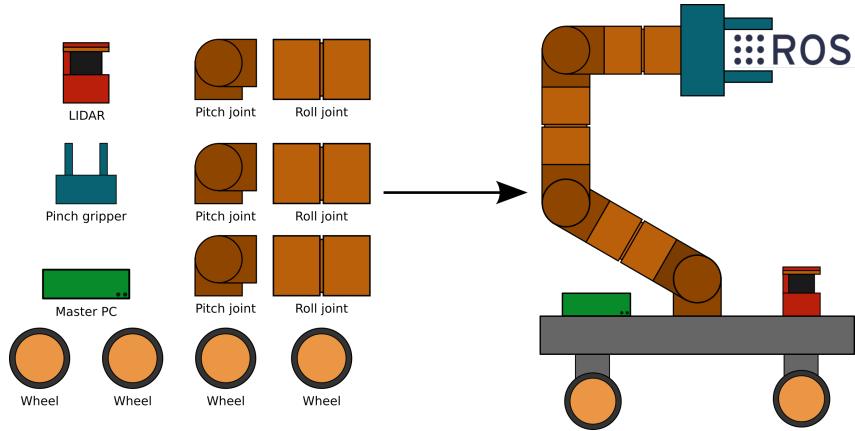


Figure 1.2: Example additive robot system

Each module in an additive robot system has a self-identity (SI)—an innate understanding of its physical form, purpose, health, and relationship with other modules. Combining these module SIs allows the full system to autonomously construct its own SI at runtime. This system SI provides information about the capabilities, form, and status of the entire robot. Inherent knowledge of module status allows the system to detect overtaxed or broken modules and specify exactly what needs to be replaced. Inherent knowledge of the form of the system ensures that whole-body planners and controllers always match the physical construction of the robot. Inherent knowledge of the system’s capabilities significantly increases autonomy: An additive robot system does not need to be told what parts of its body should be used for a

given task—it already understands which modules are useful for locomotion, manipulation, and perception.

Additive robot systems have the potential to revolutionize the robotics industry. Reusable mass-produced modules and automated system integration allow complex robots to be created at a fraction of the cost and effort of existing solutions. The ability to quickly optimize additive robot system hardware for a specific task is particularly well suited to real-world scenarios in which the tasks to be performed are not known beforehand, including disaster relief, space exploration, and flexible manufacturing scenarios.

Since most of the integration is automated, it also lowers the level of expertise needed to construct complex robots. If mass production of the modules brings them into the average consumers price range, research-grade robotics projects become available to hobbyists. Giving the general public the ability to easily create state-of-the-art robots has the potential to be as revolutionary as the PC, 3D printing, and autonomous quadcopters have been for computing, consumer manufacturing, and aerial robotics.

## 1.2 Contributions

### 1.2.1 Scalable Dock Design

The first contribution is a scalable, hermaphroditic docking interface design. The size and strength of the parameterized design can be easily scaled to support robots of all sizes. The radial symmetry of the design allows docking between docks of different sizes. The docking interface provides

actuated connection and disconnection of modules. This allows for toolless robot construction and automated module replacement.

A myriad of docking methods have been proposed over the years, including connectors that physically latch on to one another actively [19, 20, 22] or passively [10, 11, 33, 35, 36, 42], use magnets to stay connected [4, 6, 9, 18, 23, 24, 41, 45, 47, 49, 53], and even solder themselves together [30].

One thing all these docks have in common is that they are designed as a one-size-fits-all solution; They assume that a single dock design can be used for all applications. This assumption is valid for the systems they are designed for, since the systems often only have one type of module. However, a more broadly applicable system will need modules of a variety of sizes and strengths. Such a system thus needs docks that can be scaled while still remaining able to connect to one another.

### 1.2.2 Architecture

The second contribution is an architecture based on modular self-identity, as well as the recursive construction of system SI. The module functionality primitive system supports the modularization of existing designs in a variety of application domains. The presented SI definitions and algorithms are believed to be sufficiently flexible to allow for the inclusion of novel designs in the future, while not being so convoluted as to make them unusable.

Many frameworks for automatically modeling and controlling the kinematics and dynamics of a modular robot system have been proposed in the last

decade [3, 12, 13, 15, 16, 27, 51]. However, these frameworks are relatively inflexible, typically restricted to simple module models (e.g., a single joint described with D-H parameters) and / or a small set of predefined modules (particularly in lattice-type robot frameworks). The existing modeling systems are also often tailored to articulated industrial robots (e.g., serial chain manipulator arms) and do not capture many aspects of contemporary robot systems. The proposed self-identity-based architecture provides the following improvements over existing frameworks:

First, there are no restrictions on each module’s model; Each module can contain any number of links / joints / actuators / sensors arranged in any structure. The module’s model is stored inside the module itself, so the assembly process is also significantly more flexible than those that use a predefined module set.

Second, the functionality primitive system allows the robot to flexibly understand how to use each module. This is an improvement over current systems in which functionality is simply hardcoded in the predefined module list (i.e., this module is the designated tool module, and thus should be used for gripping).

### 1.2.3 Distributed Module Dynamics Decoupling

The third contribution is a distributed dynamics decoupling scheme for additive robot systems. This makes the construction of large-scale additive robot systems possible, where a centralized control scheme would be limited

by computation power.

Most existing modular robot architectures either use a model-based centralized controller, or no model-based controller at all (e.g., just using pure position control for each joint). To the best of the author’s knowledge, the only other distributed model-based control architecture is Virtual Decomposition Control [44]. The proposed distributed decoupling scheme differs from VDC in that it only attempts to passively decouple the dynamics of the module and provides an acceleration-based interface, while VDC provides feedback control with a trajectory-based interface. The use of accelerations allows for a wider range of high-level controllers. Notably it make possible the use of whole-body controllers, which typically do not perform well without a torque / acceleration interface to the hardware.

#### **1.2.4 Whole-body Controller for Modular Robots**

The fourth contribution is an automatically-synthesized whole-body controller. The controller supports joint-space, and goal-space tasks for SI-defined frames of interest. To the best of the author’s knowledge, this is the first whole-body controller that supports modular robots, and one of the few that extends to actuation methods beyond articulated joints.

### **1.3 Outline**

Chapter 2 presents the additive robot system architecture and introduces module / system SI. Chapter 3 describes distributed and centralized

techniques for decoupling the modules' dynamics. Chapter 4 provides methods for coordinating the movements of all the modules in the system to complete tasks. Chapter 5 illustrates avenues for the future improvement of additive robot systems.

# Chapter 2

## Architecture

### 2.1 Overview

Additive robot systems are composed of modules arranged in a branching structure. There are two structural requirements. First, there must be one coordinator module, the “master” module, designated by the user to serve as the user interfacing point and the coordinator of high-level behaviors. This module can be located anywhere in the structure. However, since the master module also defines the primary frame of the robot, it is recommended that it be placed somewhere conducive to human control. For example, using a quadcopter’s center point or a humanoid robot’s head would permit more intuitive teleoperation than, say, the tip of the robot’s finger.

The second requirement is that there must not be any explicit loops in the structure; the modules must always form a tree structure. Implicit kinematic loops, like those created by firmly grasping two rungs of a ladder, are handled by constraints.

Beyond these two requirements, an additive robot system can be arbitrarily structured. A simple example is a mobile manipulation robot (Figure 2.1) assembled from a master coordinator module, four wheel modules, six joint

modules, a gripper, and a LIDAR module. Figure 2.2 shows the module tree for such a robot. More complex examples can be seen in Section 2.3.1.

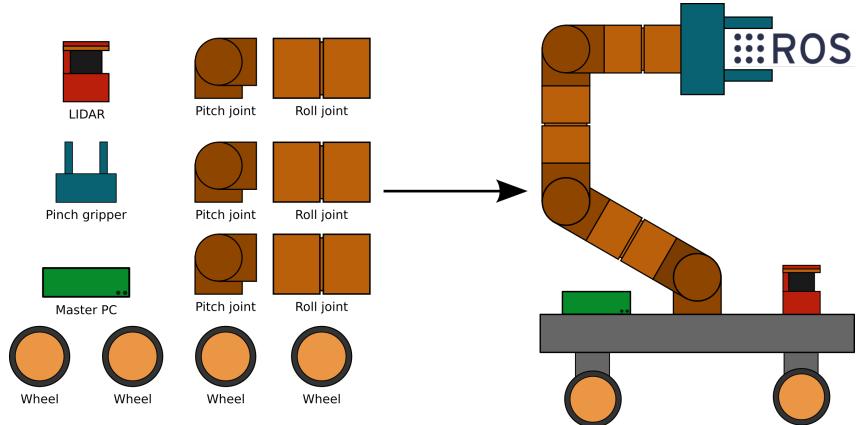


Figure 2.1: Mobile Manipulator Assembly

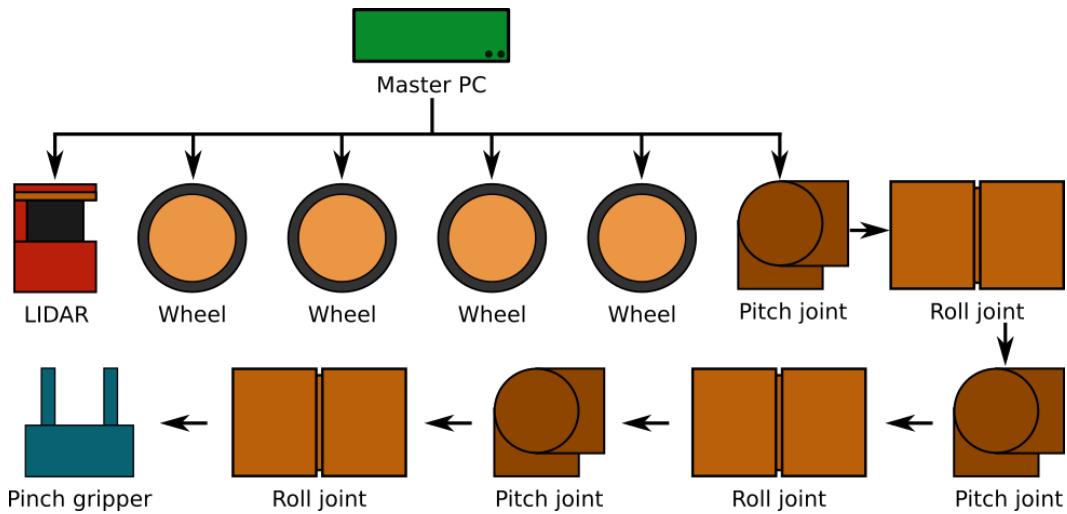


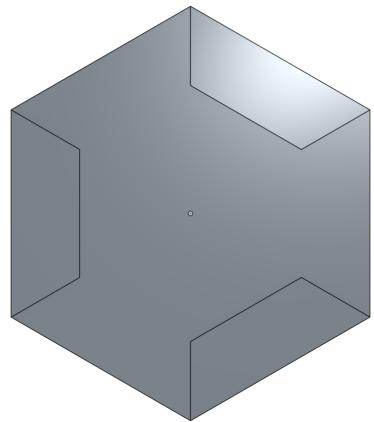
Figure 2.2: Modular Hierarchy for Mobile Manipulator

## 2.2 Physical Interfaces

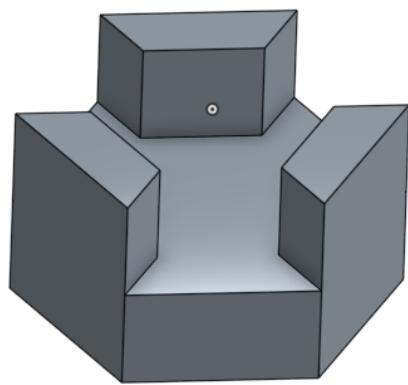
Modules physically attach to one another at “docks”. Docks are hardware features that rigidly connect one module to another and provide the ability to transfer power and communication signals. Ideally, all modules would use the a single standardized hermaphroditic dock, so that a dock on any module can be connected to any other. However, different robots have different connection needs. For example, connectors for an autonomous car need to be significantly different in size, strength, and power from those used in a quadcopter. This motivates a dock design that can be scaled while still maintaining interoperability.

One way to accomplish this is to use an outer ring of radially symmetric mechanical interlocks in conjunction with bilaterally mirrored mechanical and electrical connections. The strength of such a dock can be increased by enlarging the outer ring without changing the center. If the power connections are placed in the outer ring, the power transfer capacity naturally scales with the size of the dock as well. Increasing the strength and power in this way thus allows docks of any scale to mate (Figure 2.3). Furthermore, using bilateral mirroring for the mechanical and electrical connections ensures that the docking mechanism is hermaphroditic. This is important because a gendered docking system by definition restricts the ways in which modules can connect; The use of genderless connectors increases the flexibility of additive robot system design.

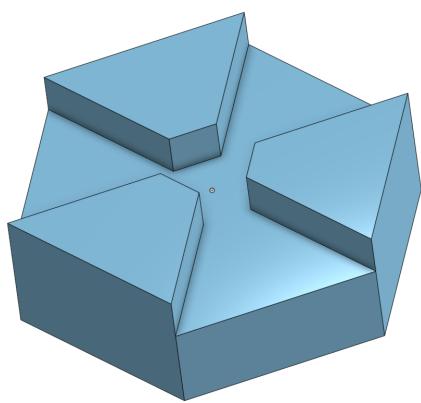
A dock design that uses this scaling method is shown in Figure 2.4.



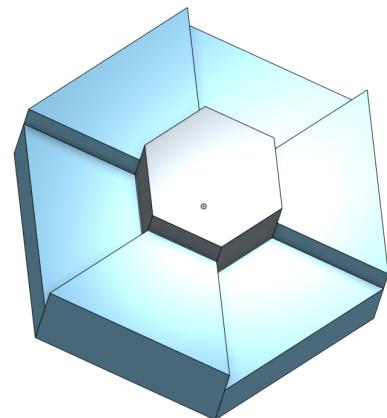
(a) Top view



(b) Corner View

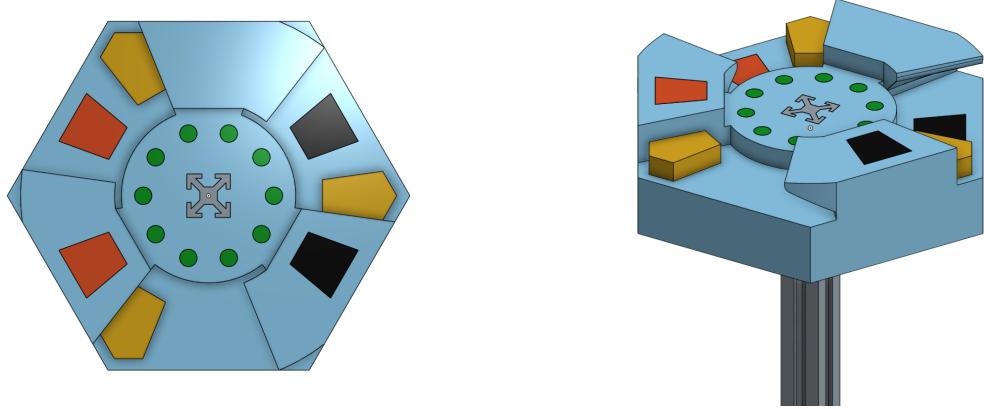


(c) Larger Interlocks



(d) Mated

Figure 2.3: Radially-symmetric interlocks



(a) Top view

(b) Corner View

Figure 2.4: Possible dock design

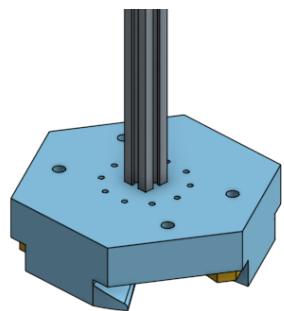
The body of the dock, containing three radially symmetric raised interlocks, is shown in blue. The central communication connections are green, while the power connections in the outer ring are red and black. The yellow blocks are spring-loaded and magnetically actuated to provide a latching mechanism. The gray rod extending from the back of the dock represents the place where the dock is mounted to the module. The yellow latching blocks and red / black power connections scale with the size of the outer dock, but always lie on the same radial slices. This ensures that the connectors still line up when the docks are scaled.

The docking procedure for this design is as follows. First the two docks are brought close to one another (Figures 2.5(a) and 2.5(b)) with the power pads properly aligned. The docks are then pushed together, compressing the spring-loaded yellow latches (Figure 2.5(c)). Finally, the docks are twisted relative to one another until the blue interlocks are fully engaged (Figure 2.5(d)).

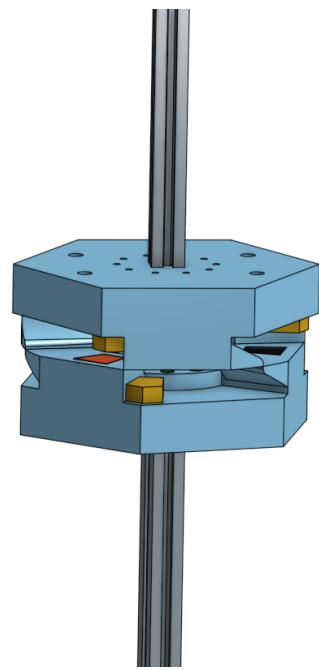
At this point the latches spring into place, preventing the docks from twisting back and releasing. Since the yellow latches passively remain in place, it takes no energy for the docks to remain connected. Because the latches move orthogonally to the rotation that would disengage the interlocks, the docking connection is also much stronger than many other dock designs; to force the docks apart you would need to apply enough force to cause all three latches to mechanically fail.

To release the dock, the yellow latches first be disengaged. This is nominally accomplished via magnetic actuation. Three electromagnets in each dock pull the latches into the dock, allowing the docks to rotate relative to one another again. Note that the power connections don't disengage until the docks are starting to pull apart, at which point the latches must already be fully disengaged. If a module's electromagnets are malfunctioning, the docks can still be disengaged by simply depressing the latches manually. Once the latches are disengaged, the docks can be twisted apart with the reverse of the docking procedure. Due to the spiraling on the interlocks, this final step can also be accomplished by gravity; If the dock to be released is pointed down, gravity will cause the dock to rotate and disengage without any further intervention.

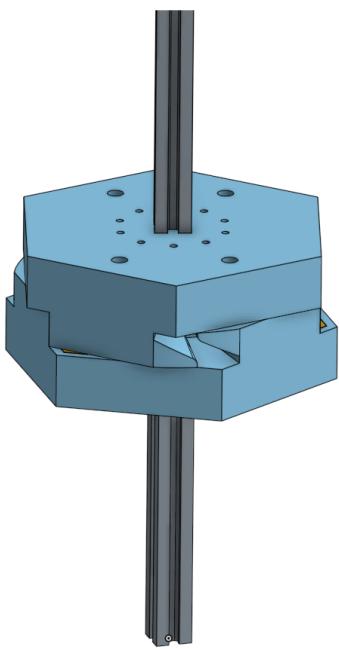
Automated docking requires the consent of the both modules, while undocking requires only one module. This asymmetry provides resilience against module failure; It ensures that docks can be released even if one of the modules dies and prevents docking if one of the modules is non-functional. This docking scheme can be accomplished with circuitry attached to GPIO pins



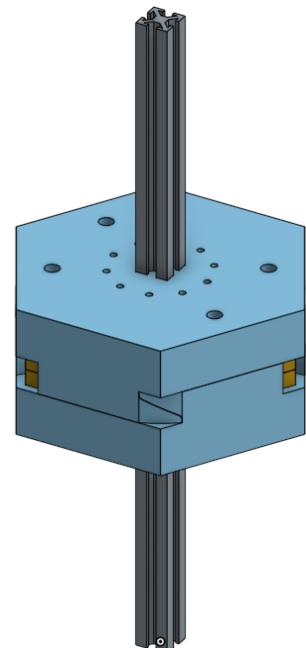
(a) Two docks



(b) Aligned



(c) Compressing springs



(d) Mated

Figure 2.5: Docking procedure

on each module's microcontroller. The circuitry should implement the logic in Table 2.1, in which H (high, indicating the desire to dock), L (low, indicating the desire to undock), and Z (high-z, indicating a powered-off / non-functional microcontroller) represent the GPIO pin states on each module. If either module drives the pin low, the latches are drawn into the docks to undock (U) the modules. In all other cases, the latches are allowed to spring up, allowing the modules to dock (D). This prevents the docking state from changing while the system is powering up; Due to timing differences, one module might be powered on and sending dock signals, while another might still be turning on, and hence still exhibits high-z.

The docking signal should be high by default to make it easy to add new modules. The signal should be driven low only when a disconnection is necessary, indicated either by a user, or through the automatic module disconnection procedure in Section 2.4.2.

	H	L	Z
H	D	U	D
L	U	U	U
Z	D	U	D

Table 2.1: Docking Logic Table

Modules communicate via two physical interfaces: A global network common to all modules in the system, and local point-to-point connecting neighboring modules.

The global network is primarily used for data that needs to be transferred

between the master and individual modules. This includes streaming data, such as commands and sensor readings, as well as module configuration / connection events. The network mirrors the physical architecture by using a tree topology, so the maximum topological depth is important; Limits on topology depth limit the number of modules that can be connected in series. Ethernet-based communication is thus a good choice for the global network, since Ethernet supports virtually unlimited depth and number of nodes (neglecting congestion considerations that would affect any tree network). Other protocols like USB are limited to topology depth of five, while CAN and RS485 support at most 30 and 32 nodes respectively.

The local point-to-point (PtP) connections are used for module localization (i.e., determining where in the structure a module has been attached), heartbeat signals, and recursive distributed computations. UART/USART are well suited to the local connections, since they provide fast, hermaphroditic, low-latency communication for the local links. Ethernet is another good choice, but is probably overkill since the connections are only PtP. I2C and SPI are also potential choices, but are non-hermaphroditic, which removes the benefits of using hermaphroditic docking mechanisms.

Module connectivity detection is performed via the local links. A newly connected module communicates with the module to which it's docked in order to determine where in the existing structure it has been attached. The module's model is then transmitted to the master coordinator via the global network. The master integrates the new module's model with the rest to produce the

full robot module. Commands and sensor data can then be sent over the global network, with recursive algorithms using the local link. A heartbeat sent on the local link allows for the detection of disconnected and malfunctioning robot modules, allowing robot health monitoring.

### 2.3 Self-Identity

Each module in an additive robot system has a self-identity (SI), an innate understanding of itself, its purpose, its relationship with other modules, and its health. There are four components of self-identity: functionality primitive, physical models, relationships, and status. A module's functionality primitive indicates the basic functionality it provides to the full system. Physical models describe the module's appearance, limits, and dynamics.

Relationships describe how the module interacts with other modules. Status details the internal state of the module, including joint positions, temperatures, and other sensor readings. The following sections detail each of these components.

Using each module's SI, the master module can automatically construct the full system's SI. The components of the system self-identity are the same, but the scope is expanded to include the whole robot. Functionality primitives

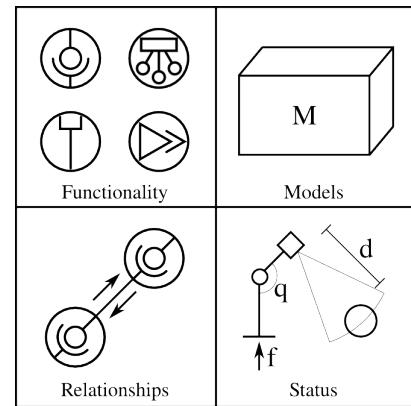


Figure 2.6: Four components of self-identity

list everything the robot can do. The physical model describes the entire robot. Relationships indicate how more modules can be added to expand the robot's functionality. Status describes the health of the robot as a whole, as well as progress towards the high-level tasks the robot is pursuing.

The system SI is used by the master module to present the robot's available functionality to the user, and to tailor whole-body planners and controllers to match the physical construction of the robot (see Chapter 4). Constructing the system SI in this manner ensures that the high-level software is always aligned with the physical reality, even when the structure of the robot is altered at run-time by damage or intentional rearrangement.

An additive robot system thus inherently understands what it can do, what to do when something breaks, and how it can be modified to do more.

### 2.3.1 Functionality Primitives

The modules comprising an additive robot system can be classified by their Functionality Primitive (FP), a key component of module SI. Each FP represents a general class of functionality that a module provides to the system. For example, the revolute FP provides the ability to rotate two docks relative to one another, while the thruster FP provides the ability to apply a force to the

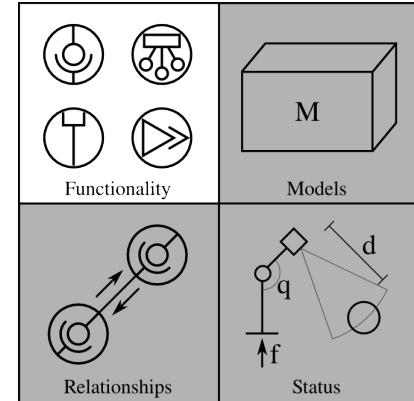


Figure 2.7: Functionality primitives

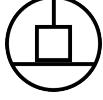
robot in the module’s frame. A full list of FPs can be found in Table 2.2.

A symbolic representation of a robot’s structure and functionality can be constructed by drawing each module’s FP symbol and connecting docked modules with lines. These compact functionality primitive representations allow for most of the robot’s SI to be communicated in a single image. Figures 2.3.1, 2.3.1, 2.3.1, 2.3.1, and 2.3.1 provide examples of existing robots broken down into their FP representations.

FPs are critical for synthesizing motions in additive robot systems. They allow the master coordinator and other coordinator modules to understand how to use each part of the robot’s body. For example, a FP-aware mobile manipulator robot can determine that it should coordinate the motion of its wheel-FP modules to move towards an object to be manipulated, and then use its revolute-FP modules to move a gripper-FP module to a place where it can grasp the object. The whole-body control synthesis process is described in depth in Chapter 4.

Symbol	Primitive	Description
	Interlink	An infrastructure module used to provide structure and connect modules together. Especially useful for adapters and branch points.
	Coordinator	An infrastructure module that coordinates other sensor and actuator modules. Typically contains a computer, microcontroller, or FPGA. One coordinator module should be designated as the master.
	Energy Source	An infrastructure module that either stores or generates the energy used by the robot. This includes batteries, fuel tanks, electrical generators, and pressure generators.
	Communication	An infrastructure module used to communicate with the outside world. May use wifi, sound, or any other method of communication.
	Storage	An infrastructure module with empty space that can be used for storage while the robot is in the field. This includes modules that are used to change the robot's density, like ballast tanks and hot air balloons.
	Revolute Joint	An actuator module with one or more rotational degrees of freedom. May be passive, rigid, or series elastic.
	Prismatic Joint	An actuator module with one or more translational degrees of freedom. May be passive, rigid, or series elastic.
	Linkage Joint	An actuator module defined by a linkage. May be passive, rigid, or series elastic.

(Table 2.2 continued on next page)

	Thruster	An actuator module that provides body-frame force along one or more axes. This includes propellers, jets, and rockets.
	Flywheel	An actuator module that provides body-frame torque along one or more axes. Includes reaction wheels and eccentric vibration modules.
	Wheel	An actuator module providing an exposed wheel intended for locomotion. This includes tires, omni wheels, and mechanum wheels.
	Contact Surface	A surface module providing a sturdy, sensorized contact surface intended for locomotion. The contact surface may or may not be flat.
	Fluid Surface	A surface module providing a surface designed to interact with fluids. This includes wings, fins, sails, parachutes, and helicopter blades.
	Gripper	A end effector module providing the ability to grasp objects. This includes pinching grippers, multi-fingered hands, and jamming grippers, as well as grippers that use chemical, pressure, and electrostatic-based adhesion.
	Tool	An end effector module providing a specialized tool. Includes drills, shovels, grappling hooks, sensors, and other specialized equipment.
	Social	A module primarily used for interfacing to humans. Includes faces, speakers, and screens.

(Table 2.2 continued on next page)

	Directed sensor	A sensor module that must be carefully positioned w.r.t. the object being measured. This includes line-of-sight and contact-based sensors like LiDARs, Velodynes, RGBD, IR, ultrasonic, cameras in various spectra, pH, and touch sensors.
	Ambient sensor	A sensor module that measures ambient conditions. This includes vibration, light, gas / air quality, pressure, temperature, and sound sensors.
	Local. sensor	A sensor module that provides localization data. This includes imus, compasses, beacon detectors, and gps.

Table 2.2: Functionality Primitives

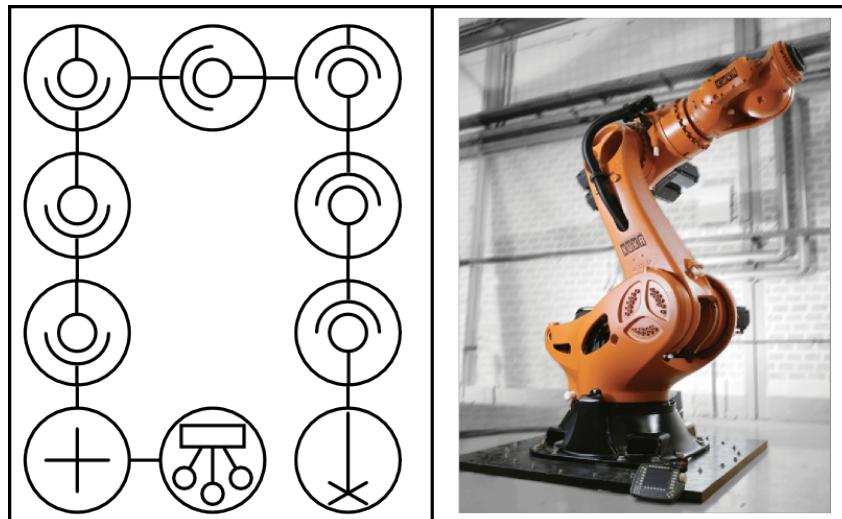


Figure 2.8: Industrial Robot Functionality Tree

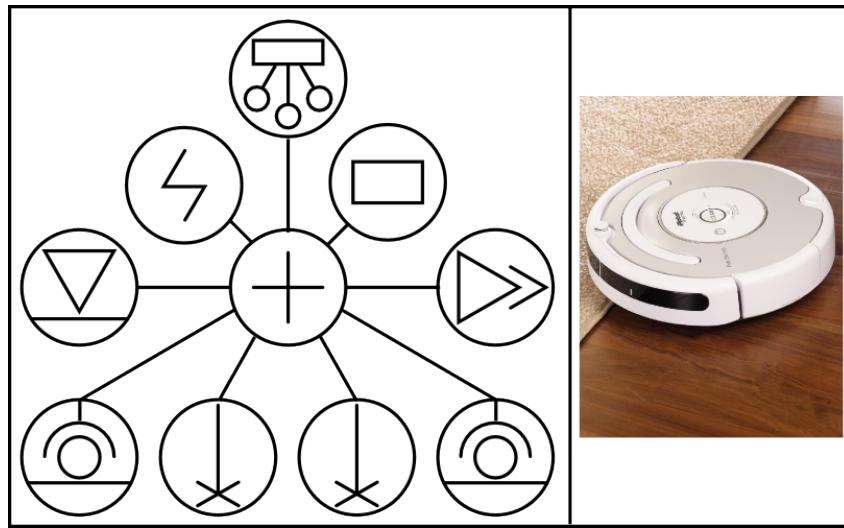


Figure 2.9: Cleaning Robot Functionality Tree

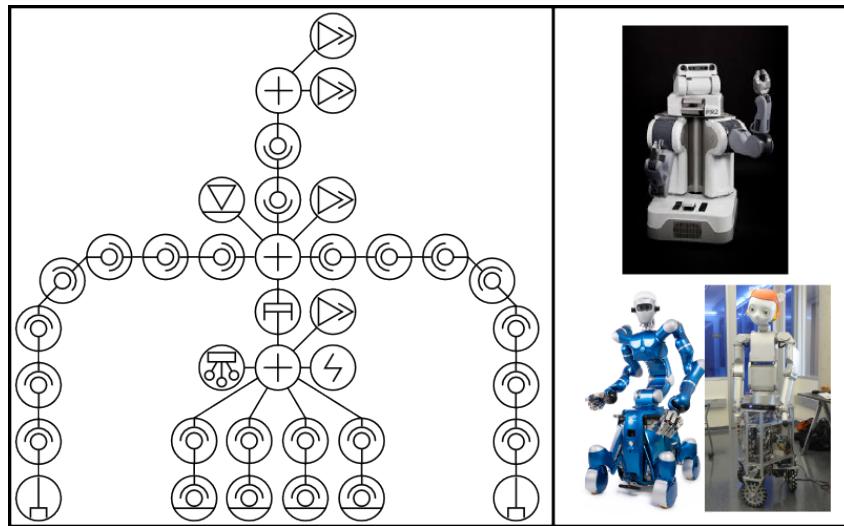


Figure 2.10: Mobile Manipulator Functionality Tree

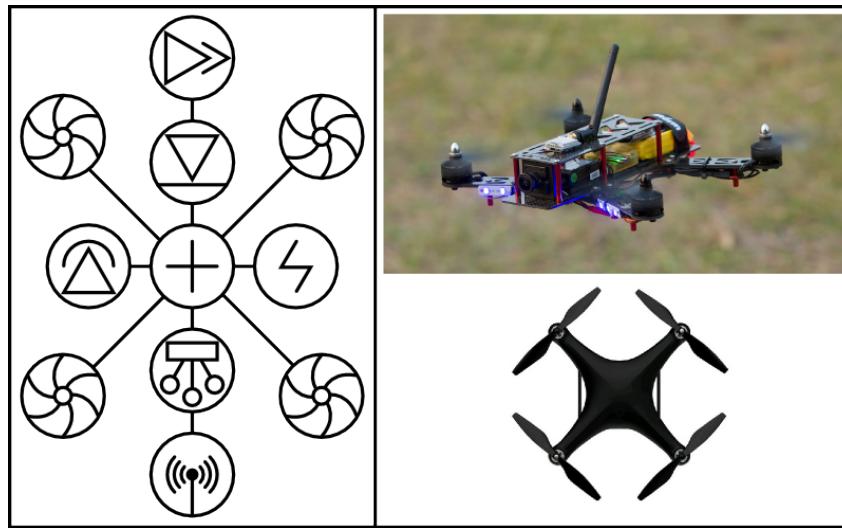


Figure 2.11: Quadcopter Functionality Tree

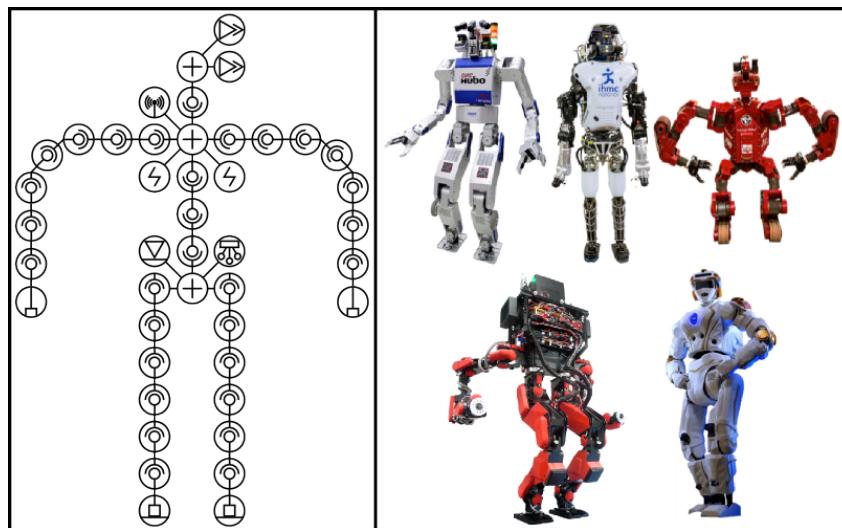


Figure 2.12: Humanoid Robot Functionality Tree

### 2.3.2 Physical Model

A module can be modeled as a tree in which each node is a “frame.” Each frame node represents a frame of reference relevant to the robot’s operation. Every frame, other than the root, has exactly one parent frame, denoted  $pj$ . A frame is related to its parent via an affine transform  ${}^jT_{pj} \in SE(3)$ . For most frames, this transform is always equal to its static component,  ${}_i^jT_{pj}^s$ . Joint frames also have a dynamic component  ${}_jT_{j_i n}^d$  derived from the joint axis and states, with  ${}^jT_{pj} = {}^jT_{j_i n}^d {}_i^jT_{pj}^s$ . The dynamic component is a constant identity matrix for non-joint frames. Frames can have any number of children,  $c \in C$ . In addition to defining a virtual frame of reference, a frame can also contain a physical entity, such as a link, actuator, closure, sensor, dock, or interaction point.

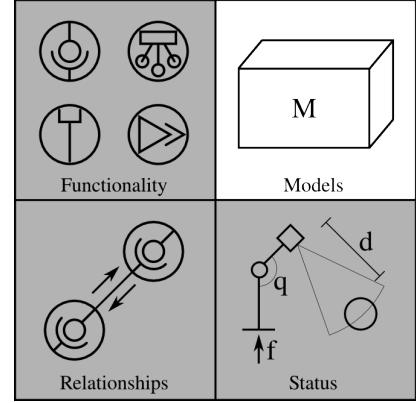


Figure 2.13: Physical models

Link frames (frame nodes that contain a link entity) describe the rigid bodies making up the module. Links have mass  $m \in R$ , a center of mass  $c \in R^3$ , inertia at the center of mass  $I^{com} \in R^{3 \times 3}$ . Detailed link geometry is expressed as one or more triangle meshes  $G \in Gs$ , with bounding boxes  $B \in Bs$  represented by geometric primitives. This geometry is related to the frame origin by a set of transforms  ${}^G T_j^s$  and  ${}^B T_j^s$ . Links associated with fluid surfaces also have an fluid dynamic force function  $f_j(\bar{v}, \rho, x)$  that describes the lift and drag on the link due to movement in a fluid.

Actuator frames exert wrenches (forces and torques) on their parent frame. If an actuator has a child frame, it is considered a joint actuator frame. Joint actuators exert an equal and opposite wrench on the child frame and constrain the motion of the two frames. Actuators have an unit-length axis ( $\bar{S} \in R^6$ , parent frame) about which they apply wrenches and / or constrain motion, and an actuator model,  $a(x, u)$ . The actuator model may be linear or nonlinear and maps the control input  $u$  and the actuator state  $x$  to the magnitude of the wrench applied to the parent frame  $w = \bar{S}a(x, u)$ . The model itself varies widely based on the type of actuator and application. The naive torque-source model would simply be  $a(x, u) = u$ . An electric motor model might be a nonlinear function of the motor's rotor inertia, torque constant, max temp, heat dissipation, winding resistance, friction, and position / velocity / acceleration / control input limits. The only restriction on  $a$  is that it must be invertible. That is, there must exist an actuator command that will produce any wrench within a valid range of wrenches.

Sensor frames describe the sensors attached to the module. This frame defines the pose of the sensor in the module, and is used as a control point for directed sensors. The sensor frame definition also includes the type of sensor as well as any model parameters.

Dock frames describe the module's docking interfaces. The z axis of a dock frame must always be normal to the docking plane, facing out of the module. The dock frame also specifies the scale of the dock. Dock frames must be either a leaf or the root of the tree.

Interaction frames specify the control point(s) of a module that should be used for intentional environment interaction. This includes things like the sole of a foot, the center of a gripper module, or the tip of a drill. The interaction frame is associated with the geometry of its parent—if an interaction frame is defined for the sole of the foot, it should be a direct child of the foot link. The interaction frame also defines the tool’s goal space.

### 2.3.3 Relationships and Status

The module’s relationships describe its external state—the module’s interaction with other modules. It identifies the neighboring modules attached to each of the module’s docks, as well as the state of each docking interface.

The module’s status describes its internal state. This includes the current value of all sensor readings along with estimated state and control variables defined by the model.

Unlike the static functionality primitive and physical models, the relationship and status components of the module’s SI can change at any time. This data must remain synchronized with the master module to ensure correct control synthesis. Relationships change relatively infrequently, so its portion of the SI is simply sent back to the master whenever it changes. Status changes

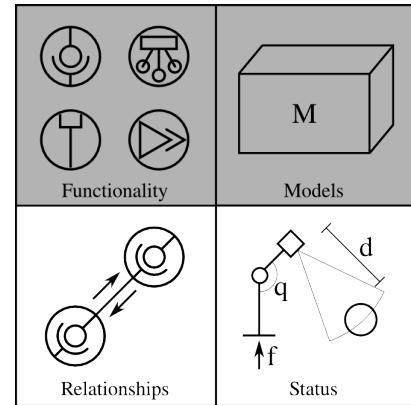


Figure 2.14: Relationships and status

constantly, so the data is continuously streamed to the master module. The status data overhead is minimized by using the SI to establish a fixed, but module-specific, format for the streaming data. This allows state / control information to be generically transferred as a large data glob, without wasting bandwidth on data that only describes its contents (e.g., protocol header, or data that simply labels joint position data with the name of the joint).

#### 2.3.4 Static Self-Identity Specification

The static portion of a module's SI can be described in software using XML. The top-level tag contains the module's type and primary functionality primitive as attributes. The physical model, along with additional functionality primitives as necessary, is contained within this tag. Every model component (links, joints, docks, etc.) has attributes specifying the component's parent, as well as the transform relating the component to its parent. This uniform hierarchy specification allows for more straightforward construction than URDF, in which only joints specify the relationship between parent and child components.

The SI specification for the example roll joint module shown in figure 2.15 follows. In this example, the module has two links ( $l0, l1$ ) connected by a revolute joint ( $j0$ ) and a dock on each link ( $d0, d1$ ).

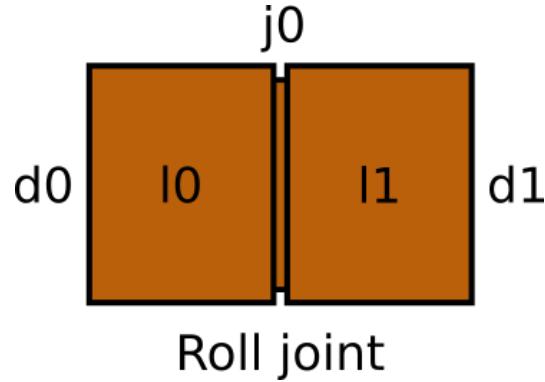


Figure 2.15: Example module

```

<module type="roll_module" primary_functionality="revolute_joint">
  <link name="l0" parent="">
    <geometric>
      <cyylinder radius="0.025" length="0.1" origin="0_0_0_0_0_0.05"/>
      <cyylinder radius="0.05" length="0.1" origin="0_0_0_0_0_0.1"/>
    </geometric>
    <inertial>
      <inertia mass="0.5" ixx="0" iyy="0" izz="0" ixy="0" ixz="0" iyz="0"
        origin="0_0_0_0_0_0.05"/>
      <inertia mass="2" ixx="0" iyy="0" izz="2.5e-3" ixy="0" ixz="0" iyz="0"
        origin="0_0_0_0_0_0.1"/>
    </inertial>
  </link>

  <joint name="j0" parent="l0" origin="0_0_0_0_0_0.1" functionality="revolute_joint">
    <state name="q" min="-2.36" max="2.36"/>
    <state name="q_dot" max="1"/>
    <control name="u" max="100"/>
    <electric_motor axis="0_0_-1_0_0_0" torque_constant="1"
      transmission="1" damping="0.3" rotor_inertia="0.01" input="u"/>
  </joint>

  <link name="l1" parent="j0" origin="0_0_0_0_0_0">
    <geometric>
      <cyylinder radius="0.025" length="0.1" origin="0_0_0_0_0_0.05"/>
    </geometric>
    <inertial>
      <inertia mass="0.5" ixx="0" iyy="0" izz="0" ixy="0" ixz="0" iyz="0"
        origin="0_0_0_0_0_0.05"/>
    </inertial>
  </link>

  <dock name="d0" parent="l0" origin="3.14_0_0_0_0_0"/>
  <dock name="d1" parent="l1" origin="0_0_0_0_0_0.1"/>
</module>

```

## 2.4 Architectural Algorithms

### 2.4.1 Module Connection

If a module is physically docked, it has access to the power / communication channels and is given an IP address by the master. However, it does not become a part of the additive robot system until the master receives its SI and validates its relationships. This means that when the robot first starts up, and when new modules are added, inter-module communication is necessary to complete the construction of the system.

Shortly after being assigned an IP address, each module sends its SI to the master. The SI initially contains no relationships, so the master just stores the SI for later use. To establish relationships, every module sends out DockID messages at a fixed rate through all unconnected local links. A DockID message contains the sender's IP address and the name of the dock through which it's being sent.

When a module receives a DockID message, it updates its internal relationships and sends a Relationship message to the master via the global network if there's been a change. A Relationship message contains the sender's dock, the DockID reported by the connected module, the status of the docking request (initially "pending") and optionally a key-based digital signature (see Section 2.4.4 for more on security). Once the master receives authentic Relationships from both modules, it stores them until one of the modules is integrated into the system. This allows the master to determine which module should be considered the parent, and ensures that the modules are added in a

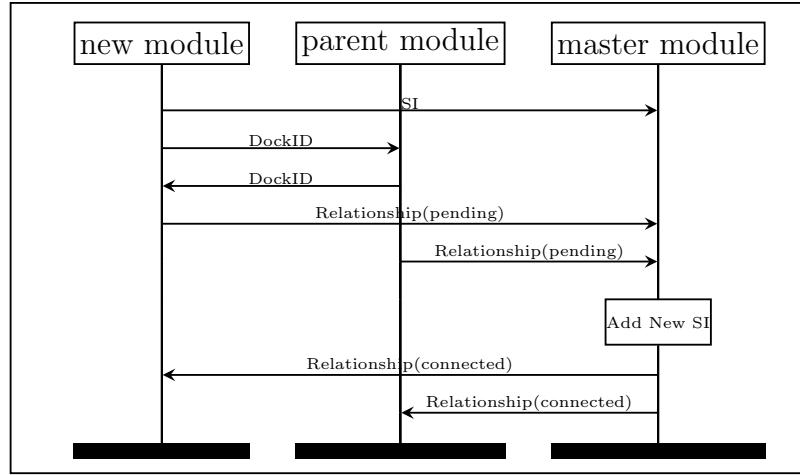


Figure 2.16: Module Connection MSC Diagram

tree structure, with the master as the root. Since the master is initially the only part of the system, the modules connected directly to the master are added first, then the ones connected to those modules, and so on. When each module is ready to be added to the system, the master adds its SI to the system SI (see Section 2.4.3 for details). It then sends Relationship messages back to the relevant modules, with status “connected,” to confirm the connection.

Figure 2.16 provides an example of a module being connected to an additive robot system. In the example, the “parent module” is already part of the system, and the “new module” has just been physically docked with the parent module.

#### 2.4.2 Module Failure Detection

Each module sends its neighbors empty Heartbeat messages on local links at a fixed rate. If a module has been unexpectedly detached or is broken electrically, it will no longer be able to send these messages. Thus, they give the system the ability to detect module failure. When module notices that its neighbor's heart has stopped beating, it sends a Relationship message to the master identifying the offending dock, with status "lost." The master moves the SIs for the lost module and all its children from the system SI into a separate tree. The SIs are separated rather than deleted to better inform a human operator about which modules most likely failed. The master then sends the Relationship back to the module that detected the failure, with status "disconnected" to officially disconnect the module. At this point, the module can optionally drive the docking signal low, releasing the dead module(s) from the system. However, since releasing the modules may cause more problems than keeping them (e.g., a broken leg is often better than no leg), a human operator should make the decision to release the modules in most cases. Figure 2.17 shows this process in a scenario where parent module is no longer receiving lost module's Heartbeat.

#### 2.4.3 System Self-Identity Generation

The system SI is incrementally constructed by the master module, starting with its own module SI. Mirroring the physical structure, the system SI is stored as two trees: one at the module level containing the FP and status

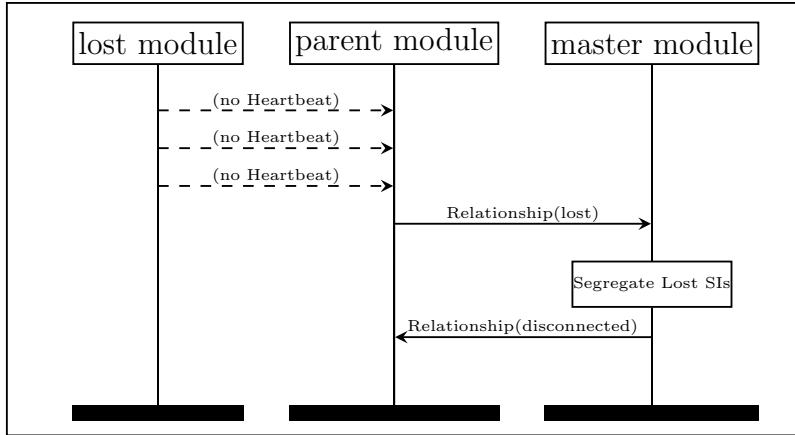


Figure 2.17: Module Removal MSC Diagram

information, the other at the frame level describing the system's physical model.

The module connection procedure in Section 2.4.1 ensures that each module's SI is added as a leaf, regardless of communication timing. Thus, to construct a system SI, the master simply repeats the procedure for adding a single leaf module until all modules in the system have been processed.

Adding a single leaf module to the FP/status tree is straightforward. The module's relationships specify one module already in the tree (parent) and the new module to be added. The new module's functionality primitive is thus simply added to the tree as a child of the module to which it is docked. The state variables and control inputs are similarly mapped to the new node in the tree.

Adding to the physical model may be more complicated, depending on which dock in the new module was connected. The dock in the existing physical tree, as well as the dock in the new module, are specified by the module's

relationship information. If the new module’s dock was the root of its physical model, the new module’s frames can be appended to the system model by simply making the existing dock the parent of the new module dock, with a fixed 180 degree rotation on the x axis but no translation. The z-axis of dock frames always points out of the module, so this rotation mates the modules in the correct orientation, with their z-axes pointing into one another.

If the new module’s dock is not the root, the module’s physical model must be modified before it can be added. Depending on the position of the dock in the tree, it may require “flipping” frame connections (changing a parent into a child and vice versa) throughout the tree. Figure 2.18 illustrates the flipping that occurs when the example module in section 2.3.4 is connected at each of its docks. Each link and its parent joint / dock are grouped in the diagram for brevity. Sensor, dock, and interaction frames can be flipped by simply inverting their parent transform. Link and actuator frames, however, require special consideration; the center of mass, inertia matrix, and joint / actuator axes may need to be updated to account for the change in parent frame. The process for flipping these frames can be found in Featherstone’s Rigid Body Dynamics book [14].

#### 2.4.4 Security

The modules allowed to connect to an additive robot system can optionally be restricted via keys. To use high-security mode, each module must create a public/private key pair. Prior to operation, the public key for each

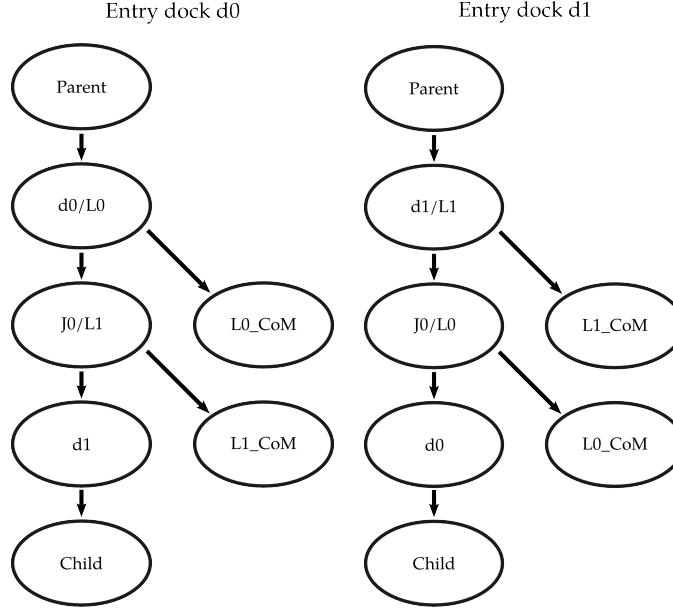


Figure 2.18: Flipping Due To Dock Change

module in the system is stored by the master module. When a module sends a Relationship message to the master in an attempt to connect to the system, it sends an encrypted message containing the current time. If the master can't decrypt the message, or the time contained in the message is old / has been received before, the connection attempt is ignored.

High-security mode thus allows a user to whitelist specific module hardware. This ensures that un-vetted hardware is mostly ignored by the system, limiting the data it can acquire and the damage it can do to the system. However, it also limits the ability to quickly repair or extend the robot, since new modules must be registered with the master. This feature is best used when security is much more important than fast reconfiguration.

# Chapter 3

## Module Decoupling

### 3.1 Overview

The module decoupling layer provides a decoupled linear interface to the underlying coupled non-linear dynamics of the additive robot system. It takes as input the desired joint accelerations  $\ddot{q}$  and external forces to apply to the environment  $f_{ext}$ . It uses the system SI to produce actuator-specific commands  $u$  for every actuator module in the system. The module controllers then apply the command to the actuator hardware in a manner specific to each module.

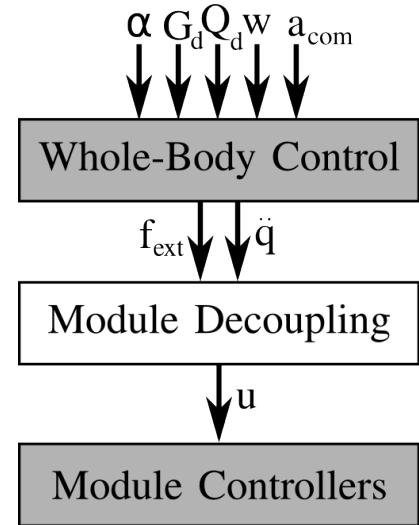
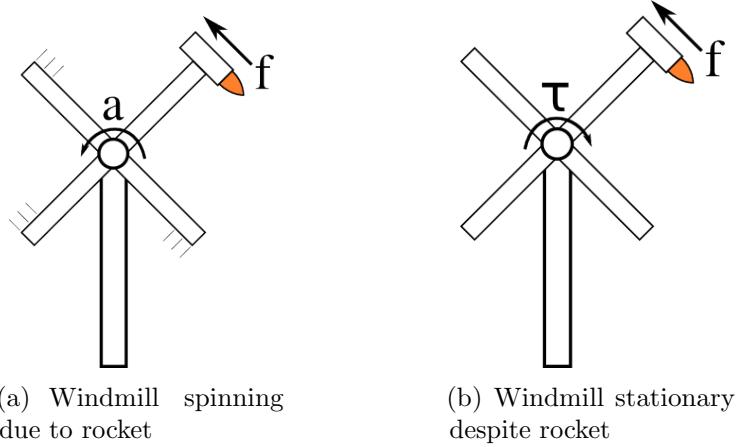


Figure 3.1: Module decoupling layer

The module decoupling layer is useful because forces exerted by one actuator will affect the behavior of other actuators in the system. For example, a rocket attached to an arm of a windmill will cause the windmill to spin, even if the windmill joint is applying no torque (Figure 3.2(a)). The modules are thus “coupled” to one another. The module



decoupling layer decouples the modules by feedback-linearizing the system—essentially computing the force an actuator needs to apply to perfectly balance the dynamic effects of the other modules. This means that the windmill joint would apply a torque exactly equal and opposite to the torque from the rocket, keeping the windmill still if no acceleration was intended (Figure 3.2(b)). The module decoupling layer is useful because it allows a higher-level controller to command each module’s actuator(s) without worrying about how commands sent to one module will affect the dynamics of other modules.

There are two types of inputs to the module decoupling layer: joint accelerations and spatial forces.

Joint accelerations ( $\ddot{q}_{i,d} \in R$ ) can be used to control each joint as if it were an isolated Newtonian system. For example, a simple joint trajectory controller might use the modular decoupling layer in the following way:

$$x_i = [q_i, \dot{q}_i]^T$$

$$\begin{aligned}\dot{x}_i &= [\dot{q}_i, \ddot{q}_{i,d}]^T \\ \ddot{q}_{i,d} &= K_i^T(x_{i,d} - x_i), \quad K_i \in R^2\end{aligned}$$

Spatial forces ( $f_d \in R^6$ ) describe the forces and torques that each thruster, flywheel, and contact / fluid surface should apply to the environment. For thruster and flywheel actuators, the spatial force to be applied should be explicitly specified. For surfaces whose force response is highly dependent on environmental conditions, the spatial forces can either be predicted / desired forces (e.g., apply half the robot's weight downward through each of its two feet), or sensed forces (balance the force the robot currently feels). If the robot's center of mass is stationary, these forces should add up to the total gravitational force on the robot. When the robot's mass is accelerating, the difference from the stationary case should be equal to the force required to cause the composite inertia to accelerate ( $\bar{f} = I^C \bar{a}^C$ ).

The algorithms presented for module dynamics decoupling are based on recursion over the system's physical model tree (see Section 2.3.2). A recursive approach is well-suited to additive robot systems because it is particularly amenable to structural modifications and modularly-distributed computation. For systems with many actuators, the recursive formulation is also faster than those based on full system matrix multiplication:  $O(N)$  vs  $O(N^3)$ .

The module decoupling scheme makes the following assumptions:

1. All links are sufficiently rigid that the robot can be modeled as an

articulated rigid body.

2. Each joint actuator can measure its own joint states  $(q, \dot{q})$ .
3. The gravity vector  $g$  is known or measurable.

## 3.2 Model Compression

Minimizing latency is critical for feedback control, and computational power is often limited on embedded systems. The algorithms below recurse over the physical model tree and the computation time scales with the number of frames in the tree. It is thus advantageous to remove any extraneous frames prior to performing the computations. For example, sensor frames and dock frames don't significantly affect the dynamics of the system, and links that are connected by fixed transforms (e.g., the links on either side of a dock) can be merged without any information loss. To that end, Algorithm 1 recursively compresses the raw SI model into a compact joint-centric tree. Though the algorithm reorganizes the tree and combines redundant frames' inertial properties, all original frames remain accessible; The frames are preserved as children of the closest parent node. Each node in the tree represents a single mass attached to a parent joint, and contains the following information (where  $j$  identifies the node):

1.  $cjs$ , a set of pointers to the node's children.
2.  $pj$ , a pointer to the node's parent.

3.  ${}^j T_{pj}^s$ , the pre-joint static pose transform relating the node to its parent.
4.  ${}^j X_{pj}^s$ , the pre-joint static motion transform relating the node to its parent.
5.  $\bar{I}_j$ , the spatial inertia of the node, expressed in the node's frame.
6.  $\bar{S}_j$ , the axis of the joint connecting the node to its parent, expressed in the parent's frame.

---

**Algorithm 1** System Model Compression

---

```

1: procedure COMPRESS( $j$ )
2:    $\bar{I}_j = \begin{bmatrix} \bar{I}_j^{com} - m_j(r_j \times)(r_j \times) & m_j(r_j \times) \\ -m_j(r_j \times) & m_j(1_{3 \times 3}) \end{bmatrix}$ 
3:   for child  $cj \in cjs$  do
4:     COMPRESS( $cj$ )
5:   end for
6:   if  $j$  is not a joint then
7:      $\bar{I}_{pj} = \bar{I}_{pj} + {}^j X_{pj}^{sT} \bar{I}_j {}^j X_{pj}^s$  ▷ Merge self into parent
8:      $\bar{I}_j = \bar{0}_6$ 
9:     for child  $cj \in cjs$  do ▷ Transfer children to parent
10:       ${}^c T_{pc}^s = {}^c T_{pc}^s {}^j T_{pj}^s$ 
11:       ${}^c X_{pc}^s = {}^c X_{pc}^s {}^j X_{pj}^s$ 
12:       $\bar{S}_c[0 : 2] = {}^p j T_j^s[0 : 2, 0 : 2] \bar{S}_c[0 : 2]$ 
13:       $\bar{S}_c[3 : 5] = {}^p j T_j^s[0 : 2, 0 : 2] \bar{S}_c[3 : 5]$ 
14:      Add  $c$  to parent's child list
15:      Remove  $c$  from  $j$ 's child list
16:   end for
17:   end if
18: end procedure

```

---

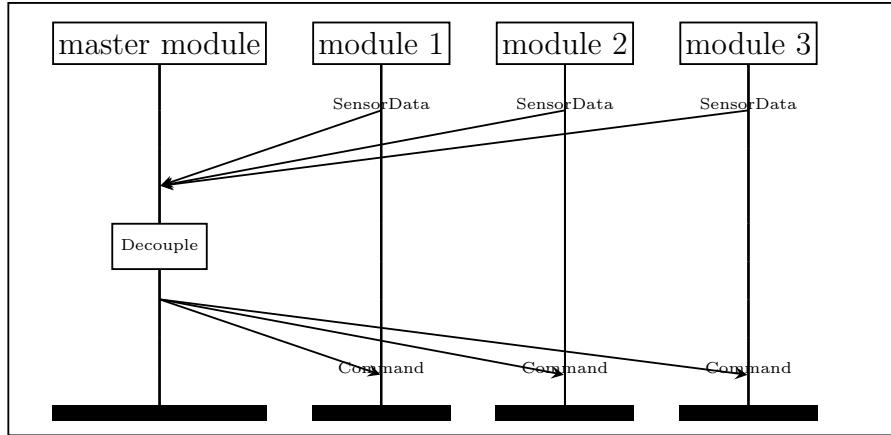


Figure 3.2: Centralized Module Decoupling MSC Diagram

### 3.3 Centralized Module Decoupling

In the centralized module decoupling scheme, each module sends its sensor readings to the master. The master then computes decoupling commands for the whole system and sends them back to each module (Figure 3.2). This is the dominant approach in robotics today, with a centralized controller typically computing inverse dynamics via the Recursive Newton Euler Algorithm (RNEA) or similar. This algorithm is often implemented using Featherstone's spatial vector algebra [14].

Algorithm 2 is essentially the same as Featherstone's floating-base inverse dynamics algorithm, but has been reformulated to facilitate the transition to distributed computation. It uses the full robot tree, the current joint states, the desired joint accelerations, and the desired spatial forces to compute the forces and torques that must be applied at each joint. These torques are then converted into module commands via the  $a^{-1}$  function provided by the module

SI<sub>s</sub>. The algorithm does this in two sequential recursions. The first recursion starts from the root and recurses outward calculating kinematics, then inwards computing the forces and composite inertias of each joint. The second recursion starts from the root again and calculates the adjustment needed to account for the acceleration of the root link, in addition to computing the final torques. For a detailed explanation on how the algorithm works, see Featherstone's Rigid Body Dynamics Algorithms book [14].

The presented algorithm differs from Featherstone's original floating-base inverse dynamics in the following ways:

1. It uses a recursive formulation rather than looping over numbered bodies.
2. It makes the assumption that  $\dot{S}_j = 0$  (i.e., the joint axis doesn't vary with time).
3. It performs gravity compensation based on external forces, rather than uniform acceleration.

### 3.4 Distributed Module Decoupling

In the distributed module decoupling scheme, each module still sends its sensor readings to the master, but the master no longer computes / sends the decoupling commands. Instead, the master sends only the desired joint accelerations and spatial forces to the relevant modules.

---

**Algorithm 2** Centralized Module Decoupling

---

```

1: procedure FIRSTPASS( $j$ )
2:   if  $j$  is root then ▷ Initialize
3:      $\bar{a}_0^r = \bar{0}$ 
4:   end if
5:    $I^c \leftarrow I_j$ 
6:    $\bar{S}^R = \bar{S}_j[0 : 2]$  ▷ Calculate kinematics outwards
7:    $\bar{S}^P = \bar{S}_j[3 : 5]$ 
8:    ${}^{j,in}E_j = \cos(q_j)I_{3x3} + \sin(q_j)(\bar{S}^R \times) + (1 - \cos(q_j))(\bar{S}^R \otimes \bar{S}^R)$ 
9:    ${}^{j,in}r_j = \bar{S}^P q_j$ 
10:   ${}^jT_{j,in}^d = \begin{bmatrix} {}^{j,in}E_j & {}^{j,in}r_j \\ 0_{1x3} & 1 \end{bmatrix}^{-1}$ 
11:   ${}^jT_{pj} = {}^jT_{j,in}^d {}^{j,in}T_{pj}^s$ 
12:   ${}^jE_{pj} = {}^jT_{pj}^d[0 : 2, 0 : 2]$ 
13:   ${}^{pj}r_j = -{}^jE_{pj}^{-1} {}^jT_{pj}^d[0 : 2, 3]$ 
14:   ${}^jX_{pj} = \begin{bmatrix} {}^jE_{pj} & 0 \\ -{}^jE_{pj}({}^{pj}r_j \times) & {}^jE_{pj} \end{bmatrix}$ 
15:   $\bar{v}_j = {}^jX_{pj} \bar{v}_{pj} + S_j \dot{q}_j$ 
16:   $\bar{a}_j^r = {}^jX_{pj} \bar{a}_{pj}^r + \bar{v}_j \times (\bar{S}_j \dot{q}_j) + S_j \ddot{q}_j$ 
17:  for child  $cj \in cjs$  do ▷ Recurse
18:    FIRSTPASS( $cj$ )
19:  end for
20:   $I_j^c = I_j + \sum_{cjs} {}^{cj}X_j^T I_{cj}^c {}^{cj}X_j$  ▷ Calculate dynamics inwards
21:   $\bar{p}_j = I_j \bar{a}_j^r - (\bar{v}_j \times)^T I_j \bar{v}_j - {}^j f^{ext} + \sum_{cj} {}^{cj}X_j^T \bar{p}_{cj}$  ▷  ${}^{cj}X_j$  = child's  ${}^jX_{pj}$ 
22: end procedure
23: procedure SECONDPASS( $j$ )
24:   if  $j$  is root then
25:      ${}^0\bar{a}_0 = -(I_0^c)^{-1} \bar{p}_0$  ▷ Accelerate the root node
26:   end if
27:    ${}^j\bar{a}_0 = {}^jX_{pj} {}^{pj}\bar{a}_0$  ▷ Reflect acceleration in other nodes
28:    $\tau_j = \bar{S}_j^T (I_j^c {}^j\bar{a}_0 + \bar{p}_j)$ 
29:   for child  $cj \in cjs$  do ▷ Recurse
30:     SECONDPASS( $cj$ )
31:   end for
32: end procedure

```

---

The distributed algorithm works by replacing the inter-module recursions in the centralized algorithm with data transfers. That is, each module computes its own part of the dynamics using the centralized algorithm, then sends dock-centric data to its neighbors to finish the calculation.

If the distributed computation is synchronized (i.e., the second recursion doesn't start until the first is complete), the net result is exactly the same math as in the centralized algorithm (Figure 3.3). However, only one command is produced for a full calculation. The calculation time is bounded by the time it takes for the algorithm to make three trips (outwards, inwards, outwards) down the longest chain in the tree. Since communication time between modules is non-zero, this slows down the control rate and often makes the algorithm uncompetitive with the centralized version.

The control rate can be significantly improved by not requiring synchronization (Figure 3.4). In this asynchronous scheme, each module computes the dynamics every time it receives new data from its neighbor. Since the data is usually not from the same computation, this approach does not perfectly reflect the centralized math. Asynchronous decoupling thus represents a “current best guess” at the decoupling commands, and comes at the cost of decreased decoupling accuracy. The accuracy loss may become significant if the control rate is slower than the dynamics of the system. The asynchronous approach is nevertheless adopted in this work.

Since the distributed computation is asynchronous, the algorithm can be modified into a single recursion (outwards + inwards). The final recursion

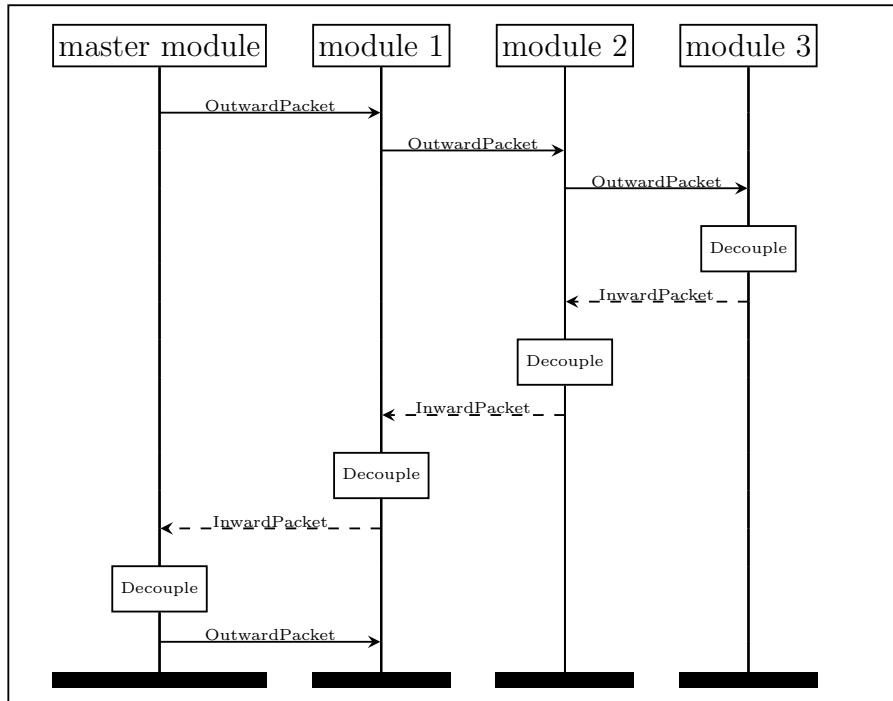


Figure 3.3: Synchronous Module Decoupling MSC Diagram

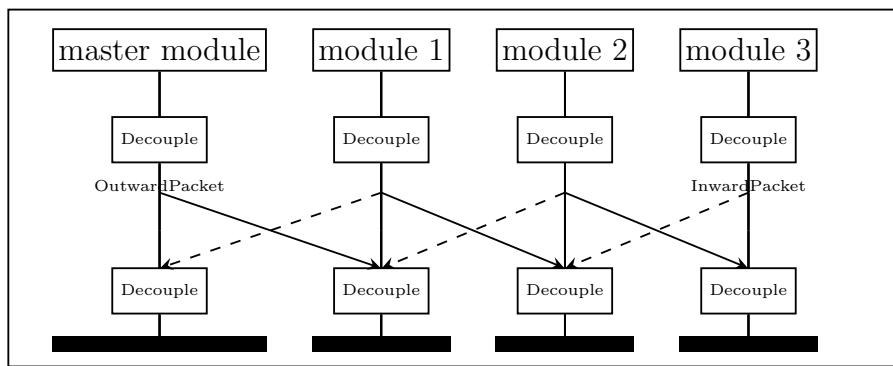


Figure 3.4: Asynchronous Module Decoupling MSC Diagram

in the centralized version accounts for the acceleration of the root node and computes the command. Replacing the root acceleration  $a_0^r$  in the first outward pass of the centralized algorithm with the true root node acceleration (if you knew it beforehand) results in the same math. The distributed version is asynchronous, so using the “current best guess” root acceleration in the first pass is not significantly more erroneous than performing the second outward pass. The command calculation can then be moved to the end of the inwards pass without loss, and the final recursion can be eliminated.

Two kinds of data packets are transferred between modules: one containing data for the outward recursion, and one containing data for the inward.

The OutwardPacket is received from each module’s parent, and contains 12 floats (48 bytes):

1.  $\bar{v}_{pj} \in R^6$ : spatial velocity at the parent dock
2.  $\bar{a}_{pj}^r \in R^6$ : spatial acceleration at the parent dock

The InwardPacket is sent to each module’s parent, and contains 16 floats (64 bytes):

1.  $\theta_{pj}^C \in R^{10}$ : parameterized composite inertia at the parent dock
2.  ${}^{pj} \bar{p}_j \in R^6$ : force at the parent dock

The parameterized inertia  $\theta = [I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}, mc_x, mc_y, mc_z, m]^T$  used in the InwardPacket saves bandwidth by sending the minimum information

needed to reconstruct the full inertia matrix,  $I$ . The inertia can be changed between these two forms as follows:

$$\theta = [I(0,0), I(1,1), I(2,2), I(0,1), I(0,2), I(1,2), I(2,4), I(0,5), I(1,3), I(5,5)]^T$$

$$I = \begin{bmatrix} \theta(0) & \theta(3) & \theta(4) & 0 & -\theta(8) & \theta(7) \\ \theta(3) & \theta(1) & \theta(5) & \theta(8) & 0 & -\theta(6) \\ \theta(4) & \theta(5) & \theta(2) & -\theta(7) & \theta(6) & 0 \\ 0 & \theta(8) & -\theta(7) & \theta(9) & 0 & 0 \\ -\theta(8) & 0 & \theta(6) & 0 & \theta(9) & 0 \\ \theta(7) & -\theta(6) & 0 & 0 & 0 & \theta(9) \end{bmatrix}$$

Algorithm 3 presents the distributed module decoupling algorithm. The Decouple function recursively computes the module's dynamics starting from the root of the module's model, integrating cached data from other modules as appropriate. The algorithm assumes that the root of the module model is the dock that connects it to its parent. The function should be called at a constant rate, ideally synchronized to local sensor data acquisition.

The worst-case command rate and latency depend on the speed of the communication channels and the structure of the robot. Consider a humanoid with two 7-dof arms and two 6-dof legs, with the master at the center and local links communicating at 100Mbit/s. The maximum depth is 7 (master to arm-tip), and the longest chain is 15 (left arm to right arm). Each outward transaction takes around 3.84us per module, while inward transactions take around 5.12us. Total time spent transferring data for each module hop is 8.96us, so the upper bound on command rate is 111.6kHz. The actual limit will be lower due to computation time, parsing, thread switching, etc. The worst-case latency occurs on the longest chain. For a dynamic event in the humanoid's left hand to reach its right hand, the algorithm must complete 21

---

**Algorithm 3** Distributed Module Decoupling

---

```
1: procedure DECOUPLE( $j$ )
2:   if root and master module then
3:      $\bar{a}_j^r = -(I_j^c)^{-1}\bar{p}_0$ 
4:      $\bar{v}_j = \bar{0}$ 
5:   else if root and not master module then
6:      $\bar{a}_j^r = {}^jX_{pj} \text{ParentOutwardPacket.} \bar{a}_{pj}^r$ 
7:      $\bar{v}_j = {}^jX_{pj} \text{ParentOutwardPacket.} \bar{v}_{pj}$ 
8:   else ▷ has a joint
9:     CALCULATEJOINTKINEMATICS( $j$ )
10:    end if
11:    for child  $cj \in cjs$  do ▷ Recurse locally
12:      DECOUPLE( $cj$ )
13:    end for
14:     $I_j^c = I_j + \sum_{cjs} {}^{cj}X_j^T I_{cj}^c {}^{cj}X_j$  ▷ Calculate dynamics inwards
15:     $\bar{p}_j = I_j \bar{a}_j^r - (\bar{v}_j \times)^T I_j \bar{v}_j - {}^j f^{ext} + \sum_{cj} {}^{cj}X_j^T \bar{p}_{cj}$ 
16:    for children attached to this node do
17:      NOTIFYCHILD( $cj$ ) ▷ Send OP to child module  $cj$ 
18:    end for
19:    if root node and not master module then
20:      NOTIFYPARENT( $pj$ ) ▷ Send IP to parent module  $pj$ 
21:    end if
22:     $\tau_j = S_j^T (I_j^{Cj} \bar{a}_0 + \bar{p}_j)$ 
23: end procedure
```

---

---

**Algorithm 4** Distributed Module Decoupling Part 2

---

```

1: procedure CALCULATEJOINTKINEMATICS( $j$ )
2:    $\bar{S}^R = \bar{S}_j[0 : 2]$                                  $\triangleright$  Calculate kinematics outwards
3:    $\bar{S}^P = \bar{S}_j[3 : 5]$ 
4:    ${}^{j,in}E_j = \cos(q_j)1_{3x3} + \sin(q_j)(\bar{S}^R \times) + (1 - \cos(q_j))(\bar{S}^R \otimes \bar{S}^R)$ 
5:    ${}^jT_{j,in}^d = \begin{bmatrix} {}^{j,in}E_j & \bar{S}^P q_j \\ 0_{1x3} & 1 \end{bmatrix}^{-1}$ 
6:    ${}^jT_{pj} = {}^jT_{j,in}^d {}^{j,in}T_{pj}^s$ 
7:    ${}^jE_{pj} = {}^jT_{pj}[0 : 2, 0 : 2]$ 
8:    ${}^{pj}r_j = -{}^jE_{pj}^{-1} {}^jT_{pj}[0 : 2, 3]$ 
9:    ${}^jX_{pj} = \begin{bmatrix} {}^jE_{pj} & 0 \\ -{}^jE_{pj}({}^{pj}r_j \times) & {}^jE_{pj} \end{bmatrix}$ 
10:   $\bar{a}_j^r = {}^jX_{pj} \bar{a}_{pj} + \bar{v}_j \times (\bar{S}_j \dot{q}_j) + \bar{S}_j \ddot{q}_j$ 
11:   $\bar{v}_j = {}^jX_{pj} \bar{v}_{pj} + \bar{S}_j \dot{q}_j$ 
12: end procedure
13: procedure NOTIFYCHILD( $cj$ )
14:    $I_{cj,d}^c = \text{UNPACK}(\text{ChildInwardPacket.}\theta_{pj,d}^C)$ 
15:    $I_j^c += {}^{cj,d}X_j^T I_{cj,d}^c {}^{cj,d}X_j$ 
16:    $\bar{p}_j += {}^{cj,d}X_j^T \text{ChildInwardPacket.}{}^{pj,d}\bar{p}_j$ 
17:   ChildOutwardPacket.  $\bar{v}_{pj,d} = {}^{cj,d}X_j \bar{v}_j$ 
18:   ChildOutwardPacket.  $\bar{a}_{pj,d}^r = {}^{cj,d}X_j \bar{a}_j^r$ 
19:   SENDOUTWARDPACKET( $cj$ , ChildOutwardPacket)
20: end procedure
21: procedure NOTIFYPARENT( $pj$ )
22:   ParentInwardPacket.  $\theta_j = \text{PACK}({}^jX_{pj,d}^T I_j^{cj} X_{pj,d})$ 
23:   ParentInwardPacket.  ${}^{pj,d}\bar{p}_j = {}^jX_{pj,d}^T \bar{p}_j$ 
24:   SENDINWARDPACKET( $pj$ , ParentInwardPacket)
25: end procedure

```

---

(7 out + 7 in + 7 out) hops, inducing 188us of latency. The worst-case latency will thus be roughly equivalent to a 5.3kHz feedback loop, though the effective latency is lower for nearby modules. For example, if force is applied to the last arm module, that module's parent will compensate for it after only one hop (8.96us).

# Chapter 4

## Whole-Body Control

### 4.1 Overview

The whole-body control layer coordinates the movement of the entire additive robot system to perform a set of tasks. Each task specifies a desired geometric relationship between two frames and produces a set of joint accelerations that move the system toward the desired state. These joint accelerations are combined into a single set of prioritized joint accelerations using the prioritization scheme in Section 4.2. The core set of tasks consists of joint-space (Section 4.3), Cartesian goal-space (Section 4.4), and center of mass (Section 4.5) tasks, automatically derived from the functionality primitives of the modules making up the system. The core is augmented by an optional set of Cartesian goal-space tasks manually specified by the user. Section 4.6 presents the whole-body control algorithm in its entirety.

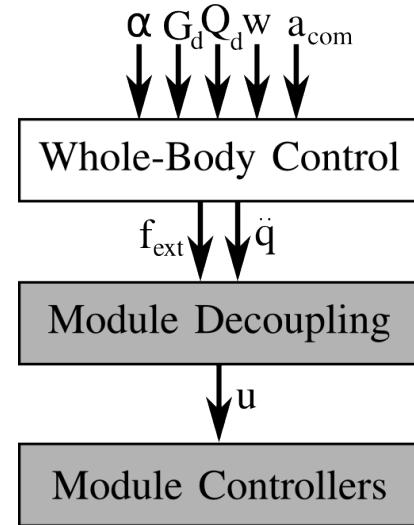


Figure 4.1: Whole-body Control

The inputs to this control layer are the priorities for each task ( $\alpha$ ), the goal-region setpoints for each Cartesian goal-space task ( $\mathbf{G}_d$ ), the setpoint for each joint ( $\mathbf{Q}_d$ ), and the desired spatial CoM acceleration ( $a_{com}$ ), and the CoM actuator weight vector  $w$ . Setpoints for disabled tasks ( $\alpha = 0$ ) will not affect the system, and do not need to be provided.

The prioritized joint accelerations and spatial forces produced by the whole-body control layer are interfaced to the robot via the module decoupling layer discussed in Chapter 3.

## 4.2 Task Prioritization

The joint accelerations produced by each task must be merged into a single set of accelerations before they can be sent to the module decoupling layer. This can be naively achieved by simply taking the average of all the joint accelerations. However, different tasks might need to use the same joints, and some tasks are more important than others; Falling over might result in permanent damage, but failing to flawlessly wave hello is not the end of the world. Thus, merging tasks based on priority is often beneficial.

Prioritized merging can be performed by weighting the average of the joint accelerations with each task's priority,  $\alpha \in [0.0, 1.0]$ . That is, if a task  $i \in \Omega$  affects joint  $j$  and has priority  $\alpha^i$ , the prioritized acceleration is  $\ddot{q}_j = \frac{\sum_{i \in \Omega} \ddot{q}_j^i \alpha^i}{\sum_{i \in \Omega} \alpha^i}$ . When the tasks are weighted in this way, higher priority / weight tasks are satisfied more quickly than low priority tasks.

Using scalar priorities has several advantages over schemes that use integer-based prioritization. The first advantage is that it's easy to turn tasks on and off: value of zero priority naturally indicates that the task is disabled and will not affect the robot's behavior. Second, the relative "strength" of a task with respect to the others gradually increases with priority. This allows for more fine-grained adjustment of priorities and allows for smooth, seamless transitioning between active tasks by simply interpolating the priority of each task up or down.

However, merging tasks in this way results in non-optimal control, since high priority tasks are not executed cleanly: Active tasks with lower priority still affect the execution of high priority tasks to some extent. On the other hand, because this technique is computationally fast, it allows the system to quickly correct the inaccuracies via feedback. The use of Cartesian goal regions also makes task conflicts less likely, since many tasks will continue to be satisfied even given small perturbations from other tasks.

### 4.3 Joint-space Tasks

A joint-space control task is automatically generated for each actuated joint in the additive robot system. These tasks allow individual joints to be controlled and are useful for manually controlling joints and executing behaviors scripted in joint-space. Each task has a separate priority, which allows individual joints to be turned on and off. The task controlling joint  $j$  takes as input the desired position, velocity, and feedforward acceleration  $Q_j^d = [q_j^d, \dot{q}_j^d, \ddot{q}_{j,ff}^d]$ . The

task also computes a PD feedback acceleration based on the current joint states to regulate position and velocity:  $\ddot{q}_{j,fb} = K_{j,p}(q_j^d - q_j) + K_{j,d}(\dot{q}_j^d - \dot{q}_j)$ . The feedback acceleration is added to the user-specified feedforward to produce the task's joint acceleration  $\ddot{q}_j = \ddot{q}_{j,ff} + \ddot{q}_{j,fb}$ .

#### 4.4 Goal-space Tasks

Cartesian goal-space tasks are generated for controlling the pose of the master module as well as all gripper, tool, wheel, directed sensor, payload, social, and contact surface functionality primitives.

Goal-space is a natural task representation that intuitively exploits the under-constrained nature of many real-world task specifications. A simple way to represent goal-space is with temporal sequences of Cartesian goal regions [1,2]. In this model, each goal region is defined as a time-varying boundary in the 6-dimensional manifold of special Euclidean group  $SE(3)$  where all solutions are equally valid (Figure 4.2(c)). The goal region is defined with respect to a goal frame, which can be a particular robot frame, a static inertial frame in the world, or the coordinate frame of some object in the environment.

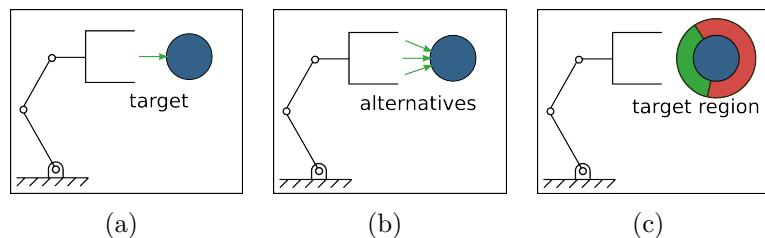


Figure 4.2: Single (a) vs. alternatively valid (b) targets and (c) a 2D visualization of a goal region (green).

Goal-space task control is an extension of traditional Cartesian-space task control, in which the controller attempts to move a frame to a specific target pose (Figure 4.2(a), green arrow). In fact, Cartesian-space setpoints are a special case of goal-space in which the goal region is infinitesimally small. Recognizing that there often exist multiple trajectories that achieve the same goal (Figure 4.2(b), green arrows), a goal-space task moves the controlled frame into the specified region (Figure 4.2(c), green region), but does not force it to be at any particular location within the region. This means that if a disturbance occurs that does not threaten the goal, it is simply allowed to happen.

Many high-level goals can naturally be described with temporal sequences of Cartesian goal regions. For example, consider the goal of picking up a cylindrical can from a table while remaining balanced. Since the can is a rotationally symmetric cylinder, it can be picked up from any angle around its primary axis, and at a variety of heights. For the robot to remain statically stable, its center of mass must be within the region defined by the convex hull of its contacts. The goals can thus be achieved by moving the gripper to the can at an *arbitrary* angle / height along its primary axis, then closing the gripper and lifting vertically with *arbitrary* position w.r.t. the table’s surface, all while keeping the robot’s center of mass at an *arbitrary* position within the footprint.

Designing goal regions is also often more intuitive to an untrained human than selecting a single Cartesian pose achievable by the robot—The average person can tell you that a cylinder can be picked up from any angle,

but few can intuitively select a specific pose that is within a robot's non-linear kinematic workspace. This is especially true for additive robot systems, since the kinematic workspace changes every time the robot's structure is altered.

Mathematically, a Cartesian goal region can be defined as a six-dimensional space: a set of minimum and maximum *roll*, *pitch*, *yaw*, *x*, *y*, and *z* that define a region of space relative to the goal frame. That is, using  $\bar{G}_\Delta = [\Delta_R, \Delta_P, \Delta_Y, \Delta_x, \Delta_y, \Delta_z]^T \in SE(3)$  as the delta between the controlled frame and the goal frame (or equivalently, the pose of the controlled frame in the goal frame),  $\bar{G}_\Delta \in [\bar{G}_{min}, \bar{G}_{max}]$  indicates containment within the goal region. The following piecewise-linear function thus represents the "goal error" (see

Figure 4.3 for a 1-D example):  $\bar{G}_{err} = \begin{cases} \bar{G}_\Delta - \bar{G}_{max} & \bar{G}_\Delta \geq \bar{G}_{max} \\ \bar{G}_{min} - \bar{G}_\Delta & \bar{G}_\Delta \leq \bar{G}_{min} \\ 0 & \text{otherwise} \end{cases}$

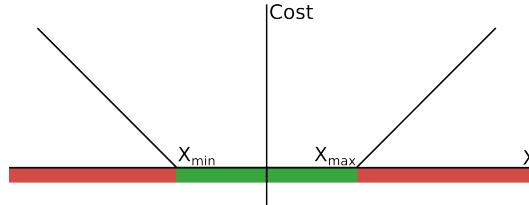


Figure 4.3: Illustration of one-dimensional goal error

To perform Cartesian goal-space task control, each task computes a Cartesian acceleration that reduces the goal error and dampens oscillations:  $\bar{a} = K_p G_{err} - K_d \dot{G}_\Delta$ . Note that when a dimension of the controlled frame's pose is inside the goal region, the corresponding position feedback dimension is zero and the damping term dominates. The task's behavior is thus that the

controlled frame accelerates toward the goal region while outside of it, then slows to a stop after it enters.

The Cartesian acceleration is converted into joint accelerations using resolved acceleration techniques [8, 25]. With  $J$  as a floating-base jacobian and  $\mathbf{q}^s$  as a vector the positions of the joints that support the controlled frame,  $\bar{a} = \dot{J}\dot{\mathbf{q}}^s + J\ddot{\mathbf{q}}_d^s$ . The joint accelerations  $\ddot{\mathbf{q}}_d^s$  can then be solved using the jacobian pseudoinverse:  $\ddot{\mathbf{q}}_d^s = J^\dagger(\bar{a} - \dot{J}\dot{\mathbf{q}}^s)$ . Since the pseudoinverse requires a matrix inversion, the time complexity of this transformation is  $O(n^3)$ . The Jacobian transpose method could be used instead for faster computation, at the cost of accuracy in some configurations:  $\ddot{\mathbf{q}}_d^s = J^T(\bar{a} - \dot{J}\dot{\mathbf{q}}^s)$ .

## 4.5 CoM Control

Controlling the robot’s center of mass is critical for safely moving an additive robot system around in the world. For example, the center of mass must be within the support polygon for a the robot to be statically stable, and the ZMP [28] / CMP [34] must be carefully controlled during dynamic maneuvers. For non-articulated robots like quadcopters, the CoM’s acceleration is essentially the acceleration of the entire craft.

The robot’s CoM is accelerated by external forces exerted on the robot by the environment. Modules providing the thruster, flywheel, wheel, gripper, contact surface, and fluid surface functionality primitives must thus be coordinated to achieve the desired CoM acceleration.

The robot's *com* frame positionally lies at its current center of mass, with orientation matching that of the inertial frame (with the  $z$  axis pointing "up" w.r.t. gravity). With  $f_i \in F$  as the set of controllable forces acting on the robot, and  $f_i^{un} \in F^{un}$  uncontrollable, the forces needed to cause a CoM acceleration of  $\bar{a}_{com}$  can be calculated with  $I_{com}^C \bar{a}_{com} - \sum_{F^{un}} {}^{com}X_i^T \bar{f}_i^{un} = \bar{f}_{com}^{desired} = \sum_F {}^{com}X_i^T \bar{f}_i$ . Note that most forces are not directly applied in the frame of the center of mass (*com*), and must be transformed with  ${}^{com}X_i^T$ . For example, if force is applied to the robot's *foot*,  ${}^{com}X_{foot}^T$  is the matrix that transforms forces from the *foot* frame to the *com* frame. Controlling the CoM acceleration can thus be achieved by selecting the appropriate  $F$  in relation to the current  $F^{un}$ .

To determine the desired  $F$ ,  $F^{un}$  must first be estimated. The components of  $F^{un}$  are gravity, buoyancy, lift, and drag. These forces can't be instantaneously changed by the robot, and are thus considered uncontrollable. Gravitational force is the familiar combination of composite mass and gravity:  $f_g = m^C \bar{g}$ . Buoyancy depends on the density  $\rho$  of the ambient fluid and the volume  $V$  of the robot  $f_B = \rho V \bar{g}$ . Fluid surface lift and drag are functions of the geometry and spatial velocity of each module, and thus must be computed with the fluid force function included in modules with the fluid surface functionality primitive.

Once  $f_{com}^{desired}$  has been estimated,  $F$  can be found by solving a linear program (LP): minimize  $w^T \Lambda$  s.t.  $f_{com}^{desired} = J^{com} \Lambda$ ,  $(\Lambda - \Lambda_{max}) \leq 0$ ,  $(\Lambda_{min} - \Lambda) \leq 0$ , with other constraints dependent on which actuators are available.

$\Lambda \in R^N$  is a vector of bounded force-scaling variables to be optimized and  $J^{com} \in R^{6 \times N}$  is a Jacobian that maps these variables to spatial forces in the CoM frame.  $w \in R^N$  is a vector weighting each force source, which allows some actuators to be favored over others. Each functionality primitive providing controllable force (thrusters, flywheels, wheels, grippers, and contact surfaces) adds to  $\Lambda$ ,  $J^{com}$ , and the constraints in different ways.

Thrusters and flywheels exert force along the spatial axis of the actuator, scaled by the actuator command:  $f_i = \lambda_i \bar{S}$ . They thus increase  $N$  by one and the Jacobian column is  $J_i = {}^{com}X_i^T \bar{S}$ . The bounds on  $\lambda_i$  are determined by the minimum and maximum command of the actuator.

Wheels also exert a force along a single axis, but the force is applied normal to the rotational axis, tangent to the surface of the wheel. Using  $iw$  as a frame at the wheel's contact point with its  $z$  axis indicating the force direction,  $J_i = ({}^{com}X_i^{Ti} X_{iw}^T)[0, 0, 0, 0, 0, 1]^T$ . The  $\lambda_i$  bounds are again based on the actuator.

Grippers that are strong enough for locomotion provide a full 6-DoF wrench. They thus increase  $N$  by six, with each  $J_i$  a column of  ${}^{com}X_i^T$ . The true minimum and maximum  $\lambda_i$  for a given joint configuration can be derived by mapping the minimum and maximum joint actuator forces through a Jacobian. However, the gripper is often the weak link, so its maximum grip force is used instead to save on computation.

Contact surfaces are significantly more complicated than the other force

sources. The correct way to model them is to account for the geometry, friction, and deformation of both the robot and the environment. However, to keep things simple (and computationally tractable), the contact surface is here assumed to be a rigid planar surface, the entirety of which is in contact with an equally flat and rigid portion of the environment. This is blatantly inaccurate in most cases, but is fast enough that feedback can typically compensate for the inaccuracies.

Like a gripper, planar contact surfaces allow 6-DoF wrenches to be exerted, and the added Jacobian entries are again the columns of  ${}^{com}X_i^T$ . However, unlike a gripper, a contact surface is not fully connected to the environment—a surface can only apply force within a friction cone without losing contact, and must keep their CoP within the contact polygon to avoid rolling. Additional constraints must be added to account for this. Each contact surface defines a contact frame with the  $z$ -axis pointing into the contact surface.  $\lambda_i^z$  is the normal force, while  $\lambda_i^x$  and  $\lambda_i^y$  are tangential forces.  $\lambda_i^R$ ,  $\lambda_i^P$ , and  $\lambda_i^Y$  represent the moments about the  $x$ ,  $y$ , and  $z$  axes respectively.

The normal force must always be positive—that is, the environment can only push on a planar contact surface, not pull. This is reflected in the constraint  $\lambda_i^z \geq 0$ .

The tangential forces must be kept lower than the friction coefficient  $\mu$  times the normal force. This constraint is properly modeled by a cone constraint:  $\sqrt{(\lambda_i^x)^2 + (\lambda_i^y)^2} \leq \mu_i \lambda_i^z$ . However, that constraint requires the problem to be converted into a second order cone problem. An inscribed

friction pyramid is used instead to retain LP compatibility. This constraint is more conservative than the full friction cone. The friction constraints are thus  $|\lambda_i^x| \leq \frac{1}{\sqrt{2}}\mu_i\lambda_i^z, \lambda_i^y \leq \frac{1}{\sqrt{2}}\mu_i\lambda_i^z$ .

The center of pressure is the point on the plane at which the tangential moments are zero. If the CoP is at the edge of the contact polygon, the surface will begin to break contact with the ground. The CoP is located at  $[-\lambda_i^P/\lambda_i^z, \lambda_i^R/\lambda_i^z, 0]$ . Given a  $d^x \times d^y$  rectangular “safe” area centered inside the contact polygon, the constraints for preventing contact loss are  $|\lambda_i^P| \leq \frac{1}{2}d^x\lambda_i^z, |\lambda_i^R| \leq \frac{1}{2}d^y\lambda_i^z$ .

The result of the LP optimization is the  $\Lambda$  vector, which can be easily converted into the set of actuator-frame forces  $F$  as described above. When  $F$  is sent to the module decoupling layer, additional torque is applied to each joint to compensate for the force. Compensating for the forces in this way allows the robot to be treated as a rigid body rather than an articulated body, which greatly simplifies the CoM control calculations.

## 4.6 Centralized Optimization

Algorithm 5 compiles the components of the whole body controller discussed in previous sections into a single centralized algorithm. As in the module decoupling layer, the algorithm operates on the compressed model and Featherstonian spatial vector algebra is used recursively to efficiently compute the robot’s kinematics and dynamics. The algorithm also assumes that the variables calculated in the module decoupling layer are available.

---

**Algorithm 5** Centralized Whole-body Control

---

```

1: procedure WBC( $\alpha, \mathbb{Q}, \mathbb{G}, a_{com}, \mathbf{w}$ )
2:   for task  $Q_i \in \mathbb{Q}, \alpha_i > 0$  do
3:      $\alpha_{total} += \Gamma \alpha_i$             $\triangleright \Gamma$  maps  $\alpha_i$  to the approp. entries of  $\alpha_{total}$ 
4:      $\ddot{q}_{total} += \text{COMPUTEJOINTTASK}(Q_i)$ 
5:   end for
6:   for task  $G_i \in \mathbb{G}, \alpha_i > 0$  do
7:      $\alpha_{total} += \Gamma \alpha_i$ 
8:      $\ddot{q}_{total} += \text{COMPUTECGSTASK}(G_i)$ 
9:   end for
10:   $\ddot{q}_{total} ./= \alpha_{total}$ 
11:   $F = \text{COMPUTECOMTASK}(a_{com}, \mathbf{w})$ 
12:  DECOUPLE( $\ddot{q}_{total}, F$ )
13: end procedure
14: procedure COMPUTEJOINTTASK( $Q_i$ )
15:   return  $\ddot{q}_j = \ddot{q}_{j,ff}^d + K_{j,p}(q_j^d - q_j) + K_{j,d}(\dot{q}_j^d - \dot{q}_j)$ 
16: end procedure
17: procedure COMPUTECGSTASK( $G_i$ )
18:    $G_{i,err} = \begin{cases} G_{i,\Delta} - G_{i,max} & G_{i,\Delta} \geq G_{i,max} \\ G_{i,min} - G_{i,\Delta} & G_{i,\Delta} \leq G_{i,min} \\ 0 & otherwise \end{cases}$ 
19:    $\bar{a} = K_p G_{i,err} - K_d \dot{G}_{i,\Delta}$ 
20:    $\ddot{\mathbf{q}}_d^s = J^T(\bar{a} - J \dot{\mathbf{q}}^s)$ 
21: end procedure
22: procedure COMPUTECOMTASK( $a_{com}, \mathbf{w}$ )
23:    $F^{un} = m^C \bar{g} + \rho V \bar{g} + \sum_{j \in FS-FPs} f_j(\bar{v}, \rho, x)$ 
24:   for actuator  $A \in \mathbb{A}, w_A > 0$  do
25:      $\mathbb{C} \leftarrow C_A$             $\triangleright C_A, J_A$  are A's constraint and jacobian entries
26:      $J^{com} \leftarrow J_A$ 
27:   end for
28:   Solve minimize  $\mathbf{w}^T \Lambda$  s.t.  $f_{com}^{desired} = J^{com} \Lambda, \mathbb{C}$ 
29: end procedure

```

---

## Chapter 5

### Future Work

This work presented a framework for module interfacing, modeling, and control in additive robot systems. However, there are improvements and additional features that could be pursued in the future to improve additive robot systems.

The proposed scalable dock was designed with electrical signals in mind, and may not be sufficient for all types of robots. Some connectors need to transfer more than just electricity; a hydraulic or pneumatic robot will also need pressurized conduits in which to move fluid. Tendon-based robots need extra holes and pulleys to route the cables that move the robot. A connector used in an underwater robot needs to be water-tight, and may need to have particular hydrodynamic properties. To make additive robot systems compatible with these types of robots, the docking system will need to be improved.

While the presented dynamics calculations include everything normally modeled in rigid body dynamics, they still ignore many aspects of physics. Some things that could be added to improve the accuracy of the calculations follow:

1. Additional aerodynamic effects, such as the magnus and ground effects.

2. “Closure frames” to close kinematic loops within the robot and improve support for linkages.
3. Detailed friction models for wheels and other contacts (slipping / skidding, hydroplaning, etc)
4. Support for dynamical actuator models, rather than static mappings.
5. Contact constraints that work for ground and contact surface modules that are soft and / or non-flat.

Controller gain tuning and model calibration is often tedious and time consuming. Since the proposed control scheme still has many gains / parameters, automating this process would make additive robot systems easier to use. One possible approach is to use a robust adaptive controller for both control and model refinement. The trajectory tracking stability and model parameter convergence of adaptive controllers can be mathematically proven, if the mathematical structure of each modules model is assumed to be correct. The controller can also automatically modulate its gains based on how well its internal model fits its observations of the system. This will allow it to reject error and retain stability when the model is not well known, while still enabling smooth and compliant motion once the model converges to the correct parameters.

Finally, the list of functionality primitives is necessarily incomplete. While it covers most robots in use today, there will likely be new innovations

in the future. The list of functionality primitives will thus need to be expanded in the future to ensure that all possible robots can be constructed as additive robot systems.

## Bibliography

- [1] Dmitry Berenson, Joel Chestnutt, Siddhartha S Srinivasa, James J Kuffner, and Satoshi Kagami. Pose-constrained whole-body planning using task space region chains. In *IEEE-RAS International Conference on Humanoid Robots*, 2009.
- [2] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson , Alvaro Collet Romea, and James Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, 2009.
- [3] Z. M. Bi, W. A. Gruver, W. J. Zhang, and S. Y. T. Lang. Automated modeling of modular robotic configurations. *Robot. Auton. Syst.*, 54(12):1015–1025, December 2006.
- [4] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Programmable parts: a demonstration of the grammatical approach to self-organization. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3684–3691, Aug 2005.
- [5] S.P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

- [6] J. Campbell, P. Pillai, and S. C. Goldstein. The robot is the tether: active, adaptive power routing modular robots with unary inter-robot connectors. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4108–4115, Aug 2005.
- [7] S.K. Chan and P.D. Lawrence. General inverse kinematics with the error damped pseudoinverse. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 834–839 vol.2, 1988.
- [8] B. Dariush, G. Bin Hammam, and D. Orin. Constrained resolved acceleration control for humanoids. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 710–717, Oct 2010.
- [9] Prithviraj Dasgupta, Jose Baca, S.G.M. Hossain, Ayan Dutta, and Carl Nelson. Mechanical design and computational aspects for locomotion and reconfiguration of the modred modular robot. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1359–1360, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.
- [10] J. Davey, J. Sastra, M. Piccoli, and M. Yim. Modlock: A manual connector for reconfigurable modular robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3217–3222, Oct 2012.
- [11] N. Eckenstein and M. Yim. Design, principles, and testing of a latching modular robot connector. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2846–2851, Sept 2014.

- [12] A. Faiña, F. Orjales, F. Bellas, and R. J. Duro. First steps towards a heterogeneous modular robotic architecture for intelligent industrial operation. In *Workshop on Reconfigurable Modular Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, 2011.
- [13] Andres Faina, Felix Orjales, Daniel Souto, Francisco Bellas, and Richard Duro. *A Modular Architecture for Developing Robots for Industrial Applications*. River Publishers, 5 2015.
- [14] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [15] Wenbin Gao and Hongguang Wang. *Advances in Reconfigurable Mechanisms and Robots I*, chapter An Automatic Dynamics Generation Method for Reconfigurable Modular Robot, pages 551–560. Springer London, London, 2012.
- [16] A. Giusti and M. Althoff. Automatic centralized controller design for modular and reconfigurable robot manipulators. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3268–3275, Sept 2015.
- [17] J.M. Hollerbach and Ki Suh. Redundancy resolution of manipulators through torque optimization. *Robotics and Automation, IEEE Journal of*, 3(4):308–316, 1987.

- [18] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 2858–2863 vol.4, May 1998.
- [19] L. Winkler J. Liedke, R. Matthias and H. Worn. The collective self-reconfigurable modular organism (cosmo). In *2013 IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, page 16, 2013.
- [20] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund. Modular atron: modules for a self-reconfigurable robot. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 2068–2073 vol.2, Sept 2004.
- [21] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Journal of Robotics and Automation*, 3(1):483–501, 2010.
- [22] K. Kotay, D. Rus, M. Vona, and C. McGraw. The self-reconfiguring robotic molecule. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 424–431 vol.1, May 1998.
- [23] K. Kotay, D. Rus, M. Vona, and C. McGraw. The self-reconfiguring robotic molecule. In *Robotics and Automation, 1998. Proceedings. 1998*

*IEEE International Conference on*, volume 1, pages 424–431 vol.1, May 1998.

- [24] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. M-tran ii: metamorphosis from a four-legged walker to a caterpillar. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2454–2459 vol.3, Oct 2003.
- [25] J. Luh, M. Walker, and R. Paul. Resolved-acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, 25(3):468–474, Jun 1980.
- [26] Oussama Khatib Luis Sentis, Jaeheung Park. Compliant control of multi-contact and center-of-mass behaviors in humanoid robots. *Transactions on Robotics*, 26(3):483–501, 2010.
- [27] E. Meister, E. Nosov, and P. Levi. Automatic onboard and online modelling of modular and self-reconfigurable robots. In *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 91–96, Nov 2013.
- [28] Branislav Borovac Miomir Vukobratovic. Zero-moment point thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157173, 2004.

- [29] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.
- [30] J. Neubert, A. Rost, and H. Lipson. Self-soldering connectors for modular robots. *IEEE Transactions on Robotics*, 30(6):1344–1357, Dec 2014.
- [31] OSRF. Gazebo.
- [32] OSRF. Urdf.
- [33] C. Parrott, T. J. Dodd, and R. Gro. Higen: A high-speed gender-less mechanical connection mechanism with single-sided disconnect for self-reconfigurable modular robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3926–3932, Sept 2014.
- [34] Marko B. Popovic, Ambarish Goswami, and Hugh Herr. Ground reference points in legged locomotion: Definitions, biological trajectories and control implications. *The International Journal of Robotics Research*, 24(12):1013–1032, 2005.
- [35] M. Rubenstein, K. Payne, P. Will, and Wei-Min Shen. Docking among independent and autonomous conro self-reconfigurable robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2877–2882 Vol.3, April 2004.

- [36] D. Rus and M. Vona. A basis for self-reconfiguring robots using crystal modules. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2194–2202 vol.3, 2000.
- [37] L. Sentis and O. Khatib. Control of free-floating humanoid robots through task prioritization. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1718–1723, April 2005.
- [38] L. Sentis and O. Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4):505–518, 2005.
- [39] J.J.E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.
- [40] B. Stephens. *Push Recovery Control for Force-controlled Humanoid Robots*. Technical report (Carnegie-Mellon University. Robotics Institute). Carnegie Mellon University, The Robotics Institute, 2011.
- [41] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: mechanical design of a module for self-reconfigurable robotics. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 4095–4101 vol.4, 2002.
- [42] Cem Unsal and Pradeep K. Khosla. Solutions for 3d self-reconfiguration in a modular robotic system: implementation and motion planning, 2000.

- [43] H. Elliott W. A. Wolovich. A computational technique for inverse kinematics. In *Conference on Decision and Control*, pages 1359–1363.
- [44] Zhong-Jun Zhang Zeungnam Bien Joris De Schutter Wen-Hong Zhu, Yu-Geng Xi. Virtual decomposition based control for generalized high dimensional robotic systems with complicated structure. *IEEE Transactions on Robotics and Automation*, 13(3):411–436, 1997.
- [45] P. J. White, K. Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2888–2893 Vol.3, April 2004.
- [46] D. E. Whitney. The mathematics of coordinated control of prosthetic arms and manipulators. *Journal of Dynamic Systems, Measurement, and Control*, pages 303–309, 1972.
- [47] K. C. Wolfe, M. S. Moses, M. D. M. Kutzer, and G. S. Chirikjian. M3express: A low-cost independently-mobile reconfigurable modular robot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2704–2710, May 2012.
- [48] Kazuya Yoshida Yoji Umetani. Resolved motion rate control of space manipulators with generalized jacobian matrix. *Transactions on Robotics and Automation*, 5(3):303–314, 1989.

- [49] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. Self-reconfigurable modular robots -hardware and software development in aist. In *Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003 IEEE International Conference on*, volume 1, pages 339–346 vol.1, Oct 2003.
- [50] Yonghwan Oh Youngjin Choi, Doik Kim and Bum-Jae You. Posture/walking control for humanoid robot based on kinematic resolution of com jacobian with embedded motion. *Transactions on Robotics and Automation*, 23(6):1285–1293, 2007.
- [51] Chih-Han Yu and Radhika Nagpal. A self-adaptive framework for modular robots in dynamic environment: Theory and applications. *The International Journal of Robotics Research*, 2010.
- [52] Chih-Han Yu and Radhika Nagpal. A self-adaptive framework for modular robots in dynamic environment: Theory and applications. *The International Journal of Robotics Research*, 2010.
- [53] V. Zykov, E. Mytilinaios, M. Desnoyer, and H. Lipson. Evolved and designed self-reproducing modular robotics. *IEEE Transactions on Robotics*, 23(2):308–319, April 2007.