Controlling Floating-Based Robots

Nathan Ratliff

April 22, 2014

Abstract

Floating-based legged robots are some of the most impressive and fascinating robots in existence. But they're also some of the most difficult to control. This document presents and analyzes the constrained equations of motion for a floating-based robot in contact with the environment. We discuss the issues that arise when we detach the robot from the world and make it mobile, and we examine the linear structure connecting dynamic quantities such as accelerations, torques, and contact forces to one another. After getting our hands dirty with objective design for control systems, exploring in detail the specific problem of controlling a robot's center of mass and some of the difficulties that arise in that context with designing objectives alone, we present the state-of-the-art building block for controller design, Quadratic Programming, the centerpiece of most of today's most advanced control systems. Almost all advanced torque controlled robots today are built on these tools and dynamic principles.

1 What is a floating based robot?

This document explores what we call "floating-based robots". What makes a floating-based robot different from other types of robots?

Traditionally, many texts first introduce kinematics and dynamics for robots that are *fully actuated* and specifically *attached to the ground* (e.g. Siciliano et al. (2010)). That means the robot, as shown in Figure 1 on the left, is bolted to the ground so tightly that we assume it will never move. The equations of motion for these robots take the form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}. \tag{1}$$

These equations relate how torques $\boldsymbol{\tau}$ on the joints turn into accelerations $\ddot{\boldsymbol{q}}$ of those joints.[1] Importantly, we assume here that when the environment exerts a

[1] The term $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ encodes both fictitious force terms (Coriolis and centrifugal forces (see Appendix A)), as well as forces such as gravity or friction. Some texts separate out the fictitious forces that arise from the non-inertial coordinate system from physical forces that we're modeling such as gravity or friction. Here, though, we care primarily about the linear algebra of these equations of motion so we follow the common tradition for this analysis of lumping them all into one term.

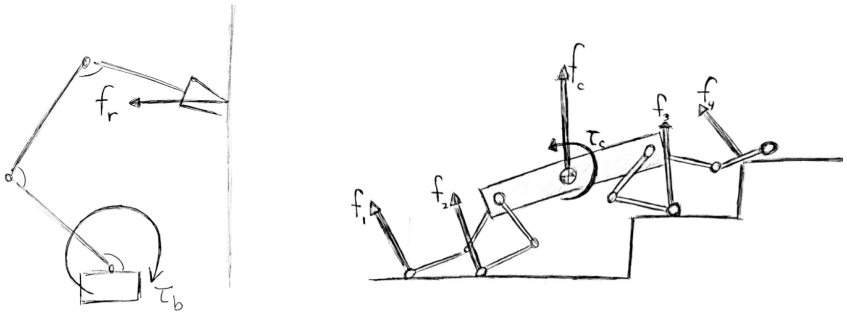

Figure 1: **Left:** Fixed based (manipulator) robot. When the robot contacts the wall, the wall's reaction force $\boldsymbol{f}_r$ translates to a torque at the base, and the ground compensates with a torque $\boldsymbol{\tau}_b$ to keep the base fixed in place. **Right:** Floating based (quadrupedal) robot. Contact reaction forces $\boldsymbol{f}_i$ at each foot translate to net linear force $\boldsymbol{f}_c$ and torque $\boldsymbol{\tau}_c$ around the center of mass.

force at the end-effector the robot's (infinitely strong and sturdy) bolts absorb all of that force so we don't need to worry about it.

Practically, this fixed-based assumption simplifies the problem significantly. The problems born from Newton's laws of reaction are assumed away in this framework because when those bolts are strong enough, the ground effectively becomes an infinite force generator. As the manipulator moves, creating significant inertia from high velocities and heavy masses, it is constantly sucking energy from the floor in the form of reaction forces automatically acting on the base to ensure that it stays fixed in place. With these systems we need to worry less about the system's *passive dynamics*[2] since almost all desired accelerations simply have an associated torque that would generate them as given by Equation 1.

Fixed-based manipulators are a core technology in factories and they've been making sophisticated and high throughput manufacturing processes possible for years. They put cars in our driveway and electronics in our homes. But some of the most interesting dynamical systems come from the borderline science fiction robots that walk around on two or four legs. These robots are what we call *floating-based* robots. The base isn't bolted to the floor. It instead floats above the ground and moves from place to place with the robot. At any moment in time, the robot must choose carefully torques to apply that will induce the perfect reaction forces from the environment at its points of contact so that those reaction forces correctly manipulate its free floating base.

If we remove those bolts from the floor, every contact the robot makes with the environment (see Figure 1 (right)) with produce equal and opposite reaction forces that aren't canceled by the environment. More than that, the robot can

[2]Passive dynamics are the dynamics that would arise from zero applied torque, such as continuing along its current inertial trajectory or falling under the force of gravity.

only get the environment to induce certain types of forces: those generated by a combination of pressing against the surface and surface friction. We've all experience what happens when we try to push off the ground with slightly too much sheer force (component tangent to the ground). Our foot slips! Friction isn't able to generate a force strong enough to keep our foot in place and our foot moves. Section 4.2 discusses techniques for modeling these constraints and Section 4 presents a general framework for handling constraints of this form in control.

But the lack of reaction force absorption isn't the only issue we now have to address. We also now have an *underactuated* system. Since the base of the robot is free to move around, we can no longer fully control all degrees of freedom. We often choose the floating-base of the robot to be situated somewhere around the robot's torso near the robot's center of mass[3] These floating base dimensions consist of three positional coordinates and three rotational components totaling a full six degrees of freedom that we can't directly control. The robot doesn't have a collection of jet engines attached to its torso that can directly induce forces and torques on the floating base. All forces and torques on that base must be generated by some contact with the environment. Physically this means that in order to fully control these unactuated degrees of freedom, the robot needs to reason about how it can generate the requisite reaction forces from the environment around it.

We'll explore how this redundancy manifests in the linear algebra of the equations of motion in Section 2. In a very explicit sense, we can view the robot's interactions with the environment during locomotion as a set of movements designed explicitly to generated forces that manipulate the robot's base. Section 3.1 explores this connection.

We start with an analysis of the space of (environmentally) constrained dynamic variables for floating-based systems.

2 The linear subspace of dynamic variables

A floating-based robot's configuration consists of two sets of variables. The first collection of dimensions $\boldsymbol{q}_{\text{base}} \in \mathbb{R}^6$ describe the configuration of the base, giving the floating base's position (3 Cartesian coordinates) and orientation (3 rotational coordinates). This base configuration $\boldsymbol{q}_{\text{base}}$ effectively forms the frame from which we can describe the remaining configuration of the robot. These remaining components $\boldsymbol{q}_{\text{jnt}} \in \mathbb{R}^d$ in turn describe rest of the robot relative to the floating base in terms of a collection of joint angles[4] defining the configuration

[3]Note that any fixed floating base that we choose can never *always* be at the center of mass since that center of mass point moves as the robot reconfigures its rigid body components.

[4]Typically, in this document, we'll refer to the joints as joint angles since *revolute* joints (that define some axis of revolution connecting two rigid body parts) are the most common in modern robotic systems. Many systems have *prismatic* joints as well, which are joints that control the offset between two body parts, but the mathematics behind such joints work out very similarly so revolute joints make a good mental model. (Different joint types change the robot's kinematics, but we mathematically abstract those differences away using the forward

of the robot's rigid body components relative to the base. The full configuration of the robot is then just the concatenation of these two sets of variables into a single vector

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_{\text{base}} \\ \boldsymbol{q}_{\text{jnt}} \end{bmatrix} \in \mathbb{R}^{d+6}. \tag{2}$$

Importantly, though, the robot can only directly apply forces to the d joints $\boldsymbol{q}_{\text{jnt}} \in \mathbb{R}^d$. Thus, the vector of applied joint torques is $\boldsymbol{\tau} \in \mathbb{R}^d$. And since these torques only apply to the bottom d components of $\boldsymbol{q}$, we need a simple *actuation* matrix of the form $\boldsymbol{S} = [\boldsymbol{0}; \boldsymbol{I}]$ that buffers the applied torques with zeros in the following way

$$\boldsymbol{S\tau} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \end{bmatrix} \tag{3}$$

to fit it into the equations of motion. In generally, the actuation matrix $\boldsymbol{S}$ may be more sophisticated, defining simply how a collection of d generalized forces $\boldsymbol{\tau} \in \mathbb{R}^d$ transform into forces on the $(d+6)$-dimensional space. But the above matrix models the common floating-based underactuated case and we'll take it as our running example throughout this document.

2.1 Environmental constraints on the system

Since we now need to reason explicitly about how contact forces affect the system, we need to add them to the model. Suppose $\phi : \mathbb{R}^{d+1} \to \mathbb{R}^k$ gives the mapping from the configuration $\boldsymbol{q}$ to a k-dimensional task space within which the environment may be able to generate forces. A canonical task space which is easy to think about is simply the Cartesian space $\mathbb{R}^3$, in which case ϕ may map the robot's configuration (base and joint angles) to a point on the robot's body, such as a fingertip or a foot. Environmental reaction forces $\lambda \in \mathbb{R}^k$ act on the robot as $\boldsymbol{\tau}_{\text{task}} = \boldsymbol{J}_\phi^T \lambda$. The transpose of the map's Jacobian[5] tells us how reaction forces in the range space of ϕ translate to generalized forces on the robot's full configuration. See Ratliff (2014) for a derivation of this relation.

Using this notation, the unconstrained equations of motion for a floating-based robot are simply

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S\tau} + \boldsymbol{J}_\phi^T \boldsymbol{\lambda}. \tag{4}$$

Here $\boldsymbol{M}$ is the generalized inertia matrix and $\boldsymbol{h}$ plays the same role as $\boldsymbol{h}$ in the fixed-based equations of motion. The only difference is that here we need our actuation matrix $\boldsymbol{S}$ and now we've introduced end-effector forces $\boldsymbol{\lambda}$. Note that

kinematics function and its derivatives).

[5]Kinematics may largely be described as the application of calculus to differentiable maps that arise in robotics, so when we say "Jacobian" we're literally just referring to the matrix of partial derivatives of the map $\frac{\partial \phi}{\partial \boldsymbol{q}}$. Traditionally, in the robotics literature this matrix of partials is often denoted $\boldsymbol{J}_\phi$. We keep to that tradition here, but emphasize that $\boldsymbol{J}_\phi$ is nothing more than the familiar Jacobian from calculus.

as of yet the robot is not constrained by the environment, we have only described how forces $\boldsymbol{\lambda}$ in the task space (for instance, applied to the end-effector) relate to $\boldsymbol{\tau}$ and $\ddot{\boldsymbol{q}}$. This equation alone is only applicable if the robot were floating somewhere out in space and $\boldsymbol{\lambda}$ were generated by thrusters attached to its end-effectors.

But now say this force $\boldsymbol{\lambda}$ is a force generated at the robot's end-effector by a fixed contact with the ground or a wall. We now have an additional set of constraints: the robot's end-effector shouldn't move.

Specifically, if ϕ is the forward kinematics map as defined above, a fixed contact says that $\phi(\boldsymbol{q}) = \boldsymbol{x}_c$ is constant. Another way of saying that that point doesn't change over time $\frac{d}{dt}\phi(\boldsymbol{q}) = \boldsymbol{J}_\phi \dot{\boldsymbol{q}} = 0$. Our equation, though, cares about accelerations so we need to go one step further. Taking another time derivative we get $\frac{d^2}{dt^2}\phi(\boldsymbol{q}) = \boldsymbol{J}_\phi \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_\phi \dot{\boldsymbol{q}} = 0$. This constraint is linear in $\ddot{\boldsymbol{q}}$ which makes it very nice analytically for use in Equation 4.

More generally we can model a broad class of constraints as linear equality constraints in $\ddot{\boldsymbol{q}}$ of the form[6]

$$\boldsymbol{A}(\boldsymbol{q}, \dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} = \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{5}$$

In the above fixed end-effector example above, $\boldsymbol{A} = \boldsymbol{J}_\phi$ and $\boldsymbol{b} = -\dot{\boldsymbol{J}}_\phi \dot{\boldsymbol{q}}$. Our example is a form of holonomic constraint (a constraint described as $g(\boldsymbol{q}) = 0$), but this framework allows us to model even nonholonomic constraints of the form $g(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0$ since taking the first time derivative of that equation gives

$$\underbrace{\frac{\partial g}{\partial \dot{\boldsymbol{q}}}}_{\boldsymbol{A}} \ddot{\boldsymbol{q}} = \underbrace{-\frac{\partial g}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}}}_{\boldsymbol{b}}. \tag{6}$$

Since we want to analyze the effect of forces generated in the task space defined by the map ϕ, generally our constraints will take the form $\boldsymbol{J}_\phi \ddot{\boldsymbol{q}} = \boldsymbol{b}$. Combining this equation with the unconstrained Equation 4, we get our full constrained equations of motion:

$$\begin{aligned} &\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_\phi^T \boldsymbol{\lambda} \\ &\boldsymbol{J}_\phi \ddot{\boldsymbol{q}} = \boldsymbol{b}. \end{aligned} \tag{7}$$

This equation, which we refer to as the floating-based constrained equations of motion, describes specifically how task constraints, such as $\phi(\boldsymbol{q}) = \boldsymbol{x}_c$, enforce a highly constrained relationship between task space forces $\boldsymbol{\lambda}$, generalized forces in joint space $\boldsymbol{\tau}$, and the kinematic accelerations of the robot $\ddot{\boldsymbol{q}}$.

2.2 Connections between dynamic variables

This section presents some simple manipulations of the above equations that reveal the specific linear connections between variables. We especially analyze

[6] Most generally this analytical framework can handle any time varying constraint of the form $\boldsymbol{A}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)\ddot{\boldsymbol{q}} = \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)$ (see Udwadia & Kalaba (1996)).

the space of combined values $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, $\boldsymbol{\lambda}$ to gain some intuition on how the variables relate to one another.

2.2.1 Relating contact forces to joint torques

First, note that since $\boldsymbol{M}$ is invertible (it's symmetric positive definite), we can expose $\ddot{\boldsymbol{q}}$ and then eliminate it using the constraint equation:

$$\begin{aligned} & \ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\left(\boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_\phi^T\boldsymbol{\lambda} - \boldsymbol{h}\right) \\ \Rightarrow & \underbrace{\boldsymbol{J}_\phi\ddot{\boldsymbol{q}}}_{=\boldsymbol{b}} = \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\left(\boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_\phi^T\boldsymbol{\lambda} - \boldsymbol{h}\right) \\ \Rightarrow & \left(\boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{J}_\phi^T\right)\boldsymbol{\lambda} = \boldsymbol{b} + \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\left(\boldsymbol{h} - \boldsymbol{S}\boldsymbol{\tau}\right). \end{aligned} \tag{8}$$

When ϕ's range (the task space of dimension k) is lower dimensional than its domain (the configuration space), and $\boldsymbol{J}_\phi$ is full rank, the matrix $\boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{J}_\phi^T \in \mathbb{R}^{k^2}$ is invertible and we can say

$$\boldsymbol{\lambda} = \underbrace{\left(\boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{J}_\phi^T\right)^{-1}\left(\boldsymbol{b} + \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\left(\boldsymbol{h} - \boldsymbol{S}\boldsymbol{\tau}\right)\right)}_{\zeta(\boldsymbol{\tau})}. \tag{9}$$

In other words, when there are more degrees of freedom in the robot than contact constraints in the environment (or at least as many), we can entirely characterize how applied forces $\boldsymbol{\tau}$ at the joints become contact forces $\boldsymbol{\lambda}$ exerted on the robot by the environment[7] at the contact points. If the robot exerts torques $\boldsymbol{\tau}$, it exerts forces $-\zeta(\boldsymbol{\tau})$ on the environment, and the environment in turn exerts forces $\zeta(\boldsymbol{\tau})$ back on the robot.

In a more general setting (without restricting the dimensionality of the task space that ϕ maps to), we can always write Equation 8 as

$$\underbrace{\begin{bmatrix} \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{S} & \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{J}_\phi^T \end{bmatrix}}_{\boldsymbol{C}} \begin{bmatrix} \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} = \boldsymbol{b} + \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{h}. \tag{10}$$

Assuming $\boldsymbol{C}$ is of (potentially reduced) rank r, and denoting the thin SVD of $\boldsymbol{C}$ as $\boldsymbol{C} = \boldsymbol{U}_{/\!/}\Sigma\boldsymbol{V}_{/\!/}^T$ with $\boldsymbol{U}_{/\!/} \in \mathbb{R}^{k\times r}$, $\Sigma \in \mathbb{R}^{r^2}$, and $\boldsymbol{V}_{/\!/} \in \mathbb{R}^{r\times(d+k)}$, the general solution of this equation is, for any coefficient vector $\boldsymbol{\beta}$,

$$\begin{bmatrix} \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} = \underbrace{\left(\boldsymbol{V}_{/\!/}\Sigma^{-1}\boldsymbol{U}_{/\!/}^T\right)}_{\boldsymbol{C}^\dagger \text{pseudo inv.}}(\boldsymbol{b} + \boldsymbol{J}_\phi\boldsymbol{M}^{-1}\boldsymbol{h}) + \boldsymbol{V}_\perp\boldsymbol{\beta}, \tag{11}$$

[7] Note that since λ arises from Lagrange multipliers which we can interpret them as forces generated by the constraints (see Udwadia & Kalaba (1996)). These forces, applied to the robotic system, keep its motion consistent with the constraints. Importantly, we need to be careful to remember that λ is *not* the force that the robot generates on the environment. It is the equal and opposite force that the environment exerts back on the robot.

where $\boldsymbol{V}_\perp$ an orthogonal matrix spanning the space orthogonal to $\boldsymbol{V}_{/\!/}$, i.e. the (right) null space of $\boldsymbol{C}$. Note that since $\boldsymbol{C}$ has $d+k$ columns and the rank of the matrix is r, this null space must be of dimension $d+k-r$. When $\boldsymbol{J}_\phi$ is of full rank $\boldsymbol{C}$ is of full rank k, and this null space is then of dimension $d+k-k=d$. In other words, when the forward kinematics map is full rank the space of valid combinations of applied torques $\boldsymbol{\tau} \in \mathbb{R}^d$ and resulting environmental reaction forces $\boldsymbol{\lambda} \in \mathbb{R}^k$ is just d-dimensional, the number of joints.

2.2.2 The full subspace of dynamic quantities

We can perform a similar analysis for the unsimplified floating-base contact constrained equations of motion given in Equation 7. Simply rearranging the equations, we can write them as

$$\underbrace{\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{S} & -\boldsymbol{J}_\phi^T \\ \boldsymbol{J}_\phi & & \end{bmatrix}}_{\boldsymbol{B}} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{h} \\ \boldsymbol{b} \end{bmatrix}. \tag{12}$$

Again, the space of feasible combinations of $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$ is given by the nominal pseudoinverse solution plus null space element:

$$\begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} = \underbrace{\overbrace{\boldsymbol{V}_{/\!/}\Sigma^{-1}\boldsymbol{U}_{/\!/}}^{\boldsymbol{B}^\dagger} \begin{bmatrix} -\boldsymbol{h} \\ \boldsymbol{b} \end{bmatrix}}_{\boldsymbol{v}} + \boldsymbol{V}_\perp \boldsymbol{\beta} \tag{13}$$

which we can easily compute using the SVD of $\boldsymbol{B} = \boldsymbol{U}_{/\!/}\Sigma\boldsymbol{V}_{/\!/}^T$. Again, $\boldsymbol{V}_\perp$ forms an orthogonal basis for the null space perpendicular to $\boldsymbol{V}_{/\!/}$. As indicated in the equation, this expression shows that the space of dynamic quantities $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$ is an affine space which can be described as $\boldsymbol{B}$'s null space offset by the nominal pseudoinverse solution $\boldsymbol{v}$.

The dimensionality of this space is the dimensionality of $\boldsymbol{B}$'s null space. Since $\ddot{\boldsymbol{q}} \in \mathbb{R}^{d+6}$, $\boldsymbol{\tau} \in \mathbb{R}^d$, and $\lambda \in \mathbb{R}^k$, the matrix $\boldsymbol{B}$ is a $(d+6)+k$ by $(d+6)+d+k$ matrix, so its null space, and hence the space of combined accelerations and dynamic quantities, is $2d+k+6-(d+k+6) = d$ dimensional (assuming full rank). And this result makes sense. There are only d avenues through which we can inject force into the system (d actuated joints), and the variations in both the accelerations $\ddot{\boldsymbol{q}}$ and reaction forces $\boldsymbol{\lambda}$ are direct consequences of those injected forces. The space spanning all combinations of accelerations, joint forces, and ground reaction forces must accordingly be d dimensional.

3 Cost function transfer between spaces

Section 2.2 demonstrated the space of dynamic quantities, both $\boldsymbol{\tau}$ and $\boldsymbol{\lambda}$ by themselves and those quantities in combination with the accelerations $\ddot{\boldsymbol{q}}$, are

affine. The floating-base constrained equations of motion define which combinations of those quantities are valid and which aren't. This allows us to effectively view the quantities $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$ as different (partial) observations of the same underlying phenomenon. One consequence of this perspective, which we explore here, is that defining quadratic cost functions over any or all of these dynamic quantities implies a combined quadratic cost function over the underlying fundamental space. And in turn we can observe how that underlying cost function manifests then on any one of the individual quantities $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, or $\boldsymbol{\lambda}$.

For instance, Equation 9 shows us that when the output dimensionality of ϕ is smaller than that of the configuration space, and when its Jacobian $\boldsymbol{J}_\phi$ is full rank, we can write the reaction forces entirely as a function of the applied torques $\boldsymbol{\lambda} = \zeta(\boldsymbol{\tau})$. Thus, we can entirely eliminate $\boldsymbol{\lambda}$ from the equations of motion by substituting this (linear) expression back into the first equation in Equation 7. Then solving for $\ddot{\boldsymbol{q}}$ we get (abstractly)

$$\ddot{\boldsymbol{q}} = \underbrace{\boldsymbol{M}^{-1}\left(\boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_\phi^T \zeta(\boldsymbol{\tau}) - \boldsymbol{h}\right)}_{\eta(\boldsymbol{\tau})}, \tag{14}$$

which now expresses $\ddot{\boldsymbol{q}} = \eta(\boldsymbol{\tau})$ entirely as a function of $\boldsymbol{\tau}$. Noting that $\eta(\boldsymbol{\tau})$ is a linear function of $\boldsymbol{\tau}$, we can now conclude that any quadratic function $Q(\ddot{\boldsymbol{q}})$ over accelerations is equivalently described as a function over $\boldsymbol{\tau}$ by *pulling it back* through the function $\ddot{\boldsymbol{q}} = \eta(\boldsymbol{\tau})$ to be

$$\widetilde{Q}(\boldsymbol{\tau}) = Q(\eta(\boldsymbol{\tau})). \tag{15}$$

We call the function $\widetilde{Q}$ the *pullback* of Q under mapping $\eta(\cdot)$ since we pull the function from the range space of η back to its domain space (simply using function composition). Note that since η in this case is linear and Q is a quadratic function over accelerations, the pullback function $\widetilde{Q}$ defined over torques is also a quadratic.

This phenomenon of quadratic functions over one variable transforming into quadratic functions over another variable connected to the first through linear constraints is very general, and widely applicable to the floating-base constrained equations of motion. The above discussion was just one example of how those transformations of quadratic functions might play out.

More generally, Equation 13 tells us we can describe the affine space of feasible dynamic quantities generally as

$$\begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} \quad = \quad \boldsymbol{v} + \boldsymbol{V}_\perp \boldsymbol{\beta} \quad = \quad \begin{bmatrix} \boldsymbol{v}^{\ddot{\boldsymbol{q}}} \\ \boldsymbol{v}^{\boldsymbol{\tau}} \\ \boldsymbol{v}^{\boldsymbol{\lambda}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{V}_\perp^{\ddot{\boldsymbol{q}}} \\ \boldsymbol{V}_\perp^{\boldsymbol{\tau}} \\ \boldsymbol{V}_\perp^{\boldsymbol{\lambda}} \end{bmatrix} \boldsymbol{\beta}, \tag{16}$$

where we've further decomposed $\boldsymbol{v} = [\boldsymbol{v}^{\ddot{\boldsymbol{q}}}; \boldsymbol{v}^{\boldsymbol{\tau}}; \boldsymbol{v}^{\boldsymbol{\lambda}}]$ and $\boldsymbol{V}_\perp = [\boldsymbol{V}_\perp^{\ddot{\boldsymbol{q}}}; \boldsymbol{V}_\perp^{\boldsymbol{\tau}}; \boldsymbol{V}_\perp^{\boldsymbol{\lambda}}]$ into its block components aligning with the individual variables $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$. More explicitly, this expression shows us that there is some fundamental d-dimensional space (the dimensionality of $\boldsymbol{B}'s$ null space), which we can represent as $\boldsymbol{\beta} \in \mathbb{R}^d$

so that every choice of $\boldsymbol{\beta}$ maps to a valid combination of dynamical quantities $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$ with the following relations:[8]

$$\begin{cases} \ddot{\boldsymbol{q}} = \boldsymbol{v}^{\ddot{\boldsymbol{q}}} + \boldsymbol{V}_{\perp}^{\ddot{\boldsymbol{q}}}\boldsymbol{\beta} \\ \boldsymbol{\tau} = \boldsymbol{v}^{\boldsymbol{\tau}} + \boldsymbol{V}_{\perp}^{\boldsymbol{\tau}}\boldsymbol{\beta} \\ \boldsymbol{\lambda} = \boldsymbol{v}^{\boldsymbol{\lambda}} + \boldsymbol{V}_{\perp}^{\boldsymbol{\lambda}}\boldsymbol{\beta} \end{cases} \tag{17}$$

Given the relations in Equation 17, we can now turn any quadratic function $Q(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda})$ over the individual dynamical variables into a function $\widetilde{Q}(\boldsymbol{\beta})$ over the canonical representation vector $\boldsymbol{\beta}$ using

$$Q(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda}) \quad \rightarrow \quad \widetilde{Q}(\boldsymbol{\beta}) = Q(\boldsymbol{v}^{\ddot{\boldsymbol{q}}} + \boldsymbol{V}_{\perp}^{\ddot{\boldsymbol{q}}}\boldsymbol{\beta},\ \boldsymbol{v}^{\boldsymbol{\tau}} + \boldsymbol{V}_{\perp}^{\boldsymbol{\tau}}\boldsymbol{\beta},\ \boldsymbol{v}^{\boldsymbol{\lambda}} + \boldsymbol{V}_{\perp}^{\boldsymbol{\lambda}}\boldsymbol{\beta}). \tag{18}$$

Furthermore, to then see how $\widetilde{Q}(\boldsymbol{\beta})$ manifests as a function over only a subset of those variables, say $\ddot{\boldsymbol{q}}$, we can use the inverse relation $\boldsymbol{\beta} = (\boldsymbol{V}_{\perp}^{\ddot{\boldsymbol{q}}})^{\dagger}(\ddot{\boldsymbol{q}} - \boldsymbol{v}^{\ddot{\boldsymbol{q}}})$ to create a quadratic function over only the quantity $\ddot{\boldsymbol{q}}$ of concern

$$Q^{\ddot{\boldsymbol{q}}}(\ddot{\boldsymbol{q}}) = \widetilde{Q}\left(\left(\boldsymbol{V}_{\perp}^{\ddot{\boldsymbol{q}}}\right)^{\dagger}(\ddot{\boldsymbol{q}} - \boldsymbol{v}^{\ddot{\boldsymbol{q}}})\right). \tag{19}$$

These observations simplify objective design for control. As we'll see, control algorithms center around defining and optimizing quadratic objective functions. This section demonstrates that we can define these objectives in terms of whatever combination of dynamic quantities $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and/or $\boldsymbol{\lambda}$ is most convenient, and we can straightforwardly use the geometry of the space of feasible combinations of these dynamic quantities to pull the objective into some canonical d-dimensional representation to ease optimization. (Generally, when we play with the linear algebra of a set of equations to pull them into a form that we can interpret as a cost function over just a subset of the variables, this is precisely what we're doing, although in a way specific to the particular problem.)

Section 4 explores these ideas further, and we see how controllers can leverage modern state-of-the-art optimization algorithms to handle not only these objectives but also linear equality and inequality constraints.

3.1 Case study: Controlling the center of mass

Consider a quadrupedal robot such as that shown in Figure 2. Each of the four legs $i = 1, \ldots, 4$ has d_l degrees of freedom. And suppose that each foot is well

[8] Note that although, we've described all of these quantities fairly abstractly in terms of the thin SVD $\boldsymbol{U}_{/\!/}\Sigma\boldsymbol{V}_{/\!/}^{T}$ of the matrix $\boldsymbol{B}$ defined in Equation 7, all modern matrix libraries, such as Eigen Jacob & Guennebaud (2011) or the basic linear algebra tools implemented in Lapack Lap (2011), readily compute the thin SVD of a matrix, so the direct implementation of these equations in real-world systems is feasible if desired. Often there are computationally more efficient methods for making these calculations, but at the expense of speed, the thin SVD provides a very robust and geometrically transparent tool that's often very useful for both prototyping and the final system.

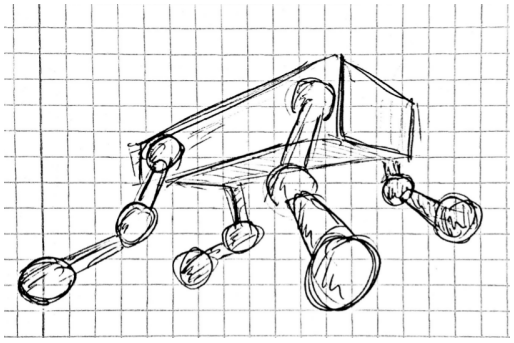

Figure 2: Quadrupedal robot.

approximated as a point contact.[9] (We'll discuss how different values of d_l lead to different properties in the control algorithms below.)

The torso of the robot is a single rigid component; we can define the base of the robot to be somewhere around the torso's center of mass. As discussed in Section 2, this base configuration has 6 degrees of freedom, 3 position components and 3 orientational components. It doesn't really matter where we place this base point, it just matters that we correctly represent the floating-base constrained equations of motion correctly with respect to that base. In fact, we'll want to control a slightly different point: the robot's center of mass. Effectively, the fixed base is just a set of numbers that help us correctly derive the equations of motion and ensure that they have the correct number of unactuated degrees of freedom. Other than that, it's location is somewhat arbitrary. Note that we can easily calculate the overall center of mass of the robot from the robot's current configuration since it's just a weighted sum of the centers of mass of each of the robot's constituent rigid body components.

Suppose we have some desired force $\widetilde{\boldsymbol{f}}_c \in \mathbb{R}^3$ and torque $\widetilde{\boldsymbol{\tau}}_c \in \mathbb{R}^3$ that we'd like to apply to the robot's center of mass point $\boldsymbol{x}_c \in \mathbb{R}^3$, for instance to move the robot into a configuration that frees one foot to take a step. We need to design a designing a controller that calculates the torque commands necessary to coax the environment into generating the requisite ground reaction forces $\boldsymbol{\lambda}_i$ for each foot i that cumulatively combine at the center of mass[10] to produce

[9] This assumption means that the robot is able to generate a linear reaction force from the ground that pushes back on it, but it isn't able to directly generate a torque. We can't use this assumption for humanoid robots (for instance), since their broad feet are designed to establish multiple points of contact with the ground and accordingly leverage friction to create torques on its center of mass. That additional analysis of foot contact makes humanoid control a bit more complicated than quadrupedal control with point contacts, but the underlying principles are similar (see Lee & Goswami (2012); Hyon et al. (2012); Herzog et al. (2013)).

[10] For any mechanical system of particles, the cumulative behavior of the external forces on the center of mass succinctly summarizes the system's behavior Halliday et al. (2001). For instance, suppose the system consists of n particles $\boldsymbol{x}_i \in \mathbb{R}^3$ with masses m_i, and suppose each particle experiences an external force $\boldsymbol{f}_i \in \mathbb{R}^3$. The center of mass is the weighted average

the desired $\widetilde{\boldsymbol{f}}_c$ and $\widetilde{\boldsymbol{\tau}}_c$.

The combined linear force at the center of mass is just the sum of external forces which is just a linear transformation of the individual forces at the feet

$$\boldsymbol{f}_c = \sum_{i=1}^{4} \boldsymbol{f}_i \quad = \quad \underbrace{\begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} \end{bmatrix}}_{\boldsymbol{G_f}} \underbrace{\begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \boldsymbol{f}_4 \end{bmatrix}}_{\boldsymbol{\lambda}}. \tag{20}$$

Similarly, since each contact force $\boldsymbol{f}_i$ contributes a torque around the center of mass of the form $(\boldsymbol{x}_i - \boldsymbol{x}_c) \times \boldsymbol{f}_i$, where $\boldsymbol{x}_i$ is the location of the ith foot, the overall mapping from contact forces to center of mass torques takes the following form

$$\boldsymbol{\tau}_c = \sum_{i=1}^{4} (\boldsymbol{x}_i - \boldsymbol{x}_c) \times \boldsymbol{f}_i \quad = \quad \underbrace{\begin{bmatrix} \mathcal{X}_1 & \mathcal{X}_2 & \mathcal{X}_3 & \mathcal{X}_4 \end{bmatrix}}_{\boldsymbol{G_\tau}} \underbrace{\begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \boldsymbol{f}_4 \end{bmatrix}}_{\boldsymbol{\lambda}}, \tag{21}$$

where we've used $\mathcal{X}_i \in \mathbb{R}^{3\times 3}$ to denoted the matrix that computes the indicated cross product $\mathcal{X}_i \boldsymbol{f}_i = (\boldsymbol{x}_i - \boldsymbol{x}_c) \times \boldsymbol{f}_i$. In other words, both the cumulative linear force and the cumulative torque acting on the center of mass $\boldsymbol{x}_c$ are both just linear transformations of the contact forces $\boldsymbol{f}_i$ generated at each foot. Specifically, $\boldsymbol{f}_c = \boldsymbol{G_f}\boldsymbol{\lambda}$ and $\boldsymbol{\tau}_c = \boldsymbol{G_\tau}\boldsymbol{\lambda}$.

Given desired forces $\widetilde{\boldsymbol{f}}_c \in \mathbb{R}^3$ and $\widetilde{\boldsymbol{\tau}}_c \in \mathbb{R}^3$ that we'd like the robot to apply at the center of mass, the control problem boils down to finding joint torques $\boldsymbol{\tau} \in \mathbb{R}^d$ that minimize the error

$$\frac{1}{2}\|\widetilde{\boldsymbol{f}}_c - \boldsymbol{G_f}\boldsymbol{\lambda}(\boldsymbol{\tau})\|^2 + \frac{1}{2}\|\widetilde{\boldsymbol{\tau}}_c - \boldsymbol{G_\tau}\boldsymbol{\lambda}(\boldsymbol{\tau})\|^2. \tag{22}$$

Here we denote the ground reaction forces $\boldsymbol{\lambda}$ as a function of the control torques $\boldsymbol{\tau}$ applied to the joints to emphasize that the only variables we really have control over are the torques $\boldsymbol{\tau}$; we are designing this controller so that the ground reaction forces $\boldsymbol{\lambda}$ the joint torques produce create a desired profile at the center of mass.

However, there are actually two issues still with this control methodology that we haven't yet addressed. First, $\boldsymbol{\lambda}(\boldsymbol{\tau})$ may not directly exist depending on how the number of actuated degrees of freedom (joints) d in the robot relates to the number of contact forces. If there are more joints than ground reaction

of particle locations $\boldsymbol{x}_c = \frac{1}{m_{\text{tot}}} \sum_{i=1}^{n} m_i \boldsymbol{x}_i$, where $m_{\text{tot}} = \sum_{i=1}^{n} m_i$. Since all internal forces cancel independent of how the particles interact with each other (they may be attached within a lattice or connected by strings or springs, or they may be a rigid body structure connect by joints, it doesn't matter), the only net force that remains is the cumulative force and torque produced at the center of mass by the external forces. Those quantities are $\boldsymbol{f}_c = \sum_i \boldsymbol{f}_i$ and $\boldsymbol{\tau}_c = \sum_i (\boldsymbol{x}_c - \boldsymbol{x}_i) \times \boldsymbol{f}_i$, respectively.

forces that we'd like them to produce, we have some redundancy. Multiple combinations of applied torques can produce the same ground reaction forces. Which do we choose?

Second, the mapping from ground reaction forces $\boldsymbol{\lambda}$ to center of mass forces $\boldsymbol{f}_c$ and torques $\boldsymbol{\tau}_c$ is highly redundant. Since the robot has four legs, if all four feet are in contact, we can produce 12 dimensions of ground reaction force (x, y, and z forces for each foot), but the linear transform described above that takes these forces to the center of mass only cares about 6 dimensions. Again, many different ground reaction forces (entire subspaces) result in the same cumulative center of mass force and torque. Which ground reaction forces are preferable?

What this means is the objective in Equation 22 doesn't fully constrain the problem and we need to address these additional modeling considerations before we have a fully specified controller.

3.1.1 Resolving ground reaction redundancy

We'll tackle the second concern first. To resolve ground reaction redundancy, we just need to construct additional objective terms $h_i(\boldsymbol{f}_i)$, one for each foot, modeling our prior biases on contact forces. We want these contact forces to be both generally small to reduce tension and, perhaps more importantly, we want to minimize the amount of tangential surface force the controller tries to generate so as to decrease the chance of slipping. If we construct quadratic (or linear) terms that implement these biases, adding them to Equation 22 still results in a quadratic objective so it doesn't fundamentally change the complexity of the controller.

Here we construct an example of one such redundancy resolving objective term. Denoting by $\boldsymbol{n}_i \in \mathbb{R}^3$ the surface normal at the ith foot contact point,[11] one such objective term could be

$$h_i(\boldsymbol{f}_i) = \frac{1}{2} \underbrace{\left\| [\boldsymbol{I} - \boldsymbol{n}_i \boldsymbol{n}_i^T] \boldsymbol{f}_i \right\|}_{\text{tang. comp. of } \boldsymbol{f}_i} + \frac{\alpha}{2} \underbrace{\left\| [\boldsymbol{n}_i \boldsymbol{n}_i^T] \boldsymbol{f}_i - \boldsymbol{a} \right\|^2}_{\text{orth. comp. of } \boldsymbol{f}_i}. \tag{23}$$

The first term penalizes the component of $\boldsymbol{f}_i$ tangent to the surface (perpendicular to the surface normal $\boldsymbol{n}_i$). This component can only be generated by friction. Unnecessarily large tangential components can easily lead to slipping, so we generally prefer these components to be as small as possible. The second term measures the component of $\boldsymbol{f}_i$ orthogonal to the surface. Note that we also have a constraint here that this component should really always be pointing up. Since the foot isn't actually *attached* to the ground, the ground can only generate upward reaction forces with positive inner product with $\boldsymbol{n}_i$. There are a number of ways to model that preference; the above term attempts to pull the orthogonal component of $\boldsymbol{f}_i$ toward some nominal value $\boldsymbol{a}$. Section 4 explores how we can more naturally explicitly model some of these criteria as constraints.

[11] A surface normal is a normalized vector perpendicular to the surface. By convention it points up and out of the surface.

3.1.2 Resolving torque space redundancy

Now, to address the first concern (torque redundancy), we first need to understand the nature of the redundancy. We can always remove the redundancy simply by adding a blanket term $\frac{1}{2}\|\boldsymbol{\tau}\|^2$ to the objective, but such a term is entirely indiscriminate in its penalization and will sometimes end up unduly fighting some of the desired behavior in an attempt to stamp out the redundancy. We can do better by analyzing the geometry of how torques relate to ground reaction forces as characterized by Equation 10.

We start by rewriting the equation as

$$\begin{bmatrix} \boldsymbol{\tau} \\ \boldsymbol{\lambda} \end{bmatrix} = \underbrace{\left(\boldsymbol{V}_{/\!/}\Sigma^{-1}\boldsymbol{U}_{/\!/}^T\right)}_{\boldsymbol{C}^{\dagger}\text{pseudo inv.}} \underbrace{(\boldsymbol{b} + \boldsymbol{J}_{\phi}\boldsymbol{M}^{-1}\boldsymbol{h})}_{\boldsymbol{a}} + \begin{bmatrix} \boldsymbol{V}_{\perp}^{\boldsymbol{\tau}}\boldsymbol{\beta} \\ \boldsymbol{V}_{\perp}^{\boldsymbol{\lambda}}\boldsymbol{\beta} \end{bmatrix}, \tag{24}$$

where we've decomposed $\boldsymbol{V}_{\perp}$ into separate blocks corresponding to the individual torque and lambda terms. From this perspective, it's clear that we can separate the equation into two parts $\boldsymbol{\tau} = \boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\beta})$ and $\boldsymbol{\lambda} = \boldsymbol{f}^{\boldsymbol{\lambda}}(\boldsymbol{\beta})$, both of which are linear functions of $\boldsymbol{\beta}$. Thus the following constrained optimization problem models the problem of finding the smallest torque $\boldsymbol{\tau}$ that's consistent with the desired ground reaction forces $\boldsymbol{\lambda}$:

$$\begin{aligned} \min_{\boldsymbol{\beta}} \; & \frac{1}{2}\overbrace{\|\boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\beta})\|^2}^{=\|\boldsymbol{\tau}\|^2} \\ \text{s.t.} \quad & \boldsymbol{\lambda} = \boldsymbol{f}^{\boldsymbol{\lambda}}(\boldsymbol{\beta}). \end{aligned} \tag{25}$$

Since both $\boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\beta})$ and $\boldsymbol{f}^{\boldsymbol{\lambda}}(\boldsymbol{\beta})$ are linear this entire problem is a linearly constrained quadratic optimization problem (see Appendix C for a brief discussion of Quadratic Programming). We can solve this constrained optimization problem analytically for the optimal solution to $\boldsymbol{\beta}$ as a linear function of $\boldsymbol{\lambda}$; plugging that $\boldsymbol{\beta}^*(\boldsymbol{\lambda})$ back into $\boldsymbol{\tau} = \boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\beta})$ gives what is effectively the *pseudoinverse* solution $\boldsymbol{\tau} = \boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\beta}^*(\boldsymbol{\lambda}))$ for the $\boldsymbol{\tau}$ that produces the desired $\boldsymbol{\lambda}$ while simultaneously minimizing the norm of $\boldsymbol{\tau}$ within the null space of the solution.

This discussion has remained very abstract so far because the actual algebraic solution to the above redundancy problem is less important than the idea of constructing a linearly constrained optimization problem to represent the redundancy resolution. Section 4 generalizes this approach by *starting* from an optimization centric perspective. Doing so enables us hand off much of the resulting linear algebra to a highly optimized Quadratic Program solver. Those solvers are often better than us anyway in finding an efficient subspace representations, and they're certainly less error prone.

Procedurally, the final controller for this problem solves the above described quadratic objective (or a similar one) at each iteration to give us an optimal set of torques $\boldsymbol{\tau}$ that, when applied, should produce the desired motion while respecting our redundancy resolution priors. There are a number of resources that describe this procedure in detail. See, for instance, Righetti et al. (2013).

4 Quadratic programming for control

Section 3.1 shows us how we might construct objective functions for control problems, but we saw some cases where quadratic objectives didn't seem sufficient, or at least seemed unnatural. In this section, we see how the above strategy is actually a special case of a much more general framework.

Appendix C gives a brief overview of Quadratic Programming along with some discussion of how we can solve these optimization problems quickly in practice. Here we simply assume that given a quadratic objective and some linear equality and inequality constraints, we have some fast black-box quadratic program solver that just magically returns the optimal solution to us. In practice, many such optimization tools exist (see, for instance, Perez et al. (2012); Grant & Boyd (2014); Nocedal & Wright (2006)), so this assumption is typically valid. In fact, real world control system typically use out-of-the-box optimizers for control. Solvers these days, for problems with around 20-30 variables can solve these optimization problems reliably within 1/1000 of a second (at 1kHz) which is sufficient for real-time control on modern torque controlled systems.[12]

4.1 Modeling control as a Quadratic Program

Section 3.1 constructed a quadratic objective that modeled the problem of finding torques $\boldsymbol{\tau}$ that produce desired forces and torques on the center of mass. Along the way, we found ourselves constructing objective terms over both $\boldsymbol{\lambda}$ and $\boldsymbol{\tau}$ and observing how the objective over both sets of variables, in conjunction with the constraints implicit in the equations of motion, interacted to imply a combined objective over just the space of $\boldsymbol{\tau}$. What we effectively did was create a large objective (over $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\ddot{\boldsymbol{q}}$), constrain it with linear equality constraints and did a lot of algebra to resolve the constraints. In the end, we found find that we could succinctly write the problem as a smaller objective over just a subset of the variables (the torques $\boldsymbol{\tau}$) and that made it nicely analytically solvable.

In days past, these nice analytical solutions were convenient because they usually just involved matrix manipulations and could be implemented on systems with critical real-time constraints. The algebraic manipulations described above were important to the success of the controller. These days, much of that's unnecessary. Computers are fast, and optimizers are fast. We have tools at our disposal that solve much of these manipulations automatically within the inner loop of an optimization routine that surpasses the limitations of basic linear algebra.

Given a Quadratic Program solver, we can write down our model of control

[12] Most solvers take advantage of warm starts, i.e. starting time step t's optimization from the solution found at time step $t-1$. Since solutions often don't shift too fast, this trick helps speed convergence improving real-time performance. They also may leverage some structure in the equations of motion to speed optimization.

succinctly as

$$\min_{\ddot{\boldsymbol{q}},\boldsymbol{\tau},\boldsymbol{\lambda}} \quad \psi(\ddot{\boldsymbol{q}},\boldsymbol{\tau},\boldsymbol{\lambda}) \tag{26}$$

$$\begin{aligned} \text{s.t.} \quad & \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{h} = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_\phi^T \boldsymbol{\lambda} \\ & \boldsymbol{J}_\phi \ddot{\boldsymbol{q}} = \boldsymbol{b} \\ & \text{[Other equality constraints]} \\ & \text{[Other inequality constraints].} \end{aligned} \tag{27}$$

This optimization problem simply jointly optimizes over over the dynamic variables, constraining them to be physically consistent using the floating based constrained equations of motion, and models other considerations, such as joint limits or constraints on the viable ground reaction forces due to the physics of frictional, using a combination of inequality and equality constraints. This control framework is a strict generalization of more traditional control techniques based on defining and optimizing quadratic objectives (and performing a lot of linear algebra by hand).[13]

Any quadratic objective used by itself to fully model control tasks can straightforwardly be used as the objective ψ in the Quadratic Program of Equation 26. But often, these free-standing objectives rely on some awkward approximations such as the objective terms in Equation 23 of Section 3.1 that model frictional preferences to avoid slipping. As we see next, there are easier ways to handle some of these issues when we have inequality constraints at our disposal.

4.2 Contact forces and friction cones

Friction acts tangentially to the surface at the point of contact (see Figure 3 left) proportionally to the normal component of the applied force (the harder the robot pushes against the surface, the more tangential frictional force the ground is able to exert back against the robot).

Consider the plane defined by the surface normal $\boldsymbol{n}$ and the desired reaction force $\boldsymbol{f}_i$. From the perspective of that plane, we can decompose the force $\boldsymbol{f}_i$ into its scalar valued tangential and normal components (see Figure 3 right). Denoting the tangential component by $\alpha_{/\!/}$ and the normal component by $\alpha_\perp$, the physical frictional constraints on the reaction forces that this surface can generate (assuming Coulomb's law Halliday et al. (2001)) are

$$\alpha_{/\!/} \quad \leq \quad \mu\alpha_\perp, \tag{28}$$

where μ is the surface's coefficient of friction. The tangential component cannot be more than some bounding force that scales with $\alpha_\perp$ defined by how hard the robot's pushing against the ground.

[13] Note that because of real time constraints, it may be prudent to perform some manipulation of the equations in the Quadratic Programming problem to ensure that it's as computationally efficient as possible, but in principle, especially in the near future as computers become faster, we can simply apply efficient optimizers out-of-the-box.

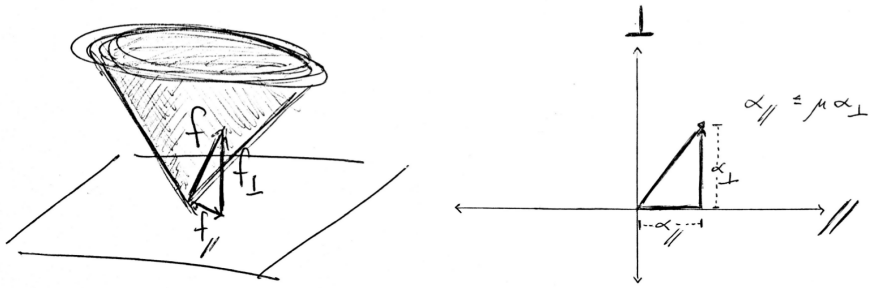

Figure 3: **Left:** Friction cone. Valid ground reaction forces $\boldsymbol{f}$ consist of two components, an orthogonal component $\boldsymbol{f}_\perp$ telling how hard the robot is pressing against the ground, and a parallel component $\boldsymbol{f}_{/\!/}$ generated by friction. **Right:** The magnitude of the frictional component $\alpha_{/\!/}$ cannot be larger than the largest possible value that can be generated by friction, which is proportional to the perpendicular force $\alpha_\perp$.

Since this constraint applies symmetrically around the surface normal, we can visualize it as a cone around $\boldsymbol{n}$. For any strength of the perpendicular force $\|(\boldsymbol{n}\boldsymbol{n}^T)\boldsymbol{f}_i\|$, the generated tangential reaction force norm $\|(\boldsymbol{I}-\boldsymbol{n}\boldsymbol{n}^T)\boldsymbol{f}_i\|$ cannot be bigger $\mu\|(\boldsymbol{n}\boldsymbol{n}^T)\boldsymbol{f}_i\|$. But we always know that if a give $\boldsymbol{f}_i$ is valid (in the sense that friction can generate it's tangential component) any desired force with the same tangential component but stronger normal component always works as well. Thus, the set of feasible frictional forces is a cone as depicted in Figure 3.

Fundamentally, this feasible set of achievable ground reaction forces is best described by an inequality constraint. Approximating the set using a quadratic potential as we did above is a rather poor approximation. At face value, Quadratic Program solvers don't handle conic constraints, but they do natural handle linear inequality constraints. Thus, if we approximate the circularly symmetric shape of the cone around the $\boldsymbol{n}$ by a polyhedron, we can well approximate this friction cone by a collection of linear inequality constraints.[14]

Additionally, we typically have an inequality constraint of the form form $\boldsymbol{n}^T\boldsymbol{f} \geq 0$ representing the fact that the robot isn't attached to the ground; it can only induce reaction forces by *pushing* against the ground. But notice that this inequality is again linear, so it, too, fits nicely within the Quadratic Programming framework.

Thus, designing a controller around a Quadratic Program solver allows us to much more explicitly represent these conic frictional constraints. In general, inequality constraints give us much more modeling flexibility than quadratic functions and linear equality constraints alone.

[14] A polyhedron is the intersection of a collection of half planes, each of which we can describe using a linear inequality.

A A note on fictitious forces

Equation 1 gives the basic fully actuated unconstrained equations of motion for a rigid body system. The equation is similar to Newton's $\boldsymbol{f} = m\ddot{\boldsymbol{q}}$, but where does $\boldsymbol{h}$ come from? The equations of motion are usually derived from the Lagrangian formulation of classical mechanics which allows us to describe physics from the perspective arbitrary accelerating frames of reference, not just inertial frames. And the behavior of physics from that perspective is sometimes less intuitive.

One such non-inertial frame is the space of joint angles of a robot— joint angles describe rotating frames where straight line through the joint space become accelerated curves through the Cartesian space. For simplicity, consider just a very simple robot with one link attached to a revolute joint at one end. The configuration of the link is fully determined by the angle of the joint; in that space of joint angles, increasing the value at a constant rate (straight line constant velocity motion through this one-dimensional space), generates a constant rotation of the link. From the perspective of a single point on the link, though, it's moving in a perfect circle around the axis. To maintain that circle, the point must be constantly accelerating toward the axis, which means that the internal lattice structure of the link's constituent particles must be colluding to generate a force on that particle to pull it inward.

From the perspective of the point, which sits on the link thinking that it's moving in a constant velocity through its joint space, it feels a force pulling it outward because it needs to hold on tight to stay where it is. We know that this is simply its urge to be moving in a straight line through the Cartesian space, but from the perspective of its coordinate system, that urge manifests as a force pulling it away from the axis. In general, this sort of force, called a *centrifugal* force, arises from the fact that for a given configuration $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$ the coordinate system is accelerating. This type of *fictitious* force is in part what we typically fold into the term $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

Another common phenomenon arises within this rotating coordinate system. How would a particle riding along the link describe the motion of a particle that's detached from the frame of reference and moving in a straight line through the Cartesian space? From the perspective of the link, who's straight line motion is actually a curve through Cartesian space, the motion of the external particle looks like a curve. Even external particles at rest relative to some fixed world coordinate frame[15] look like they're moving in a curve with respect to the rotating reference frame. And in order to explain this curved motion within the rotating frame, we must account these accelerations, or equivalently these perceived fictitious forces, again as a function of the configuration $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$. These fictitious forces, called *Coriolis* forces, are also typically part of the term $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

In general, fictitious forces that arise from non-inertial reference frames can be arbitrary in nature. The centrifugal and Coriolis forces described here are just

[15] Technically, all non-accelerating particles are at rest with respect to some inertial reference frame.

two common manifestations. Symbolically, though, we usually just denote them abstractly as $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ since these forces may be a function of the configuration $\boldsymbol{q}$ and the velocity $\dot{\boldsymbol{q}}$, but they are never a function of accelerations $\ddot{\boldsymbol{q}}$. Thus, if we fix $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ at any moment in time and analyze how forces relate to accelerations in the system, this vector of values simply becomes a constant offset.

B The mathematical abstraction of torques on joint angles

In this section, we want to emphasize that applied torques on joint are themselves actually mathematical abstractions. The actuators physically connecting two links to one another may, for instance, be hydraulic pistons. In this case, we need a very low-level controller that takes a desired torque value τ_d and chooses commands for the actuator itself to create the desired force value as measured at the joint in question. For hydraulics we need to control the valve on the piston to regulate the amount of pressurized hydraulic fluid is allowed to feed into the chamber from the action of the pump. Roughly, if the generated torque is currently too large, we close off the valve a bit, and if we aren't generating enough torque, we open it up. Abstractly at the level of control we describe in this document, we are calculating only the vector of desired torques $\boldsymbol{\tau}_d$. Those torques still need to be translated into commands to the actuators themselves through a lower layer of control. So although we typically design controllers with the assumption that we can directly command these torque values, we can do that only because we're able to measure and control these values at a lower level of control to implement this layer of mathematical abstraction on the physical machine.

C Quadratic Programming

A general constrained optimization problem takes the form

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \psi(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, \ldots, k \\ & h_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, \ldots, l \end{aligned}$$

for some k equality constraint functions g_i and l inequality constraint functions h_j. When the objective function $\psi(\boldsymbol{x})$ is quadratic and each constraint function g_i and h_j is *linear*, this optimization problem becomes much easier to solve. Such a problem is called a Quadratic Program; modern off-the-shelf solvers for these problems are very fast.

We can briefly get a flavor for some of the fastest of these solvers, by examining the linear *equality* constraints first. Each linear equality constraint g_i defines a linear subspace of feasible solutions, we can actually algebraically (and computationally) define how $\psi(\boldsymbol{x})$ looks when restricted to only that subspace.

Solving the equality constrained problem (see the Method of Lagrange Multipliers Wikipedia (2002-2014), for instance) is therefore easy. We can even solve it analytically— no optimizer necessary.

But what we find when we look at the theoretical optimality conditions under added *inequality* constraints h_i is that at the optimizer, each inequality constraint is either *active*, in which case the optimal sits on the constraint with equality $\boldsymbol{h}_i(\boldsymbol{x}^*) = 0$, or the constraint is *inactive*, in which case the optimal point doesn't care about it at all so we could have just as easily thrown it away.

Intuitively, we can think of the constraint as defining a flat surface (such as a tabletop). An inequality constraint effectively says that we can only be on one side of the surface (say above the table, for instance). In our search for the minimum, the gradient toward the minimum is either going to be pressing us into the table (because the unconstrained minimum is below the table), in which case the optimal point will rest explicitly on the table, or the gradient will be pushing the optimizer away from the table since the unconstrained minimum is already somewhere above it, in which case our search for the minimum won't even care about the table. The inequality constraint is either an equality constraint at the optimizer, or the constraint doesn't actually constrain the solution, so we could have just as easily thrown it away.

What this means algorithmically, is that we can implement a solver that tries to figure out which of these inequality constraints are on and which are off. The set of inequalities that are on is called the *active set*, and if our active set prediction is correct, the optimal solution with lie on the surface of these constraints with equality. So until we've found the right active set, given a hypothesized active set we can just solve the associated equality constrained problem and update the set based on what the constraints or gradients look like a the minimizer. If new constraints become violated, we can add those to the active set and try again. If the gradient of the objective is trying to push into the feasible region of one of the constraints already in the active set, then we can remove that constraint from the active set because we'd rather be in its interior. Active set Quadratic Program solvers are common for mid-sized problems in systems with real time constraints, such as robotic control systems (see, for instance, Herzog et al. (2013)).

We've given a heuristic, intuitive, overview of these problems and one class of algorithms for solving them, but the literature in this area is extensive and a lot of work has gone into refining the algorithms into very efficient and easy-to-use software packages. The books Nocedal & Wright (2006) and Boyd & Vandenberghe (2004) both have great discussions on the geometry of these problems and the state-of-the-art algorithms used to solve them in practice.

References

Lapack 3.4: Linear algebra package, 2011. URL `http://www.netlib.org/lapack`.

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization*. Cambridge University Press, 2004.

Grant, Michael and Boyd, Stephen. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, March 2014.

Halliday, David, Resnick, Robert, and Krane, Kenneth S. *Physics, Volume I.* Wiley, 5th edition, 2001.

Herzog, Alexander, Righetti, Ludovic, Grimminger, Felix, Pastor, Peter, and Schaal, Stefan. Experiments with a hierarchical inverse dynamics controller on a torque-controlled humanoid. Technical report, arXiv:1305.2042 [cs.RO], Cornell University Library, 2013.

Hyon, Sang-Ho, Hale, Joshua G., and Cheng, Gordon. Full-body compliant human-humanoid interaction: Balancing in the presence of unknown external forces. *IEEE Transactions on Robotics*, 23(5), 2012.

Jacob, Benot and Guennebaud, Gal. Eigen 3: C++ library for linear algebra, 2011. URL http://eigen.tuxfamily.org.

Lee, Sung-Hee and Goswami, Ambarish. A momentum-based balance controller for humanoid robots on non-level and non-stationary ground. *Autonomous Robots*, 2012.

Nocedal, Jorge and Wright, Stephen. *Numerical Optimization*. Springer, 2006.

Perez, Ruben E., Jansen, Peter W., and Martins, Joaquim R. R. A. pyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization*, 45(1):101–118, 2012. doi: 10.1007/s00158-011-0666-3.

Ratliff, Nathan. Analytical dynamics and contact analysis, 2014. Lecture notes.

Righetti, L., Buchli, J., Mistry, M., Kalakrishnan, M., and Schaal, S. Optimal distribution of contact forces with inverse dynamics control. *International Journal of Robotics Research*, pp. 280–298, 2013. URL http://www-clmc.usc.edu/publications//R/Righetti_IJORR_2013.pdf.

Siciliano, Bruno, Sciavicco, Lorenzo, Villani, Luigi, and Oriolo, Giuseppe. *Robotics: Modelling, Planning and Control*. Springer, second edition, 2010.

Udwadia, Firdaus E. and Kalaba, Robert E. *Analytical Dynamics: A New Approach*. Cambridge University Press, 1996.

Wikipedia. Lagrange multiplier, 2002-2014. URL http://en.wikipedia.org/wiki/Lagrange_multiplier.