Statics, Dynamics and Control of Robots: a Quick Overview

Wyatt Newman

Oct 29, 2013

Jacobians and Statics:

An energy argument was introduced previously to analyze gravity loads using Jacobians. A similar energy argument leads to useful expressions for controling interaction forces between a robot and the environment.

Defining a coordinate frame on an end effector (e.g. on the palm of a hand), one can refer to the position and orientation of the end effector in the environment, typically with respect to a task to be performed. Position and orientation can be expressed with 6 quantities—although these six quantities do not constitute a vector and do not conform to the rules of linear algebra. However, the the time derivative of these quantities—the hand translational and rotational velocity components—do constitute a vector. This vector is often referred to as a "twist" where:

Eqn 1: **t** = [v_x; v_y; v_z; omega_x; omega_y; omega_z] (a column vector)

The counterpart to a twist is a "wrench", **w**, comprised of a force vector and a moment vector:

Eqn 2: **w** = [f_x, f_y, f_z, m_x, m_y, m_z]

Since a wrench is composed of forces and moments, these are collectively referred to as "efforts." One ambiguity that needs to be clarified is whether the wrench corresponds to an effort exerted by the environment on the hand (e.g., the weight of an object being carried, a handshake, a reaction from a power tool, etc.) or the effort exerted by the robot on the environment (e.g. to perform scraping, sliding, pushing, etc).

Define a wrench, **w**_onEnv, exerted *by* the hand *on* the environment. If this effort is with respect to the hand frame moving with twist **t**, then the robot is performing work on the environment at a rate (power) equal to: P_out = **t** (dot) **w**_onEnv (a scalar, measured in Watts).

The work done on the environment must come from the robot. If the robot is moving slowly (and thus kinetic energy is negligible) and if, for the momen, we ignore gravity (e.g., if an arm is moving horizontally), then the work performed on the environment must balance the work injected by the motors.

The robot's joint angles are the vector **q**, and the joint velocities are **q_dot**. The corresponding joint efforts (torques) are the vector **Q**. The power injected into the system by the motors is **Q** (dot) **q_dot**. Balancing input and output power yields:

Eqn 3: $\mathbf{Q}^T\ \mathbf{q_dot} = \mathbf{t}^T\ \mathbf{w}$_onEnv

Using the Jacobian from joint angles to hand frame, we have that:

Eqn 4: **t = J q_dot**

and thus:

Eqn 5: $\mathbf{q_dot}^T\mathbf{Q} = (\mathbf{J\ q_dot})^T\mathbf{w}$_onEnv = $\mathbf{q_dot}^T\ \mathbf{J}^T\ \mathbf{w}$_onEnv

Since Eqn 5 must apply for any q_dot, we can deduce that:

Eqn 6: $\mathbf{Q} = \mathbf{J}^T \ \mathbf{w}\_onEnv$

This is the statics relationship for robots. Given a desired wrench to be exerted on the environment, we can compute the corresponding set of joint torques that is statically consistent with that effort. Note that this statement falls short of guaranteeing that we can exert any desired wrench on the environment. For example, if our robot had only one degree of freedom, we could clearly not satisfy achieving 6 target wrench values. But we could compute what single joint effort would result in static equilibrium with the proposed wrench.

The above analysis ingnored gravity. If the robot motion resulted in changes in elevation of masses, then there would be a power term associated with energy storage (or retrieval) from potential energy. However, this energy analysis is already covered by the previous gravity loading arguments. Given that we can compute the joint efforts required to oppose gravity, we can simply superimpose these efforts with the statics equation, resulting in a control policy of:

Eqn 7: $\mathbf{Q} = \mathbf{J}^T \ \mathbf{w}\_onEnv + \ \mathbf{Q}\_grav$

The above policy can often be used as-is to produced desired contact efforts in performing a task.

Gravity Loads and Stance:
We can use the statics analysis to compute required joint efforts to oppose gravity while standing. Initially, consider standing on a single foot (so as to avoid the ambiguity of how to apportion forces/torques between two feet on the ground). It is useful to compute a Jacobian with respect to the stance foot (in contrast to Jacobians computed with respect to the pelvis frame). If we compute a center-of-mass Jacobian with respect to the stance foot, then for total mass M we can say:

Eqn 8: $\mathbf{Q} = \mathbf{J}^T\_com/\mathbf{f} * M\mathbf{g}$

where $\mathbf{J}\_com$ is the Jacobian of the center of mass with respect to the stance foot.

For 2 feet, additional arguments can be invoked to apportion the gravity loading between the feet while minimizing internal stresses.

Jacobians and Null Space:
In general, a (non-zero) vector **n** lies in the "null space" of a matrix **A** if:

Eqn 9: $\mathbf{0} = \mathbf{A}*\mathbf{n}$

If A is square and invertible, then there are no vectors that satisfy this relationship, i.e. the null space of **A** is empty. If a matrix has more columns than rows (as is typical of our Jacobians), then there will be a null space. The null space typically would have dimension d_null = ncols – nrows, where ncols is the number of columns and nrows is the number of rows. A Jacobian can additionally lose rank (rank<nrows) in certain poses (e.g., an elbow or knee fully extended results in the corresponding Jacobian losing rank). In that case, the null space dimension increases.

For a null space of dimension d_null, there are "d_null" number of linearly independent vectors that lie in the null space. For a desired (typically, 6x1) target twist vector, if the Jacobian has more columns than rows, then one can find a nominal solution for q_dot via the pseudo inverse. However, one can also add any weighted sum of null-space q_dot vectors to this solution and still satisfy the desired twist. One of the uses for adding in a combination of null-space q-dot vectors is to try to avoid violating joint limits.

Additional null-space options can open up for tasks that do not require full specification of all 6 components of a twist vector. For example, if it is desired to pick up a cylinder, the rotation of the cylinder about its axis in the palm of the hand is irrelevant. Equivalently, there is an axis, **a**, about which hand-frame rotation is unconstrained. This allows a joint-space solution that adds to the null space. If a twist is specified a [0;0;0;a_x,a_y,a_z]' = **t**', then one can solve for a nominal q-dot for which **t** = **J q_dot.** In this case, the nominal solution for q_dot corresponds to a rotation of the palm about the cylinder's axis—a motion that is unconstrained. As a result, the solution q_dot for this target twist is another null-space vector that may be scaled and added to any nominal solution.

The Angular Jacobian and Target Orientations:
The angular Jacobian is rows 4 through 6 of the 6xn Jacobian matrix, relating joint velocities to a rotation vector of the output of interest (e.g., a hand frame). The translational Jacobian (first three rows) can be used to guide an origin of interest to a target location, using, e.g., the pseudo-inverse. An orientation goal can also be achieved using the angular Jacobian terms.

For the translational terms, one can define a direction of approach as (**p**_goal – **p**_start)/||**p**_goal-**p**_start||, where **p**_goal is a goal position and **p**_start is the current (e.g. hand) position. Using this direction vector, one can find joint velocities, q_dot, to move the hand along this vector.

To include consideration of orientation, there are multiple options. One can first orient the hand as desired for approach, then require that the hand orientation not change during approach. Equivalently, this sets omega_des to [0;0;0], completing the 6x1 twist specification.

If the hand orientation is not the desired orientation, then one can use Jacobians to converge on the desired orientation. Say the desired orientation is **R**_goal and the current orientation is **R**_start. The required rotation to get from **R**_start to **R**_goal is **R**_g/s, where **R**_goal = **R**_g/s **R**_start. It follows that **R**_g/s = **R'**_start **R**_goal, since for rotation matrices, the transpose is the same as the inverse. Given that we desire to rotate the hand frame by rotation operator **R**_g/s, we can transform this into an axis/angle representation. (See, e.g., http://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation for how to compute this transformation). In the axis/angle representation, the desired rotation is expressed as a rotation angle, theta, about a single axis, **a**. We may specify that the hand rotate with omega (vector) proportional to **a.** The magnitude may be selected to something of a comfortable speed, and the resulting 3 components may be specified as values 4 through 6 of the target twist vector. Using this target, the hand will both move towards the goal location while rotating towards the goal orientation. The translational velocity and/or rotational velocity should be prescribed to taper to zero as the hand converges on the target position and/or orientation, respectively.

Control of 2nd-Order Systems:
If we have a second-order system—e.g. a pure mass—on which we can exert an effort: f = m*d/dt(v). We can choose to define a controller based on a desired postion, x_des, and desired velocity, v_des, as a linear combination of state errors:
Eqn 10: f = Kp*(x_des – x) + Kv*(v_des-v)

The LaPlace transform of the resulting differential equation is:

Eqn 11: m*s^2 (x) = Kv*v_des + Kp*x_des -s*Kv*x – Kp*x
or:
Ean 12: [s^2 + (Kv/m)s + (Kp/m)] x = [(Kv/m)s + (Kp/m)] x_des

which defines a closed-loop transfer function:
Eqn 13: x/x_des = [(Kv/m)s + (Kp/m)] /[s^2 + (Kv/m)s + (Kp/m)]

We can consider the response to sinusoids by substituting j*omega for s, yielding:

Eqn 14: x/x_des = [(Kv/m)*j*omega+ (Kp/m)] /[-omega^2+ (Kv/m)*j*omega + (Kp/m)]

At low frequency (omega<<sqrt(Kp/m)) we can approximate the response as unity, i.e. perfect following of the input command. As the frequency increases, the term omega^2 starts to become significant relative to Kp/m. Near this frequency (and higher) the transfer function is no longer unity and position-command following begins to fail.

We define (approximately) the closed-loop following bandwidth as omega = omega_n = sqrt(Kp/m). Roughly, below this frequency, we get good following, and above this frequency, we get poor tracking. It is thus advantageous to make Kp as large as possible. At the same time, Kv should be adjusted to avoid excessive ringing. Typically, Kv is set such that Kv/m = 2*omega_n (for critical damping).

Although large values of Kp are desirable, excessively large values of Kp will result in poor performance due to effects such as:

- sampling rates
- latency
- sensor noise
- resonances
- other unmodeled dynamics

In practice, one often finds the maximum practical value of Kp experimentally (adjusting Kv accordingly).

Robot Dynamics and Control Decoupling:
Deriving robot dynamics can be tedious. Derivations are presented rigorously in many texts. Ultimately, though, the result comes down to a set of (vector) differential equations of the form:

Eqn 15: **Q** = **H**(**q**) **q**_ddot + **C**(**q**,**q**dot)***q**_dot + **Q**_grav(**q**)

The terms include gravity torques (previously derived), centrifugal and Coriolis terms (from the C matrix) and inertial terms (from the matrix inertia tensor **H**). The Coriolis and centrifugal terms grow as the square of joint velocities. For slow motions, these terms can be ignored. (At higher speeds, these terms are important—as evidenced in acrobatics, gymnastics, diving and figure skating).

Part of a recommended (nonlinear) control policy is to compute the gravity torque estimates and add these to the vector of joint torque commands. This approximately eliminates the gravity influence. The resulting system, after this compensation, is:

Eqn 16**: Q** = **H**(**q**) **q**_ddot

Where **Q** will be motor torque commands *in addition to* gravity influence compensation, q_ddot is the second time derivative (double-dot) of joint angles (i.e. joint accelerations), and H(q) is the matrix inertia tensor. This term is analogous to “m” in the expression: f = m*a. However, the off-diagonal terms of H result in coupling between joints. This cross-coupling is important even at low velocities.

A control approach that accounts for the cross coupling is the following. Compute and estimate for all terms of H(q) (as a function of robot pose). Define a vector of desirable joint accelerations, q_ddot. Exert joint torques (added to gravity compensation) equal to H_estimate*q_ddot_des.

q_ddot_des may be defined based on an analogy of a 2nd order system. Each term of this vector may be chosen to satisfy:
q_ddot_des = 2*omega_n*(q_dot_des – q_dot) + omega_n^2*(q_des – q)

The value of omega_n is chosen to keep the speed of response within the system limits (resonances, sampling rates, latencies, unmodeled dynamics). But the resulting control efforts are modulated by:

Q = **H**(**q**)[**K**v(**q**_dot_des – **q**_dot) + **K**p(**q**_des – **q**)]

The equivalent gains are H*Kp and H*Kv, which are modulated as a function of robot pose, and which include cross-coupling compensation.