Advanced Topic

ARTICULATED BODY DYNAMICS

Tim Johnson & Michael Su
Agenda

- Inverse Dynamics in general
- Efficiency
- A power tool for designing algorithms – Recursion
- Recursive Newton-Euler Algorithm
- Forward Dynamics in general
- Featherstone’s Algorithm
- Conclusion
- Reference
Inverse Dynamics (1)

- Given the kinematic representation of motion, inverse dynamics calculates the forces necessary to achieve that motion.
- Inverse kinematics will tell us what the motion is.
- Inverse dynamics will tell us how to do it.
Inverse Dynamics (2)

- The calculation of the forces required at a robot's joints to produce a given set of joint accelerations

- Applications
  - Robot control
  - Trajectory planning

- Rapid execution is essential as it is used heavily for real-time control
The classic approach to inverse dynamics involved a Lagrangian formulation, which was $O(n^4)$

Non-recursive approaches of either Lagrangian or Newton-Euler formulations result in equations such as:

\[
Q_i = \sum_{j=1}^{n} H_{ij} \ddot{q}_j + \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk} \dot{q}_j \dot{q}_k + g_i
\]

$O(n^2)$  \hspace{1cm} $O(n^3)$
Efficiency (2)

- Optimization technique: use recursion
  - Requires a reformulation of the equations
  - Can reduce complexity down to $O(n)$
  - Reduces computational requirement as well
Recurrence Relations

- Equations defining a member of a sequence in terms of its predecessors
- Example
  - $x_{n+1} = x_n + x_{n-1}$
  - $x_0 = x_1 = 1$
  - Fibonacci sequence
Example: Recurrence Relation

- Let the matrix $B$ be defined as
  - $B = A_1 A_2 \ldots A_n$, where $A_i$ is a matrix
- How do we compute the derivative of $B$
  - Brute force
    - Gets expensive fast
  - Use recursion
Example: Brute Force

- Say \( B = A_1 A_2 A_3 A_4 A_5 \)
- Then \( B' = \)
  \[
  A_1'A_2 A_3 A_4 A_5 + \\
  A_1 A_2' A_3 A_4 A_5 + \\
  A_1 A_2 A_3' A_4 A_5 + \\
  A_1 A_2 A_3 A_4' A_5 + \\
  A_1 A_2 A_3 A_4 A_5'
  \]

- Computational requirement is \( n^2-n \) matrix multiplications and \( n-1 \) matrix additions
Example: Recursion

- Recall $B = A_1 A_2 ... A_n$
- Define $B_{i+1} = B_i A_{i+1}$
  - It follows that $B_n = B$, thus $B_n' = B'$
- So
  
  $$B_{i+1}' = B_i' A_{i+1} + B_i A_{i+1}'$$
- To calculate $B'$, we start with $B_1' = A_1'$ and iterate up to $B_n$
Example: Recursion

- So we iteratively compute $B_2, \ldots, B_{n-1}$
  - n-2 iterations
  - 1 matrix multiplication
- Then we iteratively compute $B_2', \ldots, B_n'$
  - n-1 iterations
  - 2 matrix multiplications, 1 matrix addition
- Total: $3n-4$ matrix multiplications, $n-1$ additions
  - Non-recursive: $n^2-n$ matrix multiplications and $n-1$ additions
Example: Results

- By using recurrence relations, we were able to reformulate the solution using recursion
- $O(n^2)$ to $O(n)$ improvement
- We can get much more dramatic results with inverse dynamics
  - $O(n^4)$ to $O(n)$
Robot System Model

- $N$ movable links, labeled 1,...,N (from the root to the terminal)
- A fixed base link, labeled 0
- $\lambda(i)$ is the parent link of link $i$
  - Links are numbered so that $\lambda(i) < i$
- $N$ joints, where joint $i$ connects link $\lambda(i)$ to link $i$
Recursive Newton-Euler Algorithm (RNEA)

- The most efficient currently known general method for calculating inverse dynamics
- Input: a system model of a robot and the values of the desired joint accelerations
- Output: the joint forces required to produce the desired joint accelerations
RNEA Steps

- Step 1: calculate the velocity and acceleration of each link $i$
- Step 2: calculate the net force acting on each link from its motion and inertia
- Step 3: calculate the joint forces required to produce the forces in step 2
RNEA

Compute velocity, acceleration of links

Compute net forces

Compute joint forces
Step 1: Link Motion

- $v_i$ – absolute velocity of link $i$
- $a_i$ – absolute acceleration of link $i$
- $s_i \dot{q}_i$ – velocity across link $i$ (relative velocity of link $i$ with respect to link $i-1$
- Recurrence relation:
  - $v_i = v_{i-1} + s_i \dot{q}_i$  \hspace{1cm} (v_0 = 0)
  - $a_i = a_{i-1} + v_i \times s_i \dot{q}_i + s_i \ddot{q}_i$  \hspace{1cm} (a_0 = 0)
Step 2: Net Forces on the Link

- The net force on link $i$ is given by the link's rate of change of momentum

\[ f_i^* = \frac{d(I_iV_i)}{dt} = I_i a_i + \nu_i \times I_i \nu_i \]
Step 3: Joint Forces

- First we must find the total force transmitted from link $i-1$ to link $i$ through joint $i$.

- Rearranging, we get

\[ f_i = f_{i+1} + f_i^* \]

\[ (f_n = f_n^*) \]
Taking into Account External Forces…

- The equation for joint force becomes
  \[ f_i = f_{i+1} + f_i^* - f_i^x \]
- If you want to model gravity, you can apply a gravitational force to each joint
- It is more efficient, however, to give the robot's base an acceleration of \(-g\) (\(a_0 = -g\))
RNEA: Pseudocode

\[ v_0 = 0 \]
\[ a_0 = 0 \]
for \( i = 1 \) to \( N \) do
\[ v_i = v_{\lambda(i)} + s_i q_i' \]
\[ a_i = a_{\lambda(i)} + v_i x s_i q_i' + s_i q_i'' \]
end

Step 1

\[ f_i^* = I_i a_i + v_i x I_i v_i \]
end

Step 2

\[ f_n = f_n^* \]
for \( i = N \) to \( 1 \) do
\[ F_{\lambda(i)} = f_i + f_{\lambda(i)}^* - f_i^x \]
end

Step 3
Forward Dynamics

- Primarily for simulation, not necessary to meet real-time speed requirement
- A more difficult problem to solve than inverse dynamics
- Two approaches:
  - Solve the problem directly by calculating recursion coefficients. Ex: Featherstone’s Algorithm  
  \[
  \ddot{q} = M^{-1} F 
  \]
  - Obtain and then solve a set of simultaneous equations. Ex: Composite Rigid Body Algorithm
Featherstone’s Algorithm: Basic Idea
The Characteristics of Featherstone's Algorithm

- Also called “Articulated-Body Algorithm”
- Developed for solving forward dynamics problems
- First version only worked for joints with single degree-of-freedom.
- Second version included a general joint model and was faster.
- Complexity: $O(n)$, faster than CRBA for $N>9$
Featherstone’s Algorithm

- Linear relation between the acceleration and the force (Newton Euler equation):
  \[ f = I^A a + p \]
  - Articulated-body inertia
  - Link acceleration
  - Bias force (the value of test force when \( a = 0 \))
  - Test force

- No kinematic connection with ground, for ex: a floating system

- If there are external forces such as gravity, the equation becomes
  \[ f + f^E = I^A a + p \]
Featherstone’s Algorithm: Inertia & Bias forces (1)

\[ f = \text{net force on link 1} + \text{force transmitted to link 2 through the joint} \]

\[ = f_1 + f_2 \]

\[ = I_1 a_1 + p_1 + I_2 a_2 + p_2 \]

\[ = I_1 a_1 + v_1 \times I_1 v_1 + I_2 a_2 + p_2 + v_2 \times I_2 v_2 \]

... (I)

Constraint imposed by the joint:

\[ a_2 = a_1 + v_1 \times v_2 + s \alpha \ldots (II) \]

Active joint force:

\[ s^T f_2 = Q \ldots (III) \]
Featherstone’s Algorithm: Inertia & Bias forces (2)

From (I), (II), and (III),

\[
\mathbf{f} = \left( I_1 + I_2 - \frac{I_2 s^T I_2}{s^T I_2 s} \right) \mathbf{a}_1 + v_1 \times I_1 \mathbf{v}_1 + v_2 \times I_2 \mathbf{v}_2 + I_2 \left( v_1 \times v_2 + s \frac{Q - s^T (I_2 v_1 \times v_2 + v_2 \times I_2 v_2)}{s^T I_2 s} \right)
\]

The above derivation is for a system with 2 links only. They can be generalized to multiple links by considering the following scenario:
Featherstone’s Algorithm: Inertia & Bias forces (3)

I_1 is still a simple rigid body but I_2 becomes an articulated body. By a similar derivation, we can find out the following recursions for the inertia and the bias force:

\[ I_i^A = I_i + I_{i+1}^A \]

\[ I_{i+1}^A s_{i+1} (s_{i+1})^T I_{i+1}^A \]

\[ (s_{i+1})^T I_{i+1}^A s_{i+1} \]

Use it’s child’s information. So if we can start from the terminal link….
Featherstone’s Algorithm: Joint acceleration (1)

Definition of joint velocity:

\[ \mathbf{v}_l - \mathbf{v}_b = s \, \dot{q} \quad \text{...(IV)} \]

Take derivatives:

\[ \mathbf{a}_l - \mathbf{a}_b = \mathbf{v}_b \times s \, \dot{q} + s \ddot{q} \quad \text{...(V)} \]

Force \( f \) applied through the joint:

\[ s^T f = Q \quad \text{...(VI)} \]
Featherstone’s Algorithm: Joint acceleration (2)

From (IV), (V), (VI), and our linear relationship

\[ f = I^A a_l + v_l \times I^A v_l \]

We can obtain

\[ \ddot{q} = \frac{Q - s^T \left( I^A \left( a_b + v_b \times s \dot{q} \right) + p \right)}{s^T I^A s} \]

\[ = \varphi(a_b, v_b, a_l, v_l, I^A, s, \dot{q}) \]
Featherstone’s Algorithm: Put all steps together

FUNCTION ABA_acceleration(q, q_dot, s, Q) {
    v(1) = 0;
    // Compute velocity for all links.
    FOR link_i=2 TO N
        v(link_i) = p_link(v(link_i)) + s * q_dot();
    // Compute the inertia and the bias forces.
    FOR link_i=N TO 1 {
        compute_I(link_i, link_i-1);
        compute_p(link_i, link_i-1);
    }
    // Compute acceleration for all the joints.
    a(1) = 0; // link 1's acceleration
    FOR joint_i=1 TO N-1 {
        compute_q_dotdot(joint_i);
        // Compute link acceleration for the next joint.
        a(joint_i+1) = a(joint_i) + v(joint_i).cross(s(joint_i)) * q_dot(joint_i) +
                      s(joint_i) * q_dot_dot(joint_i);
    }
}
Conclusions

- Inverse Dynamics Vs. Forward Dynamics
- Recursive Newton-Euler Algorithm for solving Inverse Dynamics problems
- Featherstone’s Algorithm for solving Forward Dynamics issues
- Use the recursion trick to make your program faster.
References

- Karen Liu, “Articulated Rigid Bodies,” slides from CS7496/4496 Computer Animation class at Georgia Tech

Highly recommend!