Proposal

Inverse Kinematics Accelerator

Yipeng Huang, Lianne Lairmore, Richard Townsend
{yipeng, lairmore, rtownsend}@cs.columbia.edu

February 24, 2014

1 Overview

In this project, we will build specialized hardware to tackle the inverse kinematics problem.

Inverse kinematics is widely used in robotics computing and in computer graphics. The problem takes as input a configuration of mechanical joints, which can be rotational or sliding, that are present in an arm or a leg. Then, it takes as input the limb's current shape and a target shape, solving for the required joint motions to get to the desired shape.

We will build a configurable solver on an FPGA, in hopes of speeding up solutions when compared to running a same algorithm on a regular CPU.

2 Motivation

Computers are increasingly embedded in the real world, often permanently attached to sensors and actuators. An example of such a system are robots. Computer systems that must coordinate with sensors and actuators are distinct from general purpose computers, and the desired hardware to support its applications will also be different.

Specialty hardware such as GPUs can support hard-hitting sensing algorithms, especially those that involve image processing. Equally important, but less well studied, is how computer hardware should adapt to support controlling actuators. Actuator algorithms have yet to appear in well known computer architecture research workloads. Early studies of computer architecture support for robotics show that general purpose CPUs suffer when running typical robotic workloads[1].

Problems that arise when controlling actuators, such as kinematics, dynamics, obstacle avoidance, collision detection, have been found to occupy a large portion of computer runtime

in robotics. Live measurements show 33-66 percent dedicated to embedded computer on robot. In particular, the inverse kinematic problem is interesting because it has features of two distinct workload categories: sparse matrix math and graph traversal[2]. This project will focus on building hardware for inverse kinematics.

3 Inverse Kinematics Algorithms

The general inverse kinematics algorithm relies on the following concepts, which we present in order:

3.1 Homogeneous Transformations

Forward and inverse kinematics operates on frames, which are as set of axes and coordinates that describe 3D space. Frames can be global or local. A local frame would be useful in describing the x, y, z positions of an object in space, along with the orientation (direction it is pointing) in space. We can transform from one frame to the next using homogeneous transforms, which are described as 4 by 4 matrices. For background information on homogeneous transforms refer to [3].

3.2 Kinematic Chain

Each joint of a robot has a coordinate frame; by convention without exception, revolute joints are represented as a rotation about the Z axis, and prismatic joints are translations along the Z axis. A collection of joints on an actuator form a kinematic chain.

3.3 Denavit-Hartenberg Parameters

The standard notation for describing an actuator is to first describe the rotation about the Z axis by a joint angle, translate along by the Z axis by the link offset, translate along the X axis by the link length, and rotate by the X axis by the link twist.

3.4 Jacobian

The matrix that relates the differential motion of joints to differential motion in cartesian space is called the Jacobian matrix. This matrix describes the velocity relationship between joints and the end of the actuator.[4]

3.5 Inverse Kinematics

The inverse kinematics solution for any robot would be perfectly solved if the inverse of the Jacobian is available. This is because the inverse Jacobian describes the requisite joint motions to get any velocity in cartesian space.[4]

3.6 Jacobian Transpose and Jacobian Pseudoinverse Algorithms

In reality, inverting a matrix is a costly operation, so the transpose or the pseudo-inverse of the Jacobian matrix are often used in inverse kinematics solvers.[5] In this project, we will explore using dedicated hardware to compute the inverse Jacobian matrix. If that proves difficult, we will use one of the usual matrix inverse approximations.

4 Architecture

For our project we will be using the FPGA as an accelerator for inverse kinetic computations. Software running on the Linux ARM processor on the SoCKit board will be driving the FPGA and will display its outputs on a monitor. The FPGA will be configured for specific robot with given joint parameters. The software will supply current and target Cartesian coordinates to the IK accelerator which will return updated joint configurations to move towards the given target coordinates. The software will use the joint configuration to update an image on the monitor. The resulting image should result in an animation of a appendage moving towards a target position.

5 Hardware

The FPGA design will consist mostly of floating point arithmetic. Working with floating point numbers will be our biggest challenge. Altera has mega functions built in to Quartus II which will have the tools build already working and tested modules for trigonometric functions. Without these functions we most likely would not have been able to do this project. The mega functions we are planning on using are ALTFP_ATAN and ALTFPSINCOS. As to be expected these functions take time to compute results. For example the ALTFP_ATAN takes 34 cycles to compute the tangent function. While working on the design we might find that other floating point arithmetic mega functions will need to be used like addition and subtration.

6 Milestones

Milestone 1

1. Implement canonical 3D inverse kinematics algorithm using C
2. Design block diagram of our system
3. Design top-level module describing the interface between the hardware and software sections of our system
4. Determine how to represent input and output with respect to the user. For example, our output could be textual (equations representing the result of the algorithm) or graphical (a picture of the actual robot limb in 3D space).

Milestone 2

1. Divide our C implementation code into "submodules"; each submodule will correspond to a testable portion of our system that will be coded in SystemVerilog. As part of this division, formalize input and output to each submodule.
2. Construct timing diagrams for each of our submodules

Milestone 3

1. Full implementation of our system
2. Develop testbenches for the different modules in our system

Final Project Presentation

1. Finish testing our system both in simulation and on the FPGA
2. Write up our final report and prepare our final presentation

References

[1] S. Caselli, E. Faldella, and F. Zanichelli, "Performance evaluation of processor architectures for robotics," in *CompEuro '91. Advanced Computer Technology, Reliable Systems and Applications. 5th Annual European Computer Conference. Proceedings.*, pp. 667–671, May 1991.

[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, pp. 56–67, Oct. 2009.

[3] "Introduction to homogeneous transformations & robot kinematics,"

[4] H. H. Asada, "Introduction to robotics chapter 5 differential motion,"

[5] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," tech. rep., IEEE Journal of Robotics and Automation, 2004.