RoboArm: Demonstrating Control Algorithms & FPGA Acceleration Using Embedded Processors

1. Abstract

Application-specific robotic systems often must operate with low latency and under considerable compute constraints in austere environments. To this end, strategic use of fixed point computation, in conjunction with FPGA-enabled acceleration can offer considerable performance increases with acceptable space/power tradeoffs. For the RoboArm project, a simple RRT\* path-planning algorithm was implemented in C++ to run on the provided BeagleBone MCU. This was used in conjunction with a Forward Kinematics algorithm and simple obstacle detection protocol to control a robotic arm with six degrees of freedom. The functions used to control the robotic arm were profiled using gprof to identify the most computationally intensive tasks. The most expensive function was then processed using the DAISY HLS toolchain and converted into synthesizable fixed-point Verilog using the ap\_fixed type format to run on a Basys3 FPGA. Using this set-up a speedup of X% was observed in the accelerated function. Fixed-point computation remains an important strategy in embedded robotics, and this project offers valuable insight into the application of this approach to improve robotic performance in constrained environments.

1. Introduction

Low-latency, low-power robotic control is crucial to a broad range of applications, including space robotics, remote operations, and humanoid robots. In all such applications, performing computationally expensive algorithms such as Forward or Inverse Kinematics presents a challenge. Traditional MCUs may not be optimal in these settings, either because they are unable to meet real-time constraints, or are unable to meet power constraints, or both. Acceleration using specialized hardware such as FPGAs or GPUs is a promising avenue to optimizing control systems under such conditions.[[1]](#footnote-1)

With this background in mind, this project set three core goals, relating to applied robotics:

1. Implementing RRT\*, forward kinematics, and obstacle avoidance on an embedded platform
2. Profiling of the system, and identification of the most computationally expensive elements
3. Offloading of expensive elements to an FPGA, quantifying speed-up, and validating the system’s performance

RRT\* (Rapidly-exploring Random Tree Star) is a path planning algorithm that builds a tree from a starting point, and expands randomly by selecting new nodes. It differs from standard RRT in that it rewires the tree dynamically to find optimal paths. The algorithm is widely used in several applications, including robotic path planning. Forward kinematics is a well-established method to translate robotic joint angles to cartesian coordinates. In this application, it was used to determine whether a given joint configuration was collision free.

Both algorithms are computationally expensive, and can be accelerated using specialized hardware. One method is to use fixed point arithmetic, at an acceptable accuracy tradeoff, and running this on an FPGA. To this end, a modified version of the DAISY (Design Automation for Integrated Synthesis of sYstems) toolchain was used, developed by researchers at Boston University. This modified toolchain represents a high-level synthesis (HLS) application that allows developers to write RUST code which is then automatically converted to synthesizable Verilog in ap\_fixed format, that can be deployed to FPGAs. This approach enables rapid hardware acceleration of algorithmic functions without requiring hand-written HDL, while maintaining tight control over bit-widths and precision.

A full view of the system is provided in the next section, Method. However, at a high level, a BeagleBone MCU was used to run the main control loops in C++. Gprof was used to identify bottlenecks. The DAISY toolchain was then used to convert the relevant functions to ap\_fixed format Verilog, which was ported to a Basys3 FPGA board. A robotic arm was then controlled in real-time using this hybrid set-up.

The remainder of this report is organized as follows: Section 3 describes the methodology, including RRT\*, forward kinematics, obstacle detection, and hardware acceleration workflow. Section 4 presents the study’s results. Section 5 outlines limitations and potential future work.

1. Method

Method: Method used is clearly explained, with your contributions, if you used ideas from our class readings or other materials, make sure to detail in the report. Your report should still be self-contained by including the used equations and formulations necessary for understanding the approach, any algorithms. The method should be well justified, and consider complete system usability in the real-world (20 pts).

* 1. RRT\*
  2. Forward Kinematics
  3. Obstacle Detection
  4. DAISY

1. Results

Results: Experimental setup should be clearly explained. Was any data used, evaluation protocol and metric definition details, are the experiments well justified, analyze the system comprehensively (with quantitative and qualitative results) (30 pts).

1. Limitations & Future Work

Limitations and future work: Discuss what didn’t work, e.g., if different from expectations, what wanted to implement but didn’t have time (e.g., to further tune), conclusion (5 pts).

1. Plancher, B., Neuman, S,. et al. 2021. Accelerating robot dynamics gradients on a CPU, GPU, and FPGA. in IEEE Robotics and Automation Letters (RA-L). Available online: https://brianplancher.com/files/Accelerating\_Dynamics\_Gradients.pdf [↑](#footnote-ref-1)