Cable-Anchor Robot Implementation using Embedded CD++

[Poster Abstract]

Keith Holman **Jeremy Kuzub** **Mohammad Moallemi** **Gabriel Wainer**

Carleton University Dept. of
Systems and Computer Engineering
1125 Colonel By Drive, Ottawa, Canada, K1S 5B6

{keith, jkuzub, omoallemi, gwainer}@sce.carleton.ca

ABSTRACT

We show the design and implementation of a robot controller with a unique locomotion system. We demonstrate that a discrete-event simulation based design provides a cost-effective, flexible, open workflow for modular robotic development. The robot is designed to translate against a vertical surface using cables fixed at one end that can wind on motor-controlled spools attached to the robot. This architecture was implemented first as a regressively tested simulation within CD++ then ported to Real-time CD++. Using the NXT++ interface library, a hardware implementation of the robot using Lego® Mindstorms™ was shown to be controllable.

Categories and Descriptions

B.1.2 [Control Structure Performance Analysis and Design Aids] *simulation*, I.2.9 [Robotics] *Commercial robots and applications*

General Terms

Design, Verification

1. Cable-Anchored Robot

In applications where terrain is too difficult to traverse using legs or wheels, other forms of robot locomotion must be found. Examples include disaster areas, such as building collapse, or environmentally sensitive locations where no disturbance can be tolerated. One form of locomotion that could allow 2D and 3D locomotion involves a cable-anchored robot. Rather than wheels, a cable-anchored robot is designed to hang from two or more points fixed above and around the desired area of movement connected by cables. The ends of the cables meet at the robot and are each wrapped around motor-driven spools which the robot can rotate to let out cable or take it in. This effectively allows motion through a space or across a plane.

The use of DEVS allowed for a bottom-up development and testing process. The use of DEVS made it easier to produce a modular design. Unit testing each of these models was simplified because of the use of simulation.

SIMUTools 2009, Rome, Italy

This enabled design flaws identified and isolated prior to porting the design to hardware. Simulation also gave visibility into the state of the system at any given time.

Using a bottom-up development and testing process resulted in a complex robot controller system that met desired performance goals. The flexibility of the anchor-cable locomotion system is offset by its geometric complexity; however, this was shown to be successfully addressed with a path planner that could linearize robot motions with controllable fidelity.

2. Implementation

A subset of this problem using two cables and a planar surface for movement was used as a real-world design specification. The robot would translate against a vertical surface using fixed cables that can wind on motor controlled spools attached to the top of the robot (Figure 1). The attachment points to the surface can be arbitrary, and the controller is modeled in such a way that target (x, y) Cartesian coordinates can be translated to desired cable take-up and incremental motor movements. The robot model is implemented so that the path between the current robot position and the desired position is calculated in steps of defined resolution. This allows linear robot movement to the target position regardless of the geometry of the robot and cable attachment points.

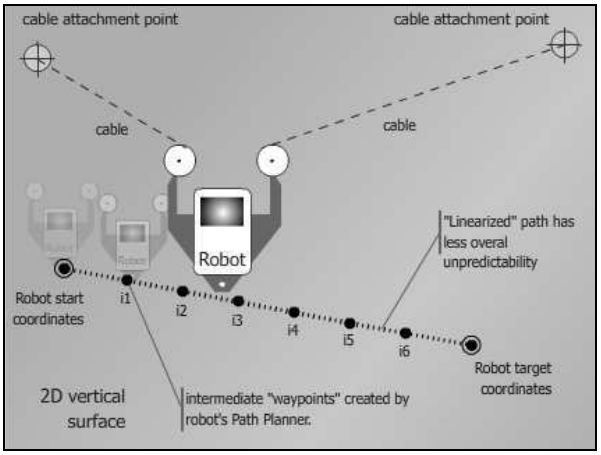

Figure 1: Robot path between start/target using path planner

The control algorithms were first simulated using the CD++ simulator [1] and it was then integrated to a hardware prototype im-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5743
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5743

plemented using the Lego Mindstorms™ robotics construction toolkit and E-CD++ [2].

3. System Architecture

A hardware implementation of the robot using Lego Mindstorms was shown to be controllable using event files. Open-loop motor control was demonstrated using the fundamentals of a simulated robot in the real world.

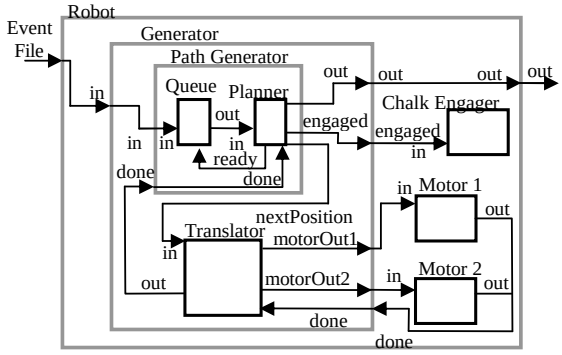

Figure 2: Overall Model of Robot System

Developing architecture for the robot controller involved breaking down the behaviour of the robot into sequential, event-driven steps. Each step had a unique function and could be more easily translated into atomic models for CD++ implementation.

The `Robot` is the top-level coupled model (Figure 2). This represents a cable-stay robot. Each robot is composed of a `Generator` and `Chalk Engager` and two motors. This simulates how the hardware cable-stay robot would work. A robot would contain controllers, in this case a chalk engager, and two motors that communicate with Mindstorms. The `Generator` translates input into a format understandable by each controller. The robot can be in either passive or active state. When in passive the robot is waiting for the next event from the event file when in active the robot is handling an event from the event file or interpreting a response from the subsystem [4].

Each atomic model within the design is responsible for a specific task. For example, the `Motor` is an atomic model that is provided with a new length for the amount of cable to expel. It also translates desired input from the system into correct values before forwarding the instructions to a connected Mindstorms actuator. If the `Motor` is requested to move to an impossible position an error flag is returned on an output port. Each movement instruction is sent as an event, which allows for testing in isolation.

4. Simulation Analysis

The initial design was implemented in CD++ to simulate robot behaviour. The simulation initially was not completely successful, and indicated an error in the path planning algorithms was not initially detected in the atomic model tests of the `Planner` and `Translator`. Specifically, the path planner failed to generate intermediate waypoints in the special case of vertical and horizontal lines. Initial model tests were done with only diagonal lines, so cases where the slope was infinite or zero were not part of that test plan. A simulation-based development like this allows more opportunities to test the system in a larger data space. The greater the level of integration when an error is found, the more difficult it is to pinpoint its source; the lesson is that exhaustive unit testing is necessary, but not sufficient, as some test cases will likely be missed.

This simulation also showed the value of testing algorithms in regression using two different derivation methods. Inputs and outputs should match if the system is working as expected; errors oversights are more easily spotted. The disadvantage may be in development time as a partial second set of robot control algorithms must themselves be derived, tested and debugged. The result, however, provides an additional level of confidence on the path to hardware implementation.

5. Robot Construction using Mindstorms

After detecting and fixing the initial problems, the simulated model behaved as expected, planning and following a physically feasible set of paths. Thus, we ported the model to E-CD++ controlling a Lego Mindstorms robot (in order to be able to reuse the model as an engineering education tool). Two servo motors were used to wind and unwind the cable spools. A third motor actuated the chalk writer.

Mindstorms hardware control via NXT++ and E-CD++ was successful. Motor commands were properly issued and acted upon by the hardware. Real-time DEVS allowed accurate timing of motor movements; with the motors properly characterized it was possible to provide good accuracy.

6. Conclusion

A hardware implementation of the robot using Lego Mindstorms was shown to be controllable using event files. Open-loop motor control was demonstrated using the fundamentals of a simulated robot in the real world as well as using simulation techniques to build a more robust system by intermediary analysis.

This implementation controlled Mindstorms hardware via a DEVS atomic model within a complex coupled model. This demonstrates that other Mindstorms sensors or actuator can be effectively modeled as atomic models within DEVS++ and the associated Real-time DEVS atomic model can be substituted in the E-CD++ framework with little additional modification required. A workflow was thus successfully demonstrated from simulation to hardware implementation with DEVS, Real-time DEVS, and Mindstorms.

7. REFERENCES

[1] Wainer, G., "CD++: a toolkit to define discrete event models". Software, Practice and Experience. Vol. 32, No.3. pp. 1261-1306. November 2002.

[2] E-CD++: a tool for modeling embedded real-time applications". J. Yu, G. Wainer. In *Proceedings of the 2007 SCS Summer Computer Simulation Conference.* San Diego, CA. 2007.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5743
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5743