Real-Time Object Transportation System Using LEGO Mindstorms RCX

Team:

- Vishnu Dasa Harish Babu vdasaha@ncsu.edu
- David Boyuka daboyuka@ncsu.edu

Project URL:

http://www4.ncsu.edu/~vdasaha/rtcs.htm

Project Overview:

We will construct an object transportation system using the LEGO Mindstorms RCX. The objects (likely small balls, e.g. ping-pong balls) will be input to the system by hand, being placed into various input bins. These objects must be delivered to a common output bin. Objects are divided into "classes" based on their input bin, which we will represent using different colors (i.e. red balls in one bin, blue in another, etc.)

The input bins and output bin will be oriented in a ring structure, in some order known a priori. The system will feature an RCX-powered vehicle capable of moving in either direction around this ring, picking up objects at input bins and storing them for transport, and dropping off all stored objects at the output bin. We will use brickOS [1,2,3,5] as our software platform.

Real-time Goals:

Consider the i'th object arriving in the system. Let a[i] be its arrival time, p[i] be its pickup time, and d[i] be its delivery time (all absolute). Also, let c[i] be the class/color of the object. Given this, there are two basic real-time goals we could impose (Problem 1 and Problem 2):

1. Guarantee that each object is picked up by a class/color-specific deadline relative to its arrival time, and also guarantee that each object will be delivered to the output by another class/color-specific deadline relative to its pickup time. That is:
   for all objects i: p[i] - a[i] <= D1[c[i]], and d[i] - p[i] <= D2[c[i]]
2. Guarantee that each object is delivered by a class/color-specific deadline relative to its arrival time. That is:
   for all objects i: p[i] - d[i] <= D[c[i]]

We will explore one or the other of these guarantees (or both, if time permits). Deciding which to pursue is also part of the project, as this will require further evaluation of their difficulty/ complexities (see below). Problem 2 seems harder, but more examination is needed.

Real-Time Challenges:

A simple strategy of checking the input bins in (counter)clockwise order, then delivering to the output bin, gives us one possible worst-case bound, but this may not be able to schedule all systems. For instance, “checking input bins” might take significant time depending on the robot, short-deadline objects may miss deadlines due to unnecessarily checking other bins on the way, and it might be unnecessary to check some bins on every circuit due to very long deadlines.

Thus, we would like to do better than this by using some scheduling algorithm(s). We would like to formulate the system in terms of tasks, which would allow us to apply various real-time techniques/algorithms to meet these guarantees. Two issues make this difficult:

1. Not only do objects arrive at arbitrary times (like sporadic tasks), but the arrival times themselves are unknown at runtime (unlike sporadic tasks): the RCX can only check input bins when physically present, and objects may have waited for a while already.
2. The time required to move to an input/output bin is variable, depending on the RCX’s current location. An overall worst-case bound on this time would be very loose.
3. Modeling pickup and delivery as simple tasks can be very inefficient, as visiting bins in arbitrary order could lead to “excessive seeking”.

One solution to the first issue we will explore is using fixed periodic bin checking (though this has efficiency issues due to problem 3). We might deal with problem 2 via some form of LLF scheduling: since LLF updates mid-job, we can also update execution time mid-job. We must ensure that such an algorithm makes progress, however, or else “thrashing” may occur.

Another issue to consider is what action to take if/when objects cannot be picked up/delivered on time (i.e. missed deadlines). The system might be well modeled with some variant of sporadic tasks, in which case we can apply acceptance tests to incoming delivery tasks and can choose whether or not to accept them based on feasibility. In any case, the reaction to a missed deadline/rejected sporadic task will probably be the same: skip the infeasible job, and raise some error signal, such as a distinctive tone sequence or flashing light.

Physical Construction Challenges:

We have access to a train LEGO set [4], which would allow mounting the RCX on a track. This would greatly simplify ring movement. We plan to use touch sensors to detect arrival at bins, touch/light sensors to detect whether objects are present in the bins, motors to empty/fill input/output bins, and a powered engine block to move the RCX (part of the train kit).

It’s possible we could run out of sensor ports, actuator ports, or CPU cycles on one RCX, in which case we will procure a second RCX, and add basic coordination over IR to the project if need be. We would like to avoid this, as it adds extra complexity and difficulty in synchronization between actions, but we will consider it as a fallback position.

Milestones:

1. Background research tasks:
    a. Learn brickOS, run example programs for practice
    b. Look at brickOS kernel and determine where we will need to make modifications
2. Algorithm design tasks:
    a. Come up with some strategies for meeting deadlines more efficiently than an extreme-worst-case bound approach (factoring in current position, etc.)
3. Physical construction:
    a. Setup the tracks and the layout
    b. Build input/output bins
    c. Construct the basic delivery vehicle (with pickup/dropoff mechanisms)
    d. Add bin station sensor to vehicle (for sensing when we arrive)
    e. Add object pickup sensor (for sensing if objects are present in a bin)
4. Programming tasks:
    a. Code the motion routines (vehicle movement, detecting current position)
    b. Code the loading/unloading routines for moving objects
    c. Code the real-time scheduler to control the above two modules
        ■ Note: this won't be a real-time scheduler in the traditional sense, as we are primarily concerned with physical action time, rather than CPU time
5. Testing tasks:
    a. Come up with (physical) test cases, both feasible and infeasible
    b. Run the tests (will record video and analyze later, as testing is time-sensitive)

All milestones within a major category (1-5) should done serially, with major categories being parallel to each other. Exceptions: 3d/e can be reordered, 4* can be done in parallel, 2a must be done before 4c.

Assignments:

- David: 3d,e, 4c, 5a
- Vishnu: 3a,b, 4a,b
- Both: 1a,b, 2a, 3c, 5b

References:

[1] http://brickos.sourceforge.net/documents.htm
[2] http://freshmeat.net/projects/brickos/
[3] http://did.mat.uni-bayreuth.de/~matthias/veranstaltungen/ws2004/mindstorms/doc/brickos-howto.html
[4] http://shop.lego.com/en-US/Cargo-Train-7939
[5] "Introduction to the legOS kernel" - http://legos.sourceforge.net/docs/kerneldoc.pdf