Proceedings of the 1999 IEEE
International Symposium on Assembly and Task Planning
Porto, Portugal - July 1999

Using Hybrid cc to Simulate Multi-tier Peg-in-hole Problems

Qingchuan Fu and Michael S. Branicky
Department of Electrical Engineering and Computer Science
Case Western Reserve University, Cleveland, OH 44106-7221
Email: {qxf2, msb11}@po.cwru.edu

Abstract

Many robotic assembly tasks can be viewed as multi-tier peg-in-hole, or "peg-in-maze," problems. Such tasks are hybrid system control problems since they combine continuous movement with event detection. This paper presents Hybrid cc as a software tool for simulating hybrid systems and uses it to simulate two-dimensional multi-tier peg-in-hole problems.

1 Introduction

Many robotic assembly tasks, such as clutch assembly and gear meshing, can be viewed as multi-tier peg-in-hole, or "peg-in-maze," problems [1]. The insertion process includes both discrete events and continuous movements, which makes such problems into hybrid system control problems. Hybrid systems are systems with both discrete and continuous dynamics. Continuous dynamics are usually represented by differential equations, while discrete dynamics can be viewed as transitions between nodes. Each node has its own continuous dynamics. Transitions occur when certain events happen. During transitions, values of variables can be reset.

Peg-in-hole problems are usually hard to solve and even simulate for lack of suitable tools to model the hybrid nature of the system. In this paper, we demonstrate ways to use a new language, Hybrid cc [2, 3, 4], for modeling hybrid systems and simulating multi-layer peg-in-hole problems.

The paper is organized as follows. Section 2 describes the Hybrid cc language. Section 3 presents multi-tier peg-in-hole problems and simulation results from Hybrid cc. Section 4 describes extensions and Section 5 gathers conclusions.

2 Hybrid cc

Hybrid cc is a hybrid concurrent constraint language. It was invented by Gupta *et al.* [2, 3, 4] and is naturally good for modeling hybrid systems because it can encode both discrete and continuous dynamics. It has been used to model and simulate photocopier systems [5].

2.1 Hybrid cc language

Hybrid cc is a constraint-based language. There are three types of constraints: continuous (arithmetic), non-arithmetic, and boolean constraints. A simple arithmetic constraint may be $x^2+y^2=1$, which defines a unit circle. A simple non-arithmetic constraint may be X = "abc", which defines a string.

To organize constraints to perform the behaviors the system will exhibit, Hybrid cc has some control constructs. Examples include *if...then...*, *unless...then...*, *hence,* etc. The control construct *hence* needs special attention. Once we say *hence A,* constraint *A* will be executed in all future phases. By using *hence*, constraints can be extended across time. Furthermore, using *hence* we can evolve other constructs that work over time as well, e.g. *always...*

In Hybrid cc, any set of statements separated by a "," is treated as a parallel composition of statements. They are concurrent, and they will be considered simultaneously. Runtime behaviors of Hybrid cc are controlled by constants. One can choose the type of the integrator, step-size of the integrator, etc. by setting those constants.

2.2 Computational Model

Execution of Hybrid cc programs is different from traditional programming languages such as C. It is not sequential but includes two types of phases, point phases and interval phases. From the literal meaning, we can easily understand that a point phase is a point on the time axis, and an interval phase is an open-ended line segment on the time axis. All the discrete changes should take place in the point phase. In an interval phase, computation progresses according to a specific continuous dynamics. So an interval phase always occupies a period of time.

0-7803-5704-3/99/$10.00©1999 IEEE

A Hybrid cc program starts in a point phase with time t = 0. Then it operates alternately in point and interval phases until the maximum chosen simulation time has elapsed or some errors occur. During a phase switch, information needs to be passed from one phase to the other. This is done by using a "store". We can think of the "store" as a memory space, but "store" in Hybrid cc not only holds values of variables as a traditional store; it maintains constraints on possible values of variables, which means it tracks the upper and lower bounds of values of variables. Just like "store" has all the current constraints of variables, the "prev store" has constraints of variables at a previous stage. So we can pass information by copying them from "prev store" to current "store". Except for the purpose of passing information, "store" has other crucial functionality. By writing values to "store" during point phases, discrete changes can be made. Based on the values in the "store", whether a Boolean constraint is true or false can be checked. The integrator uses values in the "prev store" to calculate the values for the current "store".

2. 3 An Example

Let us first look at a simple hybrid system. Consider the furnace control system depicted in Figure 1. Here, x is the room temperature (in degrees Celsius), y is the total elapsed time, and z records the time that the furnace is on. The dotted variables in Figure 1 are time derivatives. The action of the controller is such as to maintain the temperature between 21 and 25. The Hybrid cc program for this system is as follows:

```
#SAMPLE_STYLE 1
x=21, y=0, z=0,
St0(),
always {
    cont(x), cont(y), cont(z),
    St0=(){
        do always {y'=1, z'=1, x'=2}
        watching (x>=25),
        when (x>=25) do St1()
    },
    St1=() {
        do always {y'=1, z'=0, x'= -2}
        watching (x<=21),
        when (x<=21) do St0()
    }
},
sample(z), sample(y), sample(x)
```

St0 is the furnace-on state, while St1 is the furnace-off state. The primed variables in the program are the time derivatives. At first, the program defines initial conditions for the system. In each state of the system, it defines continuous dynamics (encoded inside *do always* statement) and guard (encoded as *watching* conditions). When a guard becomes true, the system has to jump from the current state to the other state. *cont(x)* means x is continuous.

The performance of the system can be found by executing the above Hybrid cc program. The data for x, y, and z is sampled in time (cf. last line of program) and stored in a file. That data file was read by MATLAB and is plotted in Figure 2.

If values of variables need to be reset during jumps, this can also be done easily in Hybrid cc. From this simple example it is clear how easy it is to use Hybrid cc programs to model and simulate hybrid systems.

3 Multi-tier Peg-In-Hole Problems

Now let us consider a multi-layer peg-in-hole problem solved by using Hybrid cc. (See Figure 3.)

There is a box with a peg and two layers inside. The peg is a cylinder. Each layer has a hole in it. The hole need not be at the center. The goal is to move the peg through the holes. The task is successful once the peg touches the bottom of the box. The peg can move in both the x and y directions, so it is a two dimensional problem. Layers can only move with the peg in the x direction.

The peg is represented by a fixed point on it, which is the center point on the bottom of it. So the parameters for the peg include: x position (px), y position (py), x velocity (vx), y velocity (vy), height (peg_height) and radius (peg_width). The reference point for each layer is the center of the hole on its surface, so the parameters for a layer are its x position (px), y position (py), radius of its hole (radius), left-width (lwidth), right-width (rwidth), and height (height).

3.1 Guidelines for Coding the System and Control Strategies into Hybrid cc

Class definitions: In Hybrid cc, there is a constraint that allows you to set up a class definition with a list of parameters and a list of properties for the class. Properties can be used in other expressions. From a class, as many objects as necessary can be created.

Different objects can be distinguished by their names. The class defines the general dynamics for all the objects created from the class. Using class definitions, classes for layers and pegs can be defined. Two different layers can be created from the layer class using different names and different parameters. For example, in the following code, a Layer class is defined and two different layers are created.

```
Layer=(initpx,initpy,x,y1,y2,z)
[px,py,vx,vy,radius,lwidth,rwidth,height,Changex]
{
  px=initpx, py=initpy,
  vx=0, vy=0,
  always {
    radius=x, lwidth=y1, rwidth=y2,height = z
  },
  always{
    cont(px), cont(py),
    px'=vx, py'=vy,
    unless Changex then vx'=0,
    vy'=0
  }
},
Layer(L1, 60, 220, 12, 60, 22, 110),
Layer(L2, 20, 110, 12, 20, 62, 110),
```

Strategies: Since the peg is not aware of the locations of the holes and velocity control is used here, there are three control strategies. One is to move the peg along surfaces of layers searching for holes using constant velocity (x direction movement only). Another is to move the peg down the hole quickly once the hole is found (y direction movement only). The last is to stop the movement of the peg once the task is successfully accomplished (peg hits bottom of the box). Different control strategies will be used when the peg is in different states. Whether or not the peg will actually be in those states depends on whether constraints encoded in the program will become true or not. For example, the following code shows constraints for checking whether the task is finished or not (the reference point of the peg is on the bottom of the box or not), and when it becomes true, it invokes control strategy *Complete( )*, which stops the movement of the peg.

```
Finish=(){
        always forall Peg(X) do {
                if(X.py<=ymin) then Complete()
        }
},
Complete=() {
        if(prev(P1.vx)!=0) then {P1.Hitx, P1.vx=0},
        if(prev(P1.vy)!=0) then {P1.Hity, P1 vy=0}
},
```

Behaviors of the system: Besides the controlled movements, there are other natural (autonomous) behaviors of the system. For example, the peg will bounce back upon hitting a wall, the peg will move freely from its start position according to its initial velocities, and a layer will move with the peg if the peg is hitting it.

Separation of the controller and the system: Hybrid cc programs can include other Hybrid cc files by using *#INCLUDE <FILENAME>*. This is equivalent to copying over the code from file *FILENAME* and inserting it at that point. By using this method, the behaviors of the controller and the behaviors of the system can be separated into two files, allowing them to be considered separately. This facilitates the development and testing of different strategies for the same underlying assembly problems.

3. 2 Results

Using the above guidelines, we completely coded the constraints for multi-tier peg-in-hole assembly problems plus their associated control strategies (code not shown for lack of space). This code was then used to simulate the system under a variety of different conditions, strategies, and problem parameterizations. The results are as follows.

Figure 3 shows an initial condition for the system. The asterisk is the starting position of the reference point on the peg. Figure 4 shows the movement of the reference point on the peg based on the initial conditions shown. The peg starts moving down and right according to its initial velocities. Upon hitting the wall, its x velocity reverses, but it continues to descend until the first layer is contacted. The search then begins in the positive direction (using velocity 5). Upon exhausting the search in that direction, it begins to search in the opposite direction until the hole is detected. It then moves down and continues in a like manner for the second layer. The figure shows the task has been successfully accomplished.

Figure 5 shows a different behavior of the system when the control strategy is changed (instead of searching the positive x direction first, the peg searches the negative x direction first).

Figure 6 shows an initial condition where the two layers are aligned. Figure 7 shows that the peg can go straight down once it finds the first hole. There is no need for the search of the second hole.

Figure 8 shows an extreme set of problem parameters in which the task can never be finished. This is apparent from the movement of the first layer shown in Figure 9, which depicts repeated oscillations.

4 Extensions

Velocity control is easy to model, but force control is typically used in robotic assembly [10]. Force control in the y direction is not hard to model, although it requires rewriting of functions and portions of the peg class concerning velocity control before. Force control along the x-axis is also easy to model if we assume that *force/mass* is constant no matter whether the peg is moving alone or with the first layer. Figure 10 shows the force control result for the initial conditions shown in Figure 3 (note peg's starting y position is different).

The multi-tier peg-in-hole problems can also be extended to three dimensions. In this case, a more complicated x-y planar search is needed instead of linear search in the two-dimensional model [1, 6, 10]. One method is rectangular search, shown in Figure 11. However, in this case, moments need to be considered because the layer can tilt inside the box.

Herein, multi-tier peg-in-hole problems were simulated by Hybrid cc, using a hybrid automata model. Such assemblies can also be verified by using tools such as HyTech [7, 11]. In HyTech, a whole set of initial conditions can be verified at once for an assembly. HyTech is able to verify whether the assembly is always possible, always impossible, or sometimes possible. It can give bounds on the assembly time and compute the set of all reachable configurations. Velocity uncertainties can be easily modeled in HyTech. Force control models make the hybrid automata for the system nonlinear, but clock-translation or linear phase-portrait approximation can be used to change the nonlinear automata into linear automata that HyTech can handle [7, 11].

5 Conclusions

Hybrid cc is suitable for modeling hybrid system control problems such as those that commonly arise in robotic assembly. We introduced the language and then showed how to model a class of multi-tier peg-in-hole problems. The code was made efficient using class definitions. Further, we structured it to separate modeling of the control strategy used and the underlying assembly problem itself. Finally, Hybrid cc was used to simulate the system under a variety of different conditions, strategies, and problem parameterizations. The program correctly deals with the mixed continuous-discrete nature of the assemblies and gives intuitively satisfying results.

Current work consists of using Hybrid cc and other tools, such as HyTech [11], to design, test, and verify control strategies for real-world assemblies like clutches and gear meshing. See [7].

Acknowledgments

This work was performed under contract #98015 with the National Center for Manufacturing Sciences. This support is gratefully acknowledged.

References

1. S.R. Chhatpar. *Experiments in Force-Guided Robotic Assembly.* M.S. thesis, Electrical Eng. and Computer Science Dept., Case Western Reserve U., Jan. 1999.
2. B. Carlson and V. Gupta. The Hybrid cc programmer's manual. http://ic-www.arc.nasa.gov/people/vgupta/hcc/papers.html, 1997.
3. V. Gupta, R. Jagadeesan, and V. Saraswat. Hybrid cc, hybrid automata and program verification. *Hybrid Systems III*, Vol. 1066, *Lecture Notes in Computer Science*, Springer, 1996.
4. V. Gupta, R. Jagadeesan, V. Saraswat, and D.G. Bobrow. Programming in hybrid constraint languages. *Hybrid Systems II*, Vol. 999, *Lecture Notes in Computer Science.* Springer, 1995.
5. V. Gupta, V.A. Saraswat, and P. Struss. A model of photocopier paper path. *Proc. Second IJCAI Workshop on Engineering Problems for Qualitative Reasoning*, Montreal, August 1995.
6. L.E. Huang. *Impedance Control for Accommodation of Position Uncertainty in Robotic Assembly.* M.S. thesis, Electrical Engineering and Applied Physics Dept., Case Western Reserve U., March 1998.
7. Q. Fu. *Simulation and Verification of Robotic Assembly Tasks.* M.S. thesis, Electrical Eng. and Computer Science Dept., Case Western Reserve U., June 1999.
8. W.S. Newman, M.S. Branicky, H.A. Podgurski, S. Chhatpar, L. Huang, J. Swaminathan, and H. Zhang. Force-responsive robotic assembly of transmission components. *Proc. IEEE Intl. Conf. Robotics and Automation*, May 1999.
9. W.S. Newman, M.S. Branicky, S. Chhatpar, L. Huang, and H. Zhang. Impedance based assembly. *Video Proc. IEEE Intl. Conf. Robotics and Automation,* Detroit, May 1999.
10. S.R. Chhatpar and M.S. Branicky. A hybrid systems approach to force-guided robotic assemblies. *Proc. IEEE Intl. Symp. Assembly and Task Planning*, Porto, Portugal, July 1999.
11. T.A. Henzinger *et al.*, A user guide to Hytech. http://www.eecs.berkeley.edu/~tah/HyTech, ver. 1.04, 1996.

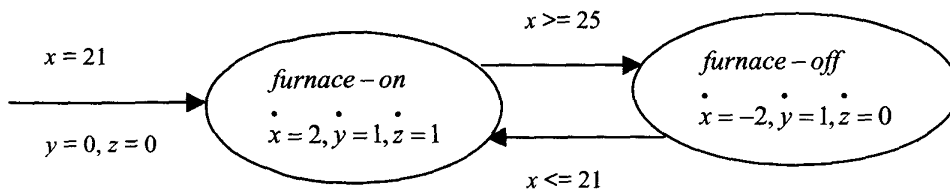

Figure 1. Furnace control system

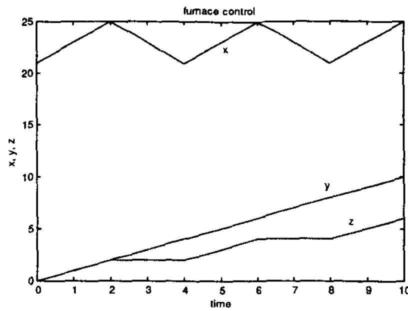

Figure 2.
Performance of the furnace control system

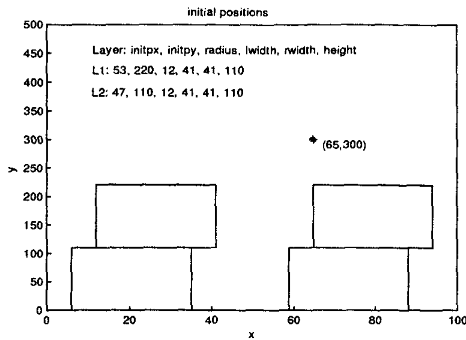

Figure 3.
Case 1: Initial conditions

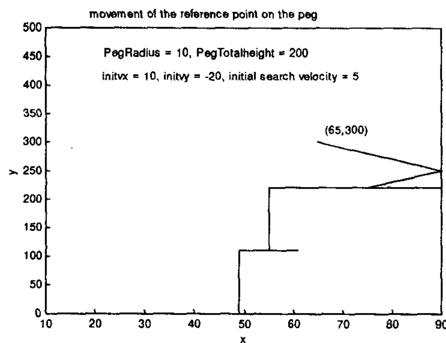

Figure 4.
Case 1: Peg's movement

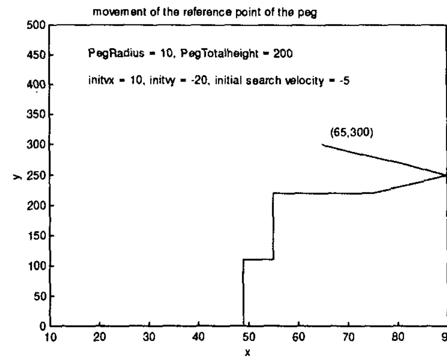

Figure 5.
Case 1: Peg's movement under different strategy

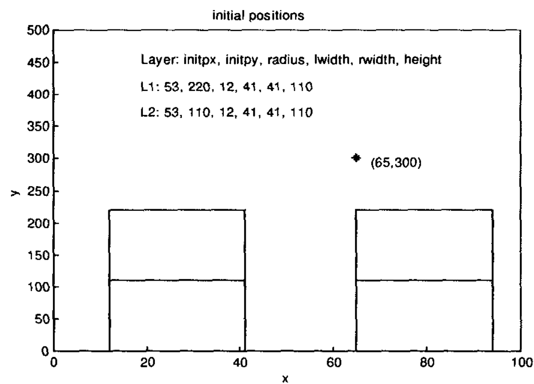

Figure 6.
Case 2: Initial Conditions

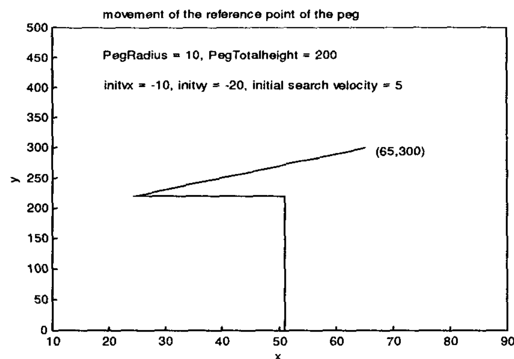

Figure 7.
Case 2: Peg's movement

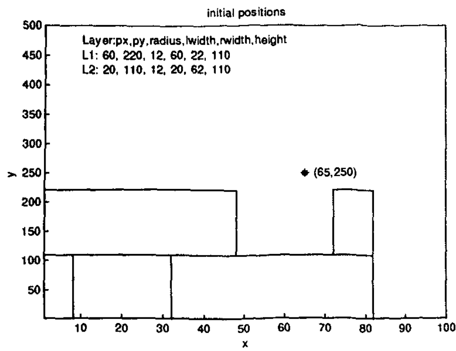

Figure 8.
Case 3: Initial conditions

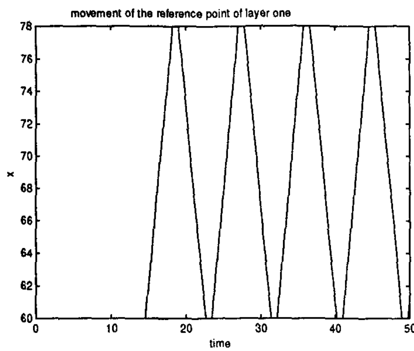

Figure 9.
Case 3: Layer 1's movement

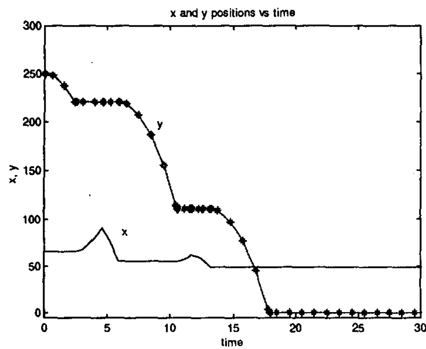

Figure 10.
Force control for two-dimensional peg-in-hole problem

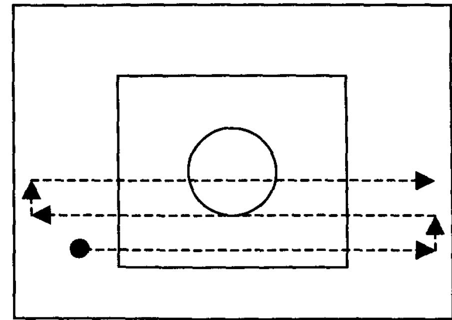

Figure 11.
Top view of the problem and an illustration of the rectangular search strategy. Big rectangle: the box; small rectangle: a layer; circle: hole; dashed lines: search route.