

A Computational Framework for the Verification and Synthesis of Force-Guided Robotic Assembly Strategies

Michael S. Branicky¹ and Siddharth R. Chhatpar²

¹ Case Western Reserve University, Electrical Eng. and Computer Science,
Glennan 515B, 10900 Euclid Ave, Cleveland, OH 44106-7071, U.S.A
`msb11@po.cwru.edu`

² Case Western Reserve University, Mechanical and Aerospace Engineering,
Glennan 109, 10900 Euclid Ave, Cleveland, OH 44106, U.S.A
`src2@po.cwru.edu`

Abstract. Robotic assemblies are inherently hybrid systems. This paper pursues a class of multi-tiered peg-in-hole assemblies that we call *peg-in-maze* assemblies. These assemblies require a force-responsive, low-level controller governing physical contacts plus a decision-making, strategic-level supervisor monitoring the overall progress. To capture this dichotomy we formulate hybrid automata, where each state represents a different force-controlled *behavior* and transitions between states encode the high-level strategy of the assembly. Each of these behaviors is set in 6-dimensional space, and each dimension is parameterized by spring and damper values (an impedance controller). Our over-arching goal is to produce a computational framework for the verification and synthesis of such force-guided robotic assembly strategies. We investigate the use of two general hybrid systems software tools (HyTech and CTool) for the verification of these strategies. We describe a computational environment developed at Case to help automate their synthesis. The implementation of these strategies on actual robotic assemblies is also described.

1 Introduction

Robotic assemblies are inherently hybrid systems [1] because they involve the making and breaking of contacts [2]. This paper pursues a class of such assemblies that we call *peg-in-maze* assemblies, which includes real-world examples such as clutch mating and gear meshing. These are multi-tiered versions of traditional *peg-in-hole* assembly problems. There are further complications due to the fact that the layers may move with respect to each other (allowing position uncertainties of the components that exceed the corresponding assembly clearances). Thus, successful assembly requires altering component locations through physical contact between the mating parts in a search for the next insertion hole. Once the hole is found, insertion may proceed. This already induces a layer-by-layer switching between search and insertion *behaviors*. Further complexity may

arise if the assembly reaches a dead-end or JAM state, requiring the invocation of some other behavior (*backing-up*, *re-planning*, or *aborting*).

We have successfully automated example *peg-in-maze* assemblies using a two-level control in previous work, including a forward clutch (described below), reverse clutch, sun/planetary gear, torque converter, and several other assemblies [3] [4] [5]. To govern physical contacts, we use a parameterized family of low-level force-responsive algorithms known as natural admittance controllers [6]. To switch between different low-level controllers, invoke new behaviors, and monitor the overall progress of the assembly, we use a higher-level discrete-event supervisor [7]. Together, these two levels of control implement an assembly strategy that guides the assembly process from start to finish.

Both the strategy and the assembly process itself are hybrid systems. Any *run* of the system consists of continuous movements (such as searching for a hole or moving down until contact) that are governed by continuous control laws and discrete events (such as finding a hole or contacting a layer) that are processed by a logical decision maker. Thus, to formally model *peg-in-maze* assemblies and their associated assembly strategies, we must use a hybrid systems approach [8]. In this paper, we formulate hybrid automata [10] models of assemblies/strategies, where each state represents a different force-controlled behavior and transitions between states encode the high-level strategy of the assembly.

Our over-arching goal is to produce a computational framework for the verification and synthesis of such force-guided robotic assembly strategies. We investigate the use of two general hybrid systems software tools, HyTech and CEtool, for the verification of these strategies. Herein, we briefly describe the results obtained with these tools and our own C++ analysis algorithms specifically tailored to *peg-in-maze* problems.

Finally, we describe a computational environment developed at Case to help automate the synthesis and implementation of force-guided robotic assembly strategies. The tool includes a State Machine Editor for entering overall strategy and a Mode Editor for creating the underlying force-controlled behaviors. There is also a Code Generator and a Real-Time Monitor.

The paper is organized as follows. Section 2 describes *peg-in-maze* assemblies further and formulates the hybrid automaton representation of the strategies to automate them. Section 3 presents a simplified two-dimensional *peg-in-maze* assembly, which is used to fix ideas throughout. Section 4 describes the verification of assembly strategies using the general hybrid systems software tools. Section 5 outlines the computational environment for strategy synthesis and code generation, used for actual experiments. Finally, Sections 6–7 summarize results, conclusions, and future work.

2 Peg-in-Maze Assemblies and Assembly Strategies

In this section, we formulate hybrid automata models for *peg-in-maze* assemblies. We start by describing a real-world example, abstract this to a representative *prototype* assembly, and subsequently formulate our model.

The real-world assemblies we have investigated are from a Ford automatic transmission. One of them, insertion of a spline-toothed hub within a forward clutch assembly is described here. See Fig. 1(a,b). The forward-clutch housing, shown in Fig. 1(b), consists of a cylinder with five internal rings (clutch plates), each with spline teeth about the inner radius. These rings are each capable of sliding independently in the horizontal plane by up to 1.6 mm and rotating about a vertical axis. A cylindrical hub with teeth on its outer surface, shown in Fig. 1(a), is to be inserted within the housing of Fig. 1(b). However, the clutch plates are not, in general, aligned either radially or tangentially. The radial assembly clearance between the hub and a ring is 0.23 mm. Thus, the radial assembly uncertainty is more than six times larger than the clearance required for success. Additionally, the hub must align with the z -rotation of each spline ring within 5 mrad for successful assembly, even though the tooth spacing is 160 mrad.

The forward clutch mating and other assemblies cited in Section 1 belong to a special class that we call *peg-in-maze* assemblies, which are multi-tiered versions of the traditional peg-in-hole problem. We have also built a prototypical peg-in-maze assembly for illustration and experimentation purposes. See Fig. 1(c,d,e). This assembly abstracts the salient features and peculiarities of our example transmission assemblies. By construction, our prototype is analogous to the forward-clutch assembly problem. Namely, the clutch plates are represented by *layers* and the hub is represented by a keyed peg.

The prototype assembly has an unconventional peg-in-hole type structure. The female part is made up of five layers; each is a Plexiglas sheet, measuring $160 \times 170 \times 11$ mm thick. These layers are stacked in a larger rectangular container, measuring 190×200 mm. Hence, the layers are capable of x - y sliding motion relative to each other and small rotations about the vertical axis. Each layer has

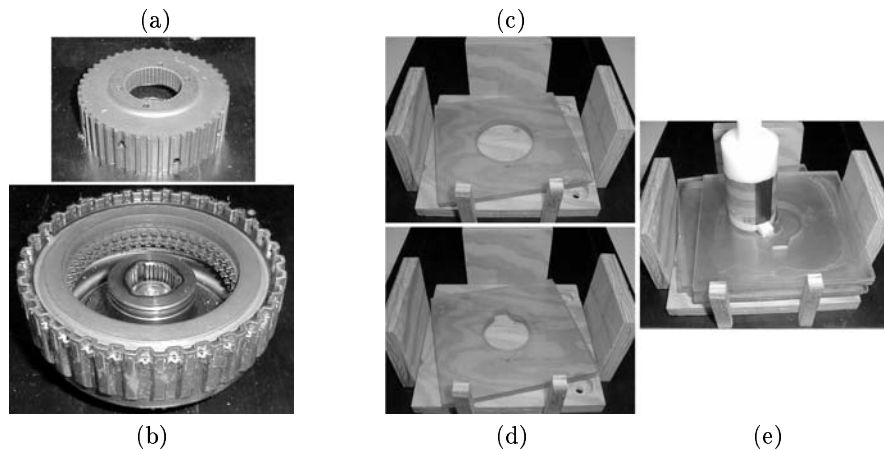


Fig. 1. (a,b) Forward clutch assembly; (c,d,e) Peg and layers in the prototype peg-in-maze assembly

a characteristically-shaped hole (circular, single-slotted, or double-slotted) cut in it through which the peg can pass, as shown in Fig. 1(c,d). The peg is cylindrical in shape, except for a small key (Fig. 1(e)). The structure has been made with large clearances (up to 11.5 mm) between the peg and the holes, however, the position eccentricities achievable by the holes (up to 42.5 mm) are much larger than the insertion clearances.

The assembly experiment starts with the peg on top of the first layer. The goal is to insert the peg through the holes on all the layers and reach the bottom of the box. To achieve insertion through all the layers, the peg should search for the hole on a layer and, finding it, move down to the next layer, and then repeat the search. The positions of the layers inside the container are arbitrary and hence the peg cannot be given a predetermined trajectory that will lead it straight to the holes. Trying to break down the complex scheme into simpler strategies, we can identify two distinct behaviors. One behavior (*Search*) is the search carried out to find the hole after contacting a layer, while the other (*Insert*) is to move down to contact another layer once alignment has been achieved. (There is also a *Layer Identification* behavior, not described here [8].) Although within a behavior the control law and search strategy are continuous, the transition between behaviors is a discontinuous jump. Hence, this problem falls under the domain of hybrid systems and should be treated as such. The hybrid automaton model of the controller for this peg-in-maze assembly is described below.

2.1 Discrete States and Events for the Peg-in-Maze

We have identified two primary behaviors for this assembly: *Search* and *Insert*, for each layer. Thus, in all there are 10 discrete states (for 5 layers) that the assembly can be in at any point of time. In addition, there is a START state, and the final ACCEPT state, making a total of 12 states. Other specialized routines, such as *Search Initialization*, are included as parts of the primary behaviors.

SEARCH states: In each SEARCH state, the controller moves the peg to perform a rectangular search for the hole (there is also a search in rotation for key-slot alignment and the aforementioned layer identification; see [5] [8]). In fact, each SEARCH state, S_i , itself consists of two functions, or substates, grouped together (Fig. 2(a)). Such hierarchical state machines can be represented using a formalism evocative of StateCharts [11].

On entering the SEARCH state, the controller shifts to the *Search Initialization* routine, and guides the peg to the center of the search area. In the *Rectangular Search* routine the controller moves the peg along a trajectory forming a rectangle of increasing side-length in the x - y plane (Fig. 2(b)). While searching, the peg is also controlled to maintain a downward force (z) on the layer. These movements are accomplished by moving the impedance controller's *attractor* [3].

If the peg finds the hole, the search is successful and the controller must shift to the corresponding INSERT state as the peg starts to move down through the layer. To recognize this transition in the state of the assembly, the controller

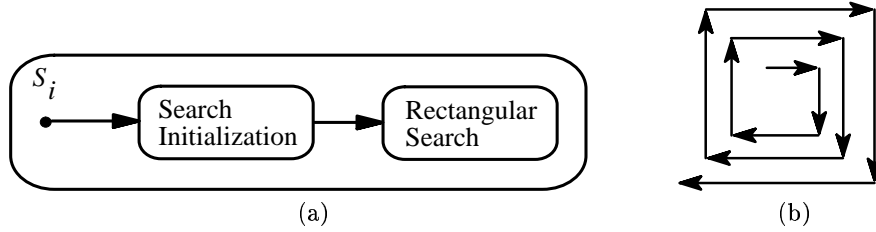


Fig. 2. (a) “StateChart” for SEARCH, (b) Simple rectangular search path in $x-y$ plane

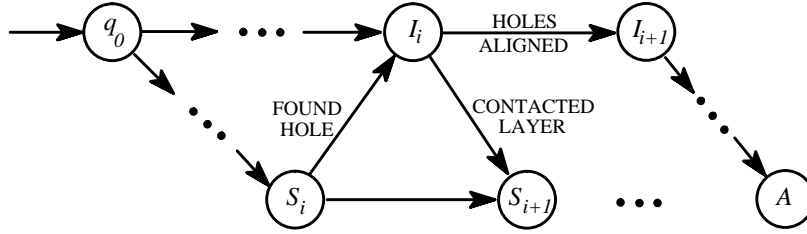


Fig. 3. Hybrid Automaton Representation

continuously runs an event-check routine. The event `FOUND_HOLE` can be distinguished by a sudden drop in the sensed force (along the vertical z). This spike in the sensed force is because the peg, while pushing against the layer, suddenly loses contact when it aligns with the hole.

INSERT states: When the controller is in an INSERT state it makes the peg move down through the hole to contact the next layer (again by moving the attractor). There are 2 events possible in this state. The event `CONTACTED_LAYER` is also distinguished by a sensed-force spike. The sensed force is nearly zero when the peg is not in contact with any layer and is moving down. But when the peg hits a layer, the sensed force suddenly shoots up.

If the holes on the current layer (that the peg is moving through) and the next layer are aligned, then the peg will move through the next layer without contacting it. Another event-check routine is used to signal this event: `HOLES_ALIGNED`. On recognition of this event, the controller shifts to the corresponding INSERT state and the peg continues to move down.

The hybrid automaton representation of this controller is shown in Fig. 3.

3 Detailed Example: Two-Dimensional Peg-in-Maze

A simplified two-dimensional version (Fig. 4) of our prototypical peg-in-maze assembly was used for exploring the simulation and verification of strategies. The problem is reduced by two dimensions with the restriction that the peg can move only in the x or z direction (i.e., no y moves, no rotations). Correspondingly,

the layers can only move with the peg in the x -direction. It is assumed there is no friction, so a layer will not move when the peg moves along the top of its surface.

3.1 Environment

The peg is represented by a fixed point, which is the center point on the bottom of the peg. The parameters for the peg include: x position (px), z position (pz), height (ph), and width (pw). Parameters for each layer are: lxn (offset from $x_{min} = 0$), $lwln$ (width of left shoulder), $lwln$ (width of hole), $lwln$ (width of right shoulder), exn (total eccentricity), lzn (z position of the top surface of the layer), and lhn (height). Here, n represents layer n . lxn should be no less than 0 and no greater than the total eccentricity of the layer (exn). Be aware that although there is no connection shown in Fig. 4 between the left shoulder and right shoulder of a layer, they are connected. So whenever one of them moves, the other moves with it.

3.2 Control Strategies

We build an event-based control strategy as described in Section 2.1, using the two previously identified controlled behaviors. One is to move the peg along the surfaces of layers, searching for holes using a constant velocity (*Search*: x direction movement only, using a constant velocity, v_x). Another is to move the peg down the hole once the hole is found (*Insert*: z direction movement only, using a constant velocity, v_z). We use an additional behavior to stop the movement of the peg once the task is successfully accomplished (the peg hits bottom of the box). Different control behaviors will be used when the peg is in different states. The *Search* behavior is itself made up of two distinct behaviors: *searchright* and *searchleft*. The peg is always made to search towards the right first. If it hits the right barrier without finding the hole, it will start searching to the left.

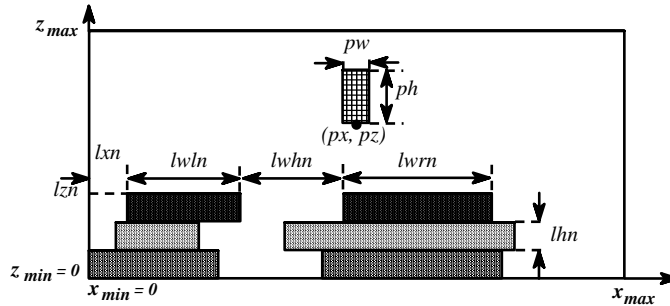


Fig. 4. Two-dimensional Peg-in-Maze Problem

4 Verification of Assembly Strategies

General peg-in-maze problems are usually hard to solve analytically for lack of suitable tools to model both the hybrid nature of the system and underlying uncertainties. Here, we build a framework for computationally testing strategies for peg-in-maze problems [12]. This includes verification to prove properties of a strategy for a whole class of situations. Two software tools have been tried for this purpose: HyTech [13] [14] and CEtool [15].

4.1 HyTech

HyTech is a software tool that can automatically verify a certain class of hybrid systems, known as linear hybrid automata. Specifically, HyTech can symbolically compute the reachability sets of such systems starting from a set of initial conditions (even if the systems are non-deterministic). Thus, one can verify safety properties (e.g., *If the initial conditions are in set INIT, will the set BAD of undesirable states never be reached?*) and liveness properties (e.g., *If the initial conditions are in set INIT, will the state SUCCESS always be reachable?*). HyTech can also perform parametric analysis, in which it solves for constraints on the values of certain design parameters necessary and sufficient for the desired properties to hold.

In order to test control strategies easily, automata for the controller, the peg, and different layers have been separated. The search strategy is formalized in the controller automaton of Fig. 5. Similarly, automata are constructed for the peg and each layer (see [12]). These automata are not trivial. For instance, the peg automaton includes 3 MOVE_DOWN states (to move down to contact the first layer and then insert through the first and second layers), 6 SEARCH_LAYER states (for each layer: searching right on layer left shoulder, right on right shoulder, and left on right shoulder), a FINISHED state, a JAM state and a special CHECK state. The peg automaton enters the CHECK state whenever a movement can not be continued. For example, when the peg moves down and it hits an object it can not move down any further. In this situation, the peg automaton will issue the FORCEZ synchronization label, go to the CHECK state, and wait for feedback from the controller.

Using the strategy given, HyTech was used to evaluate the feasibility of the assembly tasks and calculate bounds on assembly times. Fig. 5 shows assemblies with different part geometry and particular initial conditions. Also shown are verification results on these assemblies.

As mentioned before, HyTech can only handle linear hybrid automata. Also, HyTech's calculation power is limited. Adding layers to HyTech will add more states, finally causing a state explosion that exceeds HyTech's calculation ability. To use HyTech more efficiently, one should try to keep the system description as small as possible. For example, one should share locations wherever possible, minimize the number of variables used, and use the strongest possible invariants. Simplifying the constants used also helps to speed up the analysis: integers should be used whenever possible.

4.2 CEtool

CEtool is a software verification tool developed by a research group at Carnegie Mellon University. It is built on top of Matlab's Simulink toolbox; hence it has a GUI for entering models in a block-diagram fashion.

CEtool can handle so-called threshold-event-driven hybrid systems (TEDHS) because they support block diagram modeling. A TEDHS consists of three interconnected subsystems: (i) switched continuous systems (SCS) with piecewise constant inputs that select the continuous dynamics and continuous outputs, (ii) threshold event generators that take the continuous outputs of switched continuous systems and generate events when they cross certain thresholds, and (iii) finite state machines that are discrete transition systems.

The same control strategy as depicted by the finite automaton of Fig. 5 is encoded in a finite state machine (fsm). The fsm has six event inputs and one data output. The six events are: *START*, *FORCEZ*, *NOFORCEZ*, *HITBOTTOM*, *HITRIGHT*, and *HITLEFT*. The output q is connected to the SCS block. For different q values, the SCS block has different continuous dynamics. There are nine polyhedral threshold blocks (PTHB), $layer_1$, $hole_1$, $layer_2$, $hole_2$, $bottom$, x_{max} , exr , exl , and x_{min} . $layer_1$, $layer_2$, and $bottom$ mark the levels of layer one, two, and the bottom of the box. $hole_1$ and $hole_2$ mark the original positions of the two holes. x_{max} and x_{min} are the limits when the peg is searching for the first hole. exr and exl are the limits when the peg is in the first hole and is searching for the second hole. In each case, their constraints are given as the polyhedra $C\bar{x} \leq d$, where C is a matrix, $\bar{x} = [x \ z]'$, x is the x position of the peg, z is the z position of the peg, and d is a threshold vector.

The verification requirement for the system is $(AF (AG (fsm == FINISH))) \ \& \ (AG \sim fsm == JAM)$, which means the finite state machine will reach the FINISH state and stay there forever, and the JAM state will never be reached.

In CEtool, one can potentially verify bounds on assembly time also. For example, iterating the expression $(AG (\sim (fsm == FINISH \mid timer)))$ can give us a lower bound on assembly time if we include the assembly time t as one of the state variables (with derivative equal to 1 in all modes) and adding a PTHB timer to check whether t is greater than a maximum lower bound. Similarly,

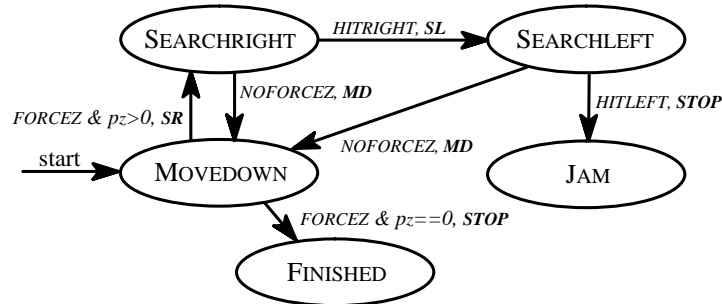


Fig. 5. The Controller Automaton

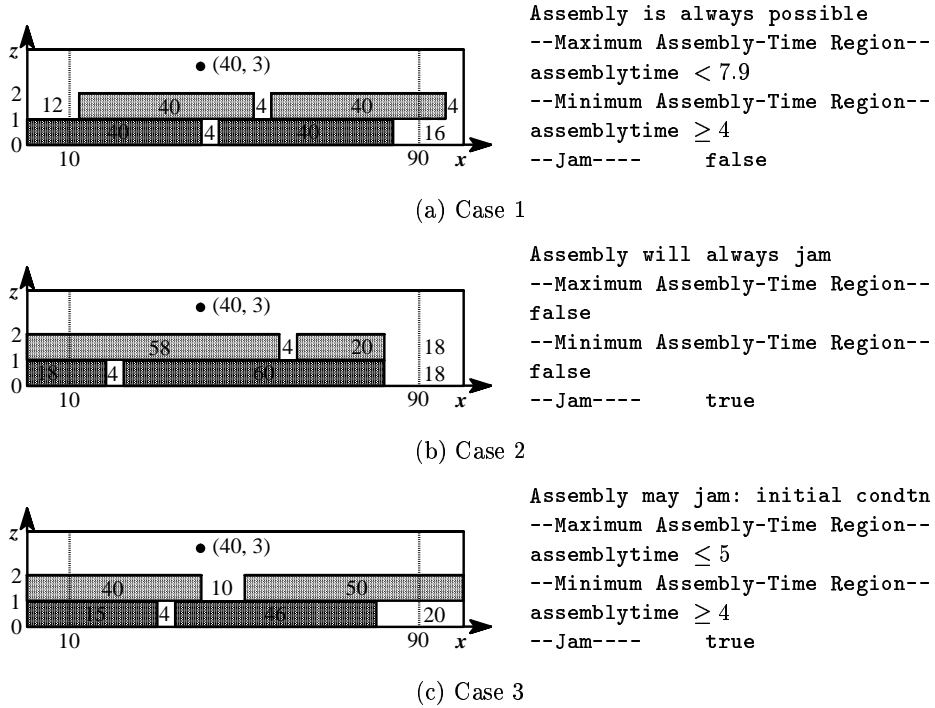


Fig. 6. Verification results for three different cases using HyTech

the expression $(AG (fsm == FINISH \mid timer))$ can be used to give us an upper bound on assembly time if the PTHB checks whether the assembly time t is less than a least upper bound. Unfortunately, CETool did not converge on these two properties for the two-layer peg-in-maze assembly model.

A limitation of CETool (in the version we downloaded, 1998) is that it does not allow the clock derivative constant to be parallel to any hyperplanes specified in the system. Thus, to use the tool to verify the system, we had to use the following *de-parallelization* trick to accommodate this constraint. When moving along the x -axis, for example, instead of setting the z velocity to zero, it was given a small negative value. Similarly, when moving down along the z -axis, instead of setting a zero x -velocity, a small positive value was tried. This latter *fix* caused a problem, however. If the peg finds the right wall of the hole, a NOFORCEZ event will be triggered and the peg goes to the MOVEDOWN state and tries to move down. But there is a small positive x -velocity when moving down, so the peg goes out of the region of the hole immediately and a FORCEZ event is falsely triggered. To solve this problem, the following changes can be made. The MOVEDOWN state in the finite state machine can be split into DOWN1 and DOWN2, and the NOFORCEZ event can likewise be split into two events. The NOFORCEZ1 event is triggered when the first hole is found; the NOFORCEZ2 event, when the second

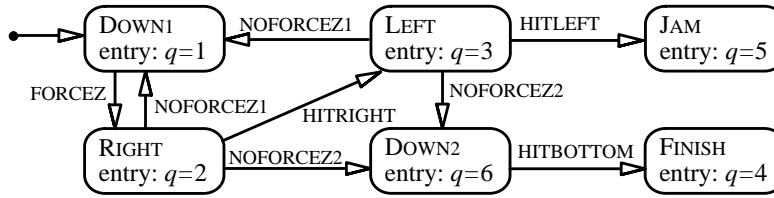


Fig. 7. fsm for verification

hole is found. The initial analysis region should be on the left of the first hole, so the problem will not happen with the first hole.

Another limitation is that CTool does not allow resetting of continuous variables. For example, when the peg hits the layer, the z velocity has to be reset to zero. To model the reset and still keep the variable continuous, we can use another trick: *fast-time scale resetting*. That is, we add new states to the finite state machine to decelerate the velocities to zero quickly.

5 Assembly Strategy Synthesis and Code Generation

Our software synthesis framework is comprised of three levels or layers, as shown in Fig. 8. Each level of the framework addresses a fundamental problem associated with peg-in-maze/robotic assemblies and provides a solution for the same. The layers are implemented on separate platforms, and are independent subsystems by themselves, thus making it a truly heterogeneous system. The layers from bottom to top are explained briefly below. See [16] for more details.

5.1 Robot Client-Server (Lower Layer)

At the lowest level, we tackle the problem of controlling the robot. For this we need a set of commands. Different robot controllers have different command sets, and to compound the problem, the communication media for different robots vary. The framework is designed for a heterogeneous distributed platform with

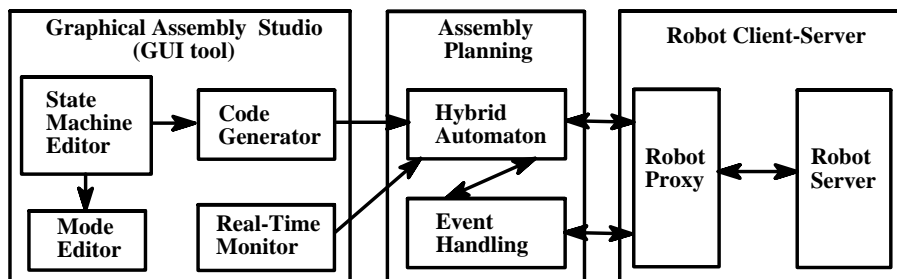


Fig. 8. System Diagram

multiple robots running on different controllers. We chose to follow the client-server paradigm to extend the object-oriented design beyond a single machine. The Robot Client-Server sub-system provides the following: (1) Generic Robot API, a universal command set to control any robot; (2) Robot Proxy, a local agent which acts as surrogate for a remote robot and hides communication details from the client; (3) Communication Architecture to handle different Inter-Process Communication (IPC) mechanisms; (4) Multi-Threaded Robot Server, residing locally in the platform hosting the real robot. The robot server handles multiple requests from more than one client concurrently.

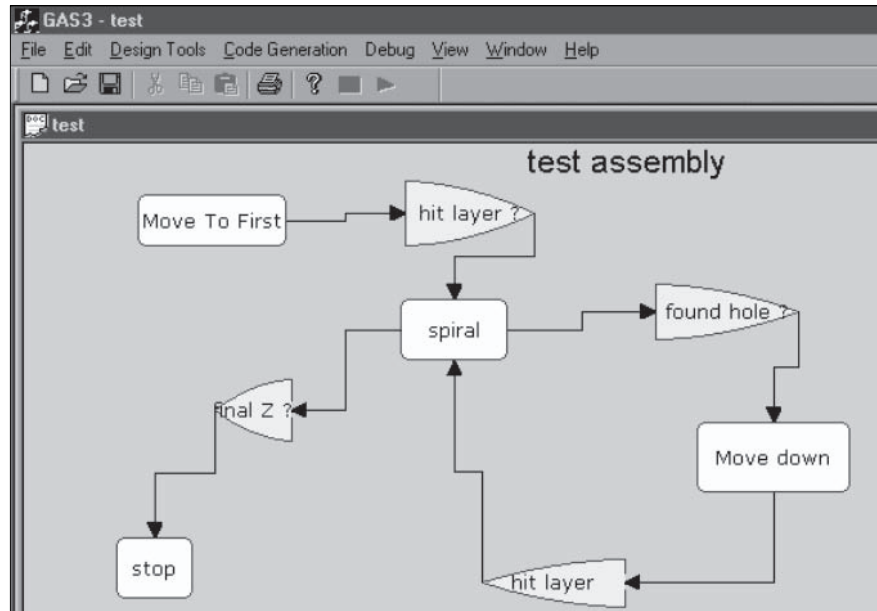
5.2 Assembly Planning (Middle Layer)

Residing right above the Robot Client-Server is the Assembly Planning layer. Peg-in-maze assemblies are modeled as Hybrid Automata, or Hybrid State Machines. Jump conditions in the hybrid automata correspond to event sources in the framework. These events are identified and the transitions are triggered using the Event Handling Framework. Events include device, timer, software interrupts, digital input transitions, force feedback, operator input etc. The Event Handling Framework is based on the Delegation Model, an object oriented model for event handling. The Assembly Planning layer has the software implementation of these state machines. The uppermost layer, the Graphical Assembly Studio, automatically generates the main body of these state machines.

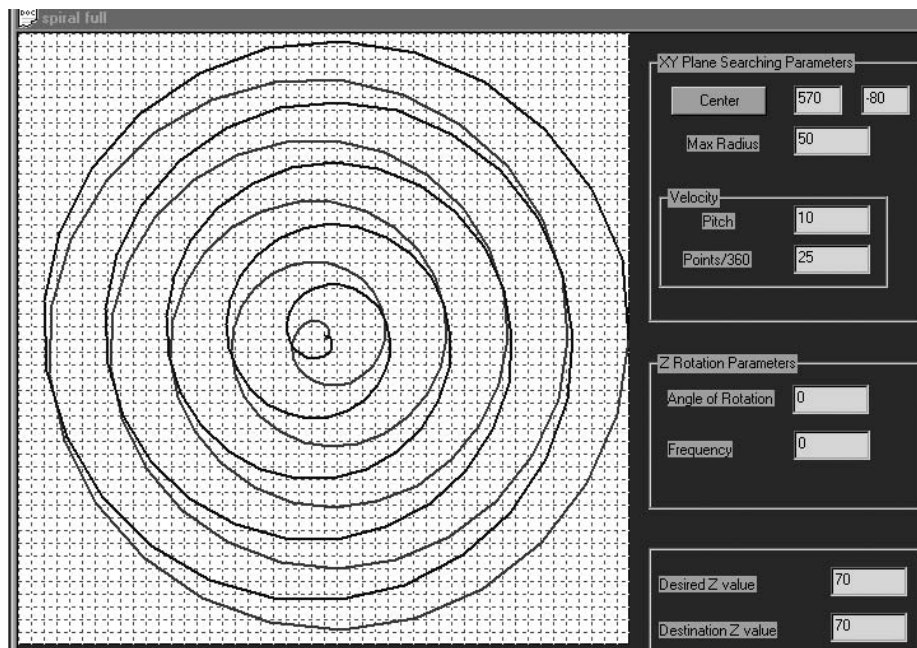
5.3 Graphical Assembly Studio, a GUI Tool (Upper Layer)

At the highest level resides the GUI tool, which we have named Graphical Assembly Studio. See Fig. 9. It comprises four main applications: State Machine Editor, Mode Editor, Code Generator, Real-Time Monitor. The state machine and mode editors are visual editors which allow the user to plan the assembly strategies in a WYSIWYG environment, and save assemblies as persistent objects. The State Machine Editor allows the user to (1) model the whole assembly as a hybrid automaton, (2) associate each location in the automaton with a mode, and (3) select from a library of registered events for transitions between locations. The Mode Editor is used to (1) select and customize predefined search patterns, (2) divide search patterns into neighborhoods and define primitive behaviors within them, (3) specify the subspace of the assembly, (4) specify other parameters specific to the search strategy, and (5) specify certain periodic behaviors, e.g. *jiggle*. The Code Generator parses these graphs to automatically generate C++ code, which forms the body of the state machine. This code becomes part of the middle-layer. The Real-Time Monitor is a useful application that allows the user to monitor assemblies as hybrid automata, when they actually execute. Active states are highlighted; events are visually noted.

Fig. 9(a) shows an assembly strategy for the forward clutch modeled as a hybrid automaton using the State Machine Editor. The rectangular blocks (MOVE_TO_FIRST, SPIRAL, MOVE_DOWN, STOP) are locations and are associated with corresponding strategy modes. The triangular blocks (HIT_LAYER,



(a) Hybrid automaton model of forward clutch-assembly strategy



(b) Defining SPIRAL search mode (attractor trajectory and impedance controller parameters) using the Mode Editor

Fig. 9. Graphical Assembly Studio example screen captures

FOUND_HOLE, FINAL_Z) are transitions between the locations, and are selected from a library of registered events. A double-click on a location or a transition opens up the corresponding mode or event-definition for editing. For example, selecting the SPIRAL block would open up the Mode Editor shown in Fig. 9(b).

We used these tools to perform both the prototype peg-in-maze assembly and our example transmission assemblies [16].

6 Results

The State Machine Editor and Mode Editor helped specify the hybrid automaton model quickly. Once the finite state machine has been entered and the specific behaviors have been attached to the corresponding states, the Code Generator produces C++ code for the strategy, ready to be used in actual assembly.

Assemblies were conducted successfully using constructed strategies modeled as hybrid automata. Real-world example transmission assemblies (forward clutch and several others) were performed with average assembly times comparable to manual assembly (for more details, see [3] [4] [5]).

With HyTech, linear hybrid automata were used to model the controller, the peg, and each layer. The possibility of performing assembly tasks was evaluated under a variety of different geometries for whole sets of initial conditions (which encompassed all possible, a priori unknown, locations of the layers). Upper and lower bounds on assembly time were calculated. Also, velocity uncertainty was introduced and different strategies were tried. Finally, clock translation and linear phase-portrait approximation methods were used to convert nonlinear (force control) hybrid automata to linear hybrid automata (see [12]).

CEtool was used to simulate and verify peg-in-maze assemblies using both velocity and force control. The tool has limitations when applying it directly to peg-in-maze assemblies, but after modifications (*de-parallelization*, *fast-time scale resetting*), it dealt with the system correctly and gave satisfying results.

7 Future Work

In relation to verification of strategies, future work consists of extensions to higher dimensions, more realistic modeling, development of special purpose verification code, and incorporation of our hybrid systems modeling paradigm into control software tools. Due to the limitations of the general hybrid systems tools we used, a possible alternative is to write specific programs in a general language (e.g. C++) to verify the specific assemblies that one wants to solve. For verification, (in the two-dimensional case) a specialized analysis program would work as follows. For layer i (cf. Fig. 4), the peg can always find the hole if $pw < h_i$ and $l_i + h_i \geq l_{max}$ and $x_{min} + e_i + l_i \leq x_{max} - r_{max}$. If this condition is not satisfied, the peg can sometimes find the hole (i.e., for fortuitous initial conditions of the layers) if $pw < h_i$ and $x_{max} - r_i \geq x_{min} + l_{max}$ and $x_{min} + l_i \leq x_{max} - r_{max}$. Otherwise, the assembly is never possible. The conditions can be interpreted as follows: the peg can always find a hole if the hole is within the search limit of

the peg, and the peg can sometimes find a hole if it is possible that the hole can be inside the search limit of the peg. The verification program was used to verify the three cases shown in Fig. 6(a,b,c) and gave the same results. More generally, one can use verification techniques based on linear programming.

Acknowledgements: NSF (DMI97-20309) and NIST (70NANB7H3024).

References

1. Branicky, M.S.: *Studies in Hybrid Systems: Modeling, Analysis, and Control*. Ph.D. Thesis, Electrical Engineering and Computer Science, M.I.T. (1995).
2. McCarragher, B.J., Asada, H.: The Discrete Event Modeling and Trajectory Planning of Robotic Assembly Tasks. *ASME J. of Dynamic Systems, Measurement and Control*, 117(3):394–400, (1995).
3. Newman, W.S., Branicky, M.S., Podgurski, H.A., Chhatpar, S.R., Huang, L., Swaminathan, J., Zhang, H.: Force-Responsive Robotic Assembly of Transmission Components. *Proc. IEEE Intl. Conf. Robotics & Automation*, Detroit, MI, pp. 2096–2102, (1999).
4. Newman, W.S., Branicky, M.S., Chhatpar, S.R., Huang, L., Zhang, H.: Impedance Based Assembly. *Video Proc. IEEE Intl. Conf. Robotics & Automation*, (1999).
5. Chhatpar, S.R.: *Experiments in Force-Guided Robotic Assembly*. M.S. Thesis, Electrical Engineering and Applied Physics, Case Western Reserve Univ. (1999).
6. Newman, W.S.: Stability and Performance Limits of Interaction Controllers. *ASME J. of Dynamic Systems, Measurement and Control*, 114(4):563–570, (1992).
7. Ramadge, P.J.G., Wonham, W.M.: The Control of Discrete Event Systems. *Proc. of the IEEE*, 77(1):81–98, (1989).
8. Chhatpar, S.R., Branicky, M.S.: A Hybrid Systems Approach to Force-Guided Robotic Assemblies. *Proc. IEEE Intl. Symp. on Assembly and Task Planning*, pp. 301–306, Porto, Portugal, (1999).
9. Branicky, M.S., Chhatpar, S.R.: A Computational Framework for the Simulation, Verification, and Synthesis of Force-Guided Robotic Assembly Strategies. *Proc. IEEE/RSJ Intl. Conf. Intelligent Robots & Systems*, pp. 1465–1470, Maui, (2001).
10. Henzinger, T.A.: The Theory of Hybrid Automata. *Proc. IEEE Symp. on Logic in Computer Science*, New Brunswick, (1996).
11. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, (1987).
12. Fu, Q.: *Simulation and Verification of Robotic Assembly Tasks*. M.S. Thesis, Electrical Engineering and Computer Science, Case Western Reserve Univ. (1999).
13. Henzinger, T.A., Ho, P-H., Toi, H.W.: HyTech: A Model Checker for Hybrid Systems. *Proc. Ninth Intl. Conf. on Computer-Aided Verification*, 1997. Lecture Notes in Computer Science, Vol. 1254. Springer, Berlin (1997) 460–463.
14. Henzinger, T.A., Ho, P-H., Toi, H.W.: HyTech: The Next Generation. *Proc. IEEE Real-time Systems Symp.*, pp. 56–65, (1995).
15. Chutinan, A.: *Hybrid System Verification Using Discrete Model Approximation*. Ph.D. Thesis, Electrical and Computer Engineering, Carnegie Mellon U. (1999).
16. Swaminathan, J.: *An Object-Oriented Application Framework for Peg-in-Maze Assemblies*. M.S. Thesis, Electrical Engineering and Computer Science, Case Western Reserve Univ. (1999).