Hybrid System Modeling, Control, and Fault Diagnosis on a Three Tank Testbed

Nathan Allotey Brian Turnbull

August 4, 2006

Abstract

For this project, we studied modeling, simulation, control and fault diagnosis of a hybrid system using a three tank testbed. To facilitate our project, we developed software for interfacing the laboratory testbed with other software components. The system was modeled using hybrid bond graphs and the model was implemented using Matlab/Simulink. To validate the model, we compared the simulated output to the testbed's behavior using several control sequences and computed the average absolute error for each. With the validated model, we then implemented a set-point controller for the testbed using a limited-lookahead approach. The controller's behavior was verified first on the simulation and then on the testbed. Finally, we integrated the testbed with the Fault Adaptive Control Technology (FACT) software, developed at the Institute for Software Integrated Systems (Vanderbilt University), to detect and identify faults in the system. The results show that the model-based controller effectively maintains the system state around the set points, and the FACT Diagnoser can be used to successfully detect and identify different failure modes of the testbed.

1 Introduction

This project is a study of model-based control and diagnosis of hybrid systems. More specifically, we are looking at these concepts as they apply to a three-tank testbed. The focus was twofold: build model-based controller(s) that run on the actual system and integrate a diagnosis engine to detect, isolate, and identify system faults. The first step in both cases is the development of an accurate model. We chose to build our model using hybrid bond graphs (HBG) because they represent the combination of discrete and continuous behavior found in hybrid systems such as our testbed. From the HBG, we then used a model interpreter to generate a Simulink model. We built a simple m-file model using the discrete-time equations of our modes of interest and also built a Simulink block model. During this time, we developed a package of software which can be used to collect data and control the testbed. Real data collected from the testbed was used to validate the model. The valid model was then used as a basis for a set-point controller using a limited-lookahead approach. The controller was tested first on a Matlab simulation of the system and eventually run in real-time using the developed software. At the same time, the FACT Diagnoser was integrated with the testbed. After tuning the Diagnoser's hybrid observer offline with collected data, it was tested in real-time on the system. Finally, the Diagnoser was used to detect and quantitatively estimate several classes of system faults.

2 Three Tank Testbed

Our testbed is a fluid system composed of three identical tanks. The first two tanks are connected to a variable speed pump. In addition, the three tanks are connected to a common transfer pipe and a common drain pipe. A loopback pipe is included which allows the pump flow to be enabled or disabled without needing to shutdown and restart the pump. Nine solenoid-operated valves control the transfer of water through each of the paths, and three ultrasonic level sensors provide measurements of the water level in each tank.

Four independent nodes provide distributed monitoring and control for the system. Each node is managed by a Network Capable Application Processor (NCAP) which provides an HTTP-based API for querying information about and sending commands to the system's transducers. A Smart Transducer Interface Module (STIM) at each node interfaces with the node's transducers and its NCAP using the IEEE 1451.2 standard for transducer to microprocessor communication. This specification includes an electrical interface along with read/write functions for accessing the data from the transducers[1].

3 Interfacing With the Testbed

Before we could begin experiments on the testbed, we needed software to communicate with the nodes. Since no package existed, we developed a collection of applications and libraries to fulfill these requirements:

1. Reading/Writing transducers
2. Sampling and logging data to a file
3. Distributing data to multiple components (i.e. controller, Diagnoser, and logger)
4. Receiving distributed data
5. Framework for integrating control algorithms

The software was written using Windows sockets for communication with the nodes. This aspect of the project was a continuation of work which started in January of 2006 as an independent study project. In order to interface various applications with the testbed, the package provides reusable classes in dynamic link libraries, and the entire package can be configured from files. The software is capable of reading from an arbitrary number of nodes/channels on the testbed. For example, diagnosis experiments could be run on a single tank by specifying the tank in a configuration file.

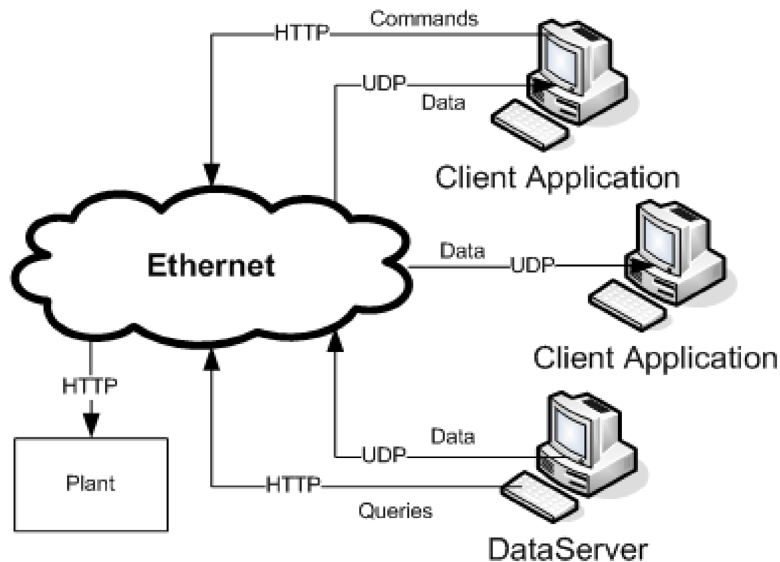

Figure 1: Diagram of software communication architecture.

3.1 ThreeTankInterface

The interface class provides access to the testbed at the lowest level. It provides information about the system and its configuration, and it includes methods for reading/writing (raw text) and polling at a configurable rate. In the interest of efficiency, the polling algorithm takes advantage of the NCAP's

"PushData" functionality. This extension of HTTP is not a true persistent connection, but does allow data to be collected multiple times without reconnecting to the node.

3.2 ThreeTankDataProcessor

The DataProcessor class wraps the interface class to provide applications with binary datasets from the system at a constant rate. It is derived from an abstract DataSource class to allow applications to be written independent of the source (local/remote).

3.3 ThreeTankClient/Server

In order to allow multiple components to use data from the testbed, we needed a method to efficiently distribute this data. The client and server classes can be used to stream the data from a single server to an unlimited number of clients using a UDP multicast. This method was chosen because the testbed is on a dedicated network, and this method requires little overhead on the server side.

3.4 ThreeTankController

This base class provides an abstract interface for system controllers. Sample applications were written to illustrate different controller design approaches. The class includes read/write methods, and defined constants provide a standard way for addressing specific transducers and nodes.

3.5 Applications

In addition to the libraries, we included three applications which implement the logging and distribution functionality mentioned previously. The `Logger` can either read data from its own Processor object or receive data from an existing multicast. It logs to a CSV file with labels for the columns. Controllers can be run using the `Controller` application. It loads the controller as a dynamic link library and feeds its data from either a local or remote source. Controllers are developed from a common base class and can be switched without reconfiguration. Finally, the `Server` simply reads data from the testbed using a DataProcessor object and distributes it in a UDP multicast.

4 Modeling the Testbed

4.1 Hybrid Bond Graphs

Bond graphs are energy based diagrams which capture the energy structure of the system and give a concise description of the systems behavior [2]. Bond graphs can be applied to electrical, hydraulic, mechanical and other physical systems. They represent the various elements of the system as resistive (R), capacitive(C), inertial (I) components, and sources of flow and effort (S_f, S_e). In order to model the discontinuous mode changes of a hybrid system, hybrid bond graphs expand the model to include the notion of idealized switching junctions[6]. In our system, the switching junctions are used to represent the valves which determine the flow of water through the system. For each distinct mode of the system the HBG simplifies to a bond graph. Following a standard process, the bond graph can be used to generate a set of continuous time equations[4]. From the continuous time equations, discrete time equations can be derived using standard discretization procedure.

4.2 Implementing the Model

The hybrid bond graph of the three tank system was put directly into GME using the FACT paradigm. The valves were represented by switched junctions. In Figure 3, (R)s represent the pipe resistances and (C)s represent the capacity of the tanks. The pump is modeled as an ideal source of flow and the common pressure and flow points of the testbed are represented by 0 and 1 junctions respectively. Modulated functions (ModFunctionR13, ModFunctionR23, and ModFuncDrain3) were used to model the nonlinear pipe resistances. Each function contains a polynomial determined through experimentation.

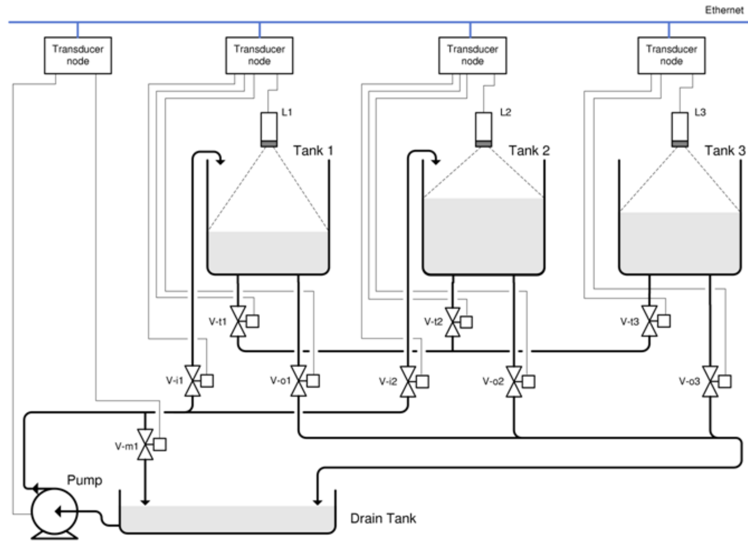

Figure 2: Diagram of the testbed's physical layout.

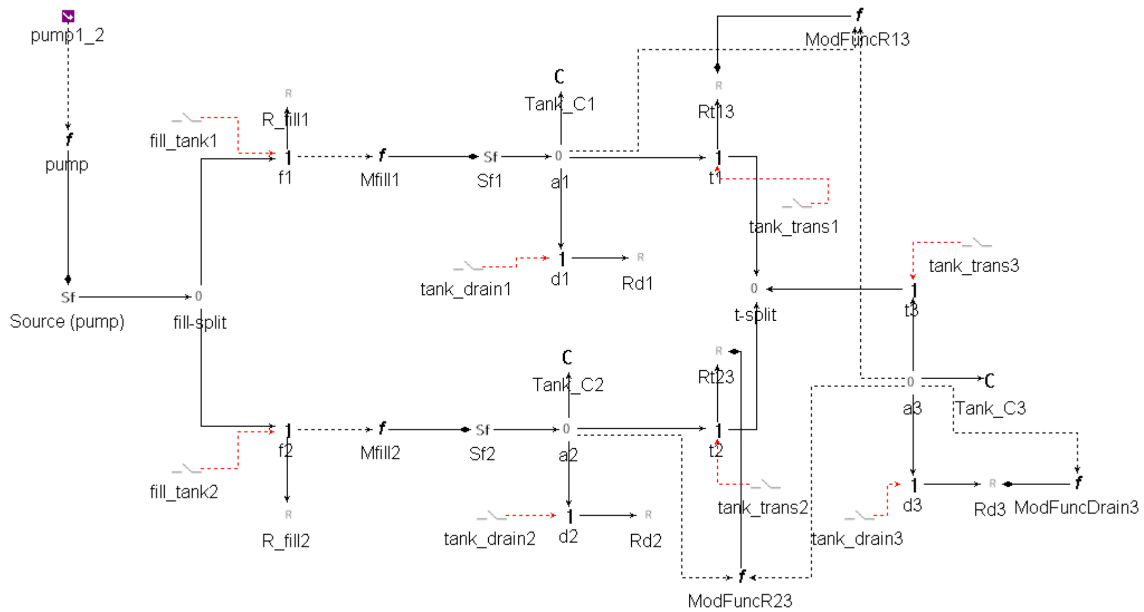

Figure 3: Hybrid bond graph of the testbed.

The complete model was then fed into an interpreter which converted it into a ready to use Simulink model. By using GME to create the bond graph we were able to generate a Simulink model quickly and directly from the HBG. However, one of the disadvantages to this is that the internals of the generated model are not easily understood, and making adjustments to this model's parameters can be difficult because of the complexity of the model layout. This led us to build a Simulink block model manually. We also used an m-file to test the discrete time equations which we obtained from the HBG.

4.3 Parameter Estimation

Before any of the models could be useful, we needed to estimate the parameter values. Each resistance and flow rate on the HBG must be determined from experimental data. To accomplish this, we used the developed software to collect data for each of the given modes. Using Matlab fitting functions, we estimated the parameters of interest as polynomials in terms of the tank heights. For example, for calculating the drain resistance of tank 3, we collected the height data (Figure 4) while draining tank 3.

$$R = \frac{-P}{C \cdot \frac{dP}{dt}} \tag{1}$$

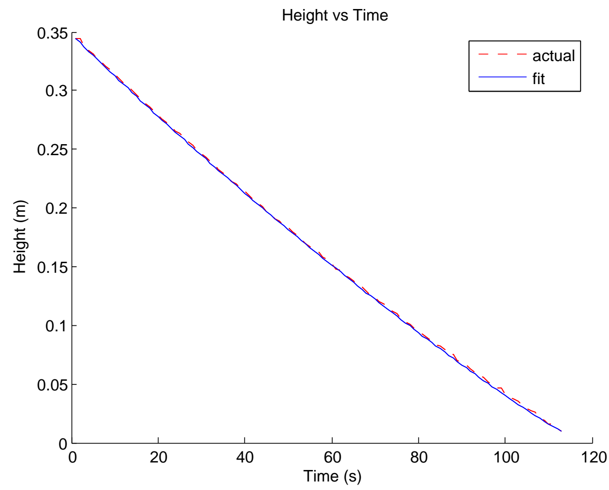

Figure 4: Actual height and simulated height.

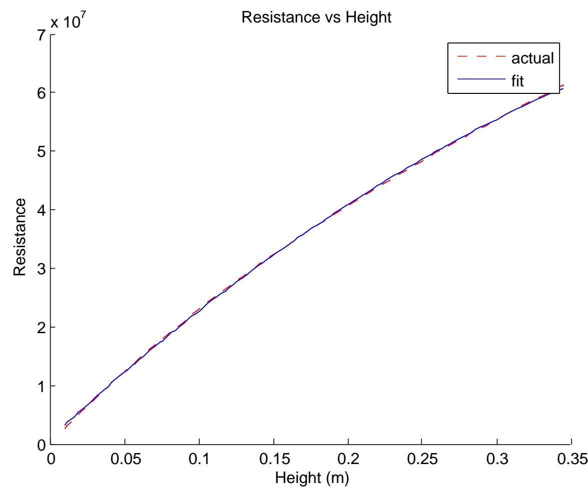

Figure 5: Calculated resistance and its 2nd order fit.

Using the relationship from the bond graph (Figure 1) we calculated the resistance for each height (Figure 5). Using Matlab, we determined the best-fit quadratic (Figure 5) which was then used as the resistance parameter. To verify the accuracy of this fit, we used the equation to calculate height values given the initial height. This result was then compared to the actual height values produced by the system as shown in Figure 4.

5 Validating the Model

In order to determine the accuracy of the model, we selected several multi-mode control sequences. These were run on the model and on the testbed. The first control sequence (Figure 7) consisted of filling tank 1 to a certain height at the high pump speed and then transferring the water to tank 3. This process was then repeated. The second control sequence (Figure 6) consisted of filling both tanks and transferring the water to tank 3. In this experiment both of the tanks were filled at the same time and then tank 1 was transferred to tank 3, followed by tank 2. While tank 2 was transferring, tank 1 began filling. This experiment included nine distinct modes.

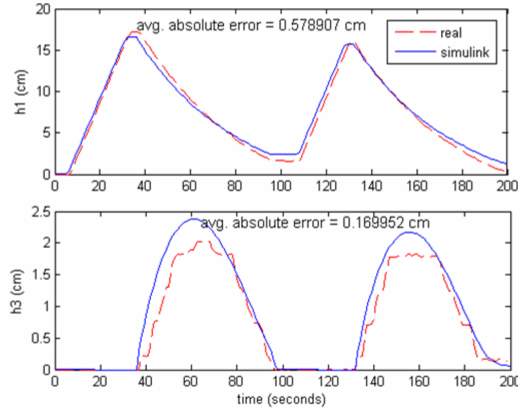

Figure 6: Validating the model with the first control sequence.

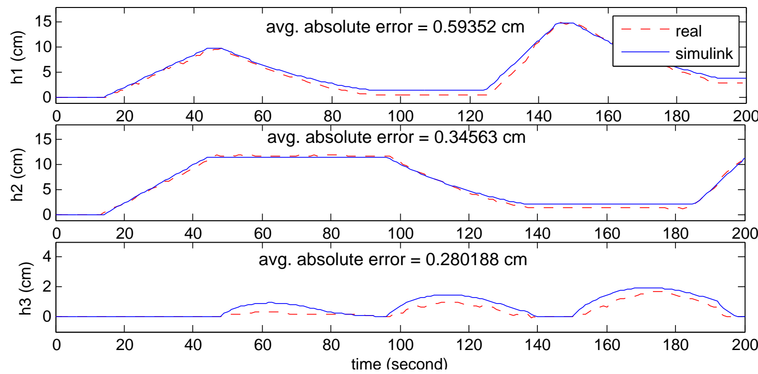

Figure 7: Validating the model with the second control sequence.

6 Model Based Control

Using the validated model, we implemented a set-point controller that utilized a limited-lookahead approach. Using the relationships from the model, the controller first predicts the possible future states of the system over a horizon of all possible control sequences. By minimizing the cost function, the controller chooses the trajectory which gives the lowest cost values. Cost represents deviation from the desired set points. The first element of the resulting control sequence is then applied. The controller accepts data from the client libraries at a rate of 3Hz.

By modeling the three tank testbed in Simulink we were able to simulate the controller on a model, rather than to construct the controller and run "trial and error" tests on the physical system. This provided a clear view of the controller's expected behavior. The controller was first implemented in Matlab with the model developed earlier. Figure 8 shows the results of the controller when run on the Simulink block model. With this working properly, we integrated the controller with the other libraries developed for this project to run in real-time.

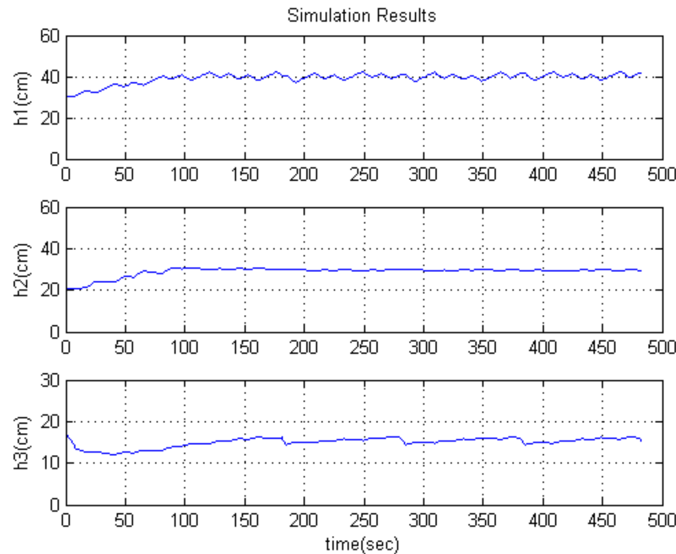

Figure 8: Simulated results for setpoints of 40cm, 30cm, and 15cm.

In order to monitor the system in real-time, the graphical interface designed for the the Diagnoser (discussed later) was used. Figure 9 shows the controller's results being displayed in real-time. In "Online Monitoring" mode, individual plots show the state of each tank. The controller was tested by starting at different starting points to verify that the heights in each tank approach and maintain the desired values.

7 Fault Diagnosis

The FACT Diagnoser is a qualitative diagnosis engine for hybrid systems developed by the Institute for Software Integrated Systems at Vanderbilt University. It uses an annotated hybrid bond graph model to implement a hybrid observer which tracks the system's behavior. In order to account for small deviations and refine the model's predictions, the hybrid observer uses a Kalman filter[5]. Deviations of model predictions from the actual values are analyzed by the fault detector. The fault detector uses statistical methods to differentiate between model error and system faults.

When a fault is detected, the Diagnoser uses relationships in the model to back-propagate through possible causes (components). Each possible component is recorded as a fault candidate with its associated qualitative change (+/-). The Diagnoser then looks at incoming data and attempts to revise the

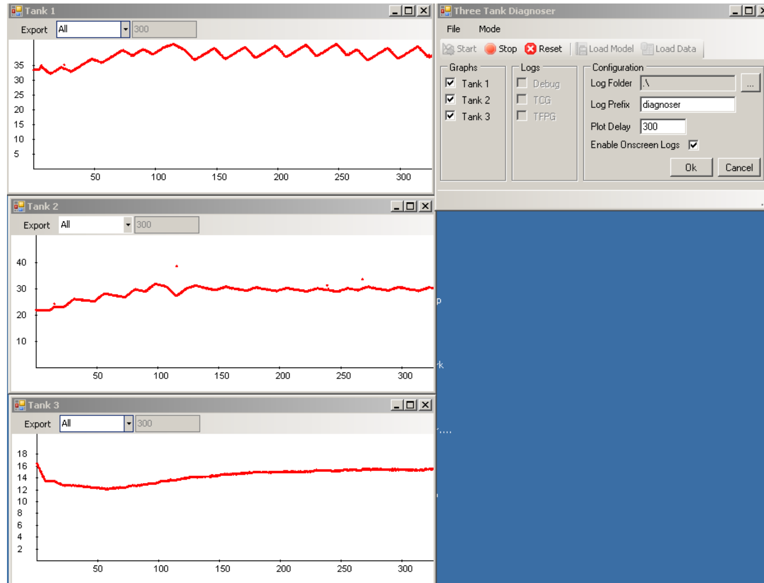

Figure 9: Real-time display of controller experiment.

candidate list based on the system's behavior. The process is described with more detail in [3]. This continues for a configurable time (200 samples for our implementation). The Diagnoser then initiates a parameter estimation process for each of the remaining candidates. During the parameter estimation, the Diagnoser attempts to quantitatively estimate the value of each change as a scale factor. The estimated values are then applied to the actual data collected. Error calculations are performed for each possibility, and the candidate with the minimal error is selected. This parameter is updated in the observer, and it attempts to track the system. If precision of the parameter estimation is lacking, the fault could be identified again and the process would continue. Successful parameter estimation allows the hybrid observer to track the faulty behavior.

7.1 Diagnosis Model

The model used by the Diagnoser was an extension of the HBG model used for the control aspects of this project. The model (Figure 10) was created and tuned with the help of Mr. Nagabhushan Mahadevan of ISIS. The implementation of this model is beyond the scope of this project, but there were some specific issues that needed to be resolved before the model could be used for diagnosis experiments. First, the model needed to account for filling delays present in the tank. These delays are caused by the filling valves being located under the tanks with sections of pipe on either side. During initial filling, the entire pipe must be filled before the water reaches the tank, and on subsequent fillings, the pipe up to the valve must be filled. These properties introduce a variable delay which, depending on the pump speed, can last from five to thirty seconds. The delay on each pipe is modeled as a capacitance which fills when filling is enabled on a particular tank. Filling is disabled via a switched junction while this capacitance is below a particular level. The specific parameter for this level was based on the delay times observed in experimental data.

In addition to the physics of the system, the model incorporates information about failure modes. These "Fault Detectors" must be tuned to recognize actual faults, but not modeling error. They were adjusted along with the model until the Diagnoser was able to track offline without detecting a fault due to modeling error.

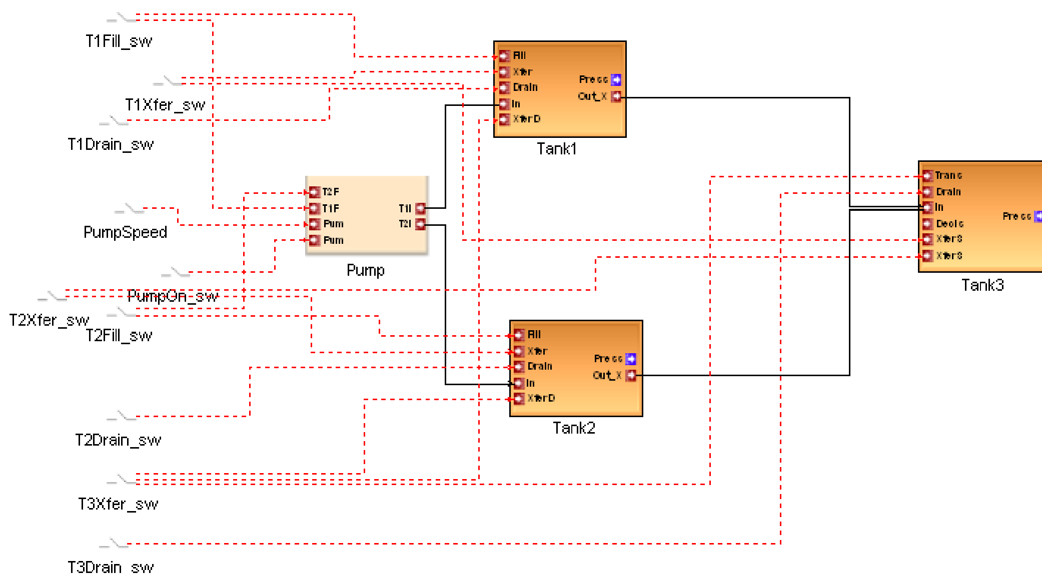

Figure 10: High-level view of Hybrid Bond Graph model used for diagnosis.

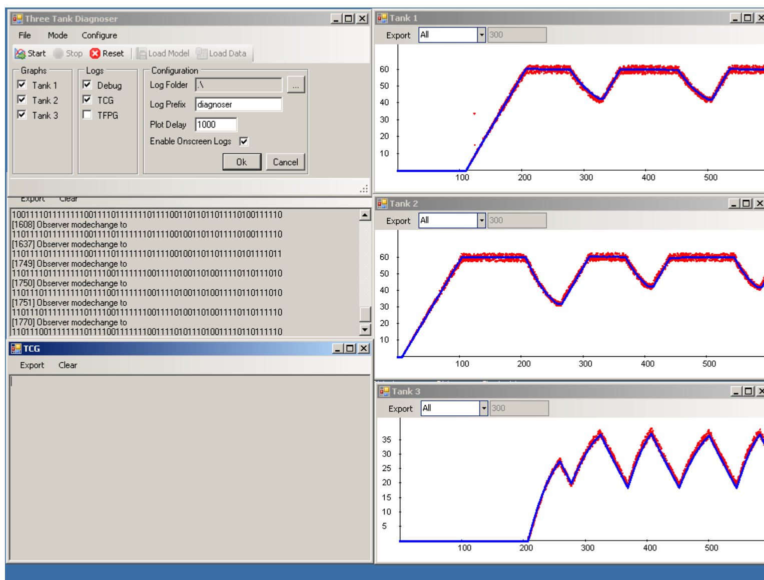

Figure 11: Diagnoser GUI tracking the nominal system in realtime.

7.2 Diagnoser Integration

While tuning the model and fault detectors, a reusable class ("ThreeTankDiagnoser") was developed which integrated the Diagnoser with the data collection libraries developed in this project. From this class, a command line application was written in C++ which provided a simple user interface to start and stop the Diagnoser. Once this was working online, a graphical application(Figure 11) was developed to provide various functions related to the system. In order to build the interface in C#, wrappers

were created for the ThreeTankDiagnoser class. These wrappers were developed using SWIG (Software Wrapper and Interface Generator), a GPL application which automates the generation of wrappers for C/C++ applications in various other languages. The GUI operates in 4 modes:

1. Online Monitoring - Live monitoring and display of system state
2. Offline Monitoring - Plots log files collected using the other applications
3. Online Diagnosis - Monitors the system in real-time while plotting model values as well
4. Offline Diagnosis - Runs data from log files through the Diagnoser and plots the results.

Data is plotted on the screen, and is exportable in both the graphical format and as a csv file. The interface also provides real-time access to the Diagnoser's logs when it is running.

7.3 Testing the Diagnoser

A simple controller was written for testing the Diagnoser. The controller was written using the framework developed as part of this project. The control algorithm was designed to operate only one process on a given tank at once (i.e. no filling and draining tank 1 at the same time). This was done to simplify controller and Diagnoser testing. The control algorithm consists of the following steps:

1. Fill Tank 2 to 60cm
2. Fill Tank 1 to 60cm
3. Transfer Tank 2 to Tank 3 (until difference < 1cm)
4. Drain Tank 3 (until < 20cm)
5. Transfer Tank 1 to Tank 3 (until difference < 1cm)
6. Drain Tank 3 (until < 20cm)
7. Repeat from stage 3.

Each tank fills at a constant speed starting when its transfer completes and ending when its height reaches 60cm or transfer completes with the other tank.

8 Experimental Results

To determine how accurate our model was we computed the average absolute error relative to the testbed results. For both control sequences, we found the average absolute error to be less than 0.6cm (Figures 7) and 6. With the validated model, we moved on to model-based control and diagnosis experiments.

The controller was run in several real-time experiments to verify its performance. The controller's objective is to maintain the water levels in tanks 1, 2 and 3, at 40cm, 30cm, and 15cm respectively. The data was logged in real-time. The mean values and standard deviations were calculated for each tank starting when they first reached their desired heights. The low standard deviations show the controller's ability to maintain the desired heights. The data was plotted using Matlab for clarity (Figures 12 and 13).

	Mean	Std. Deviation
Tank 1	40.203cm	1.018cm
Tank 2	30.061cm	1.063cm
Tank 3	15.998cm	0.582cm

Table 1: Real-time limited-lookahead controller results (after initialization)

Diagnosis experiments were run with three classes of faults: transfer resistance, drain resistance, and instantaneous capacitance changes. The transfer resistance fault was introduced by partially closing the manual valve associated with tank 1's transfer pipe. This caused deviations in measurements for both tank 1 and tank 2. The time required to detect the fault depends on when it is introduced. Due to the nature of the testing controller, particular faults only manifest during certain parts of the control

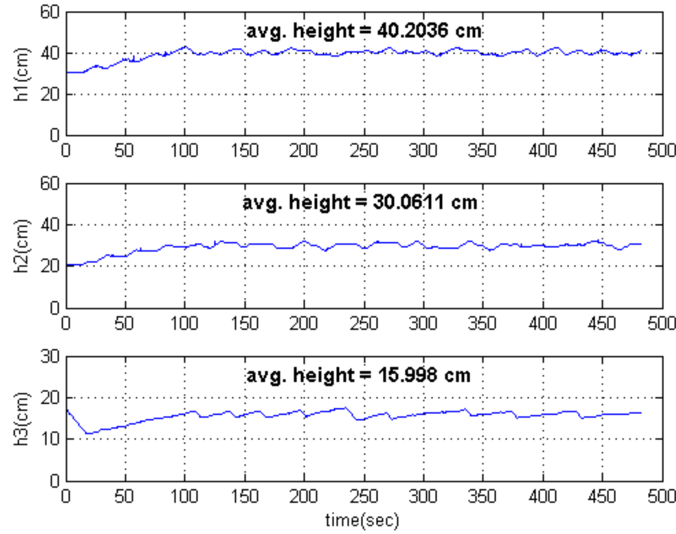

Figure 12: Water levels in each tank during a real-time experiment.

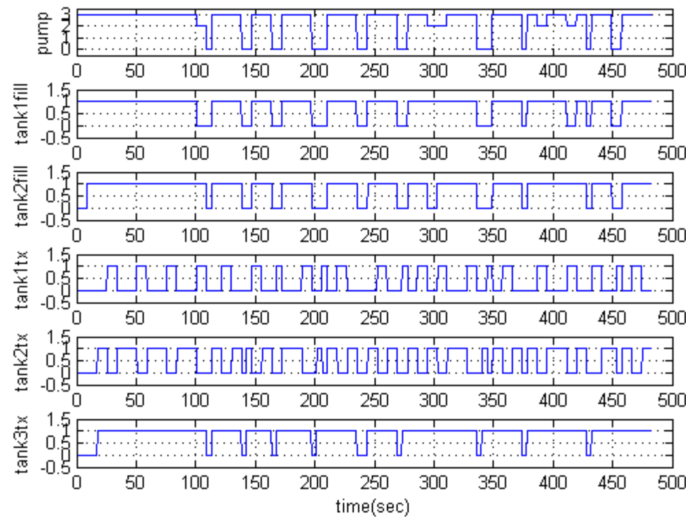

Figure 13: Controller mode changes during a real-time experiment.

sequence (in this case, because transfers only occur during certain stages). If the fault occurs near the beginning of the transfer stage, the Diagnoser identifies the fault within approximately 10 seconds. However, if the fault is introduced later in the sequence, it may not be detected until the next transfer. With the transfer resistance fault, the Diagnoser is unable to precisely estimate the new resistance on the first detection. Because the observer is unable to track with the new parameter, a fault is detected on the next transfer. The process repeats, but on the second attempt, the Diagnoser is able to estimate well enough to allow the observer to track the fault for the remainder of the experiment.

Similar results were obtained for faults caused by a leak in tanks 1 or 2. In these cases, the drain

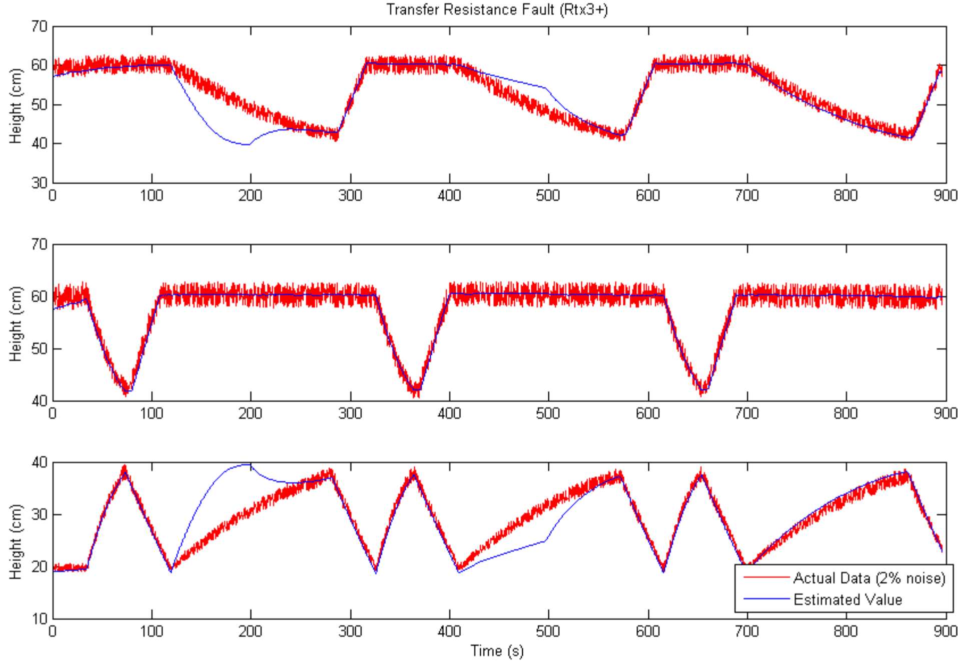

Figure 14: Hybrid observer tracking through fault detection and parameter estimation (redrawn in Matlab).

valve was opened on the given tank during the experiment. In both cases, the Diagnoser identifies the fault and is able to estimate well enough for tracking on the first attempt.

Finally, we tested a capacitance fault in tank 3. This was caused by dropping an object into the tank. In this situation, the Diagnoser was again able to detect the fault. However, it was unable to correctly identify the fault as a capacitance change. Instead, the fault was identified as a drain resistance increase.

	Fault Time	Detection Time
Transfer 1-3 (R+)	119.2407	131.88
Tank 2 Drain (R-)	100.346	178.508
Tank 3 (C-)	92.9176	98.91

Table 2: Fault manifestation and detection times (in seconds) for three failure scenarios.

9 Problems

In this system, the drain and transfer resistances are estimated as cubic and quadratic functions respectively. On the other hand, the Diagnoser uses a linear scale factor to estimate the fault coefficient. This limits its ability to estimate these faults accurately. With the Kalman filter, the hybrid observer can track these estimated parameters, but truly precise estimation will not be possible without considering the parameters as the appropriate nonlinear functions.

One of our ultimate goals of this project was to integrate the limited-lookahead controller with the fault diagnosis to implement a fault-adaptive control scheme. However we found there are problems with either our model or the low level data collection. The problem manifested itself when we attempted to track a system which constantly drains from tank 3. In this situation, the model fails to track tank 3

	Nominal	Faulty	First Estimation	Second Estimation
Tank 1	0.4173cm	5.7250cm	2.8112cm	0.9488cm
Tank 2	0.0156cm	0.2079cm	0.0978cm	0.0394cm
Tank 3	0.8357cm	5.0180cm	3.2018cm	0.8451cm

Table 3: Average absolute error tracking transfer between tanks 1 and 3 before, during, and after a fault

correctly. This issue was somewhat hidden as we used a simple controller for initial testing. The problem persisted over several parameter estimations, and thus does not appear to be a modeling problem. At this point, we have not identified the root cause of this problem, but we do have some theories. The most likely cause is related to previous work done on the system. During initial development of our software, we ran into an issue with the the third tank's response time. In every query, tank 3 responds approximately 200ms later than the other two tanks. The software was adjusted to account for this time such that the data points received from the NCAPs for a particular timestamp were very close. However, in light of this problem, we suspect that this may be evidence of a more significant problem with tank 3. In the recent past, flow sensors were added to tank 3, and the STIM code was updated to read and process these sensors. While these are not currently functional, the NCAP is still looking for results. This could cause some delay in processing. In addition, as equipment was added to this system, things were added to the same power supply. It is possible that the power supply has become overloaded with the extra equipment added and this is now badly affecting the timing and performance of tank 3's transducers or its STIM/NCAP.

10 Conclusions and Future Work

The developed software libraries provide an interface for easily integrating controllers and other components with the testbed. Through validation we have shown that the final model is sufficiently accurate for use with the control and diagnosis applications. Results show that the limited-lookahead controller maintains the given set points, and that the Diagnoser successfully identifies and estimates the quantitative value of several classes of system faults.

In the future, the tracking problem needs to be fixed by investigating the timing issue with tank 3 and a more advanced parameter estimation scheme could be considered to replace the current linear process. Then fault adaptive control would be a logical next step for this research. The controller could use fault estimation data generated by the Diagnoser to explicitly adjust its model or control algorithms. There is also much more fault diagnosis work to be done. Experiments need to be performed on other failure scenarios with less significant changes. It is also possible to investigate diagnosis of intermittent faults by extending the software developed in this project.

References

[1] J. Lyons. Distributed monitoring and control and physical system modeling for a laboratory three tank-system. Master's thesis, Dept. Electrical Engineering, Vanderbilt University, 2004.

[2] P. J. Mosterman and G. Biswas. Model based diagnosis of dynamic systems. *Seventh Journees du L.I.P.N.*, pages 143–154, September 1997.

[3] P. J. Mosterman and G. Biswas. Diagnosis of continuous valued systems in transient operating regions. In *IEEE Trans. on Systems, Man, and Cybernetics*, pages 554–565, November 1999.

[4] R. C. Rosenberg and D. C. Karnopp. *Introduction to Physical System Dynamics.* McGraw-Hill, New York, 1983.

[5] G. Welch and G. Bishop. An introduction to the kalman filter. *Technical Report TR 95-041, University of North Carolina, Department of Computer Science*, 1995.

[6] J. Wu, G. Biswas, S. Abdelwahed, and E. Manders. A hybrid control system design and implementation for a three tank testbed. In *Proceedings of the 2005 IEEE Conference on Control Applications*, pages 645–650, August 2005.