REPORT ON
A PROCESS CONTROL
SIMULATION PROGRAM

Submitted To

Dr. D. W. Russell
Real-Time Computing
Pennsylvania State University
Great Valley, Pennsylvania

By

Joseph T. Wunderlich
Engineering Graduate Student

Group Partner: Philip Swayne

July 27, 1990

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

I. INTRODUCTION

This report discusses the design and testing of a process control simulation program to be used as a decision tool to support the design of a bottling plant.

The design of the program is based on the physical system shown in [Figure I-1]. The system mixes ingredients and cooks them according to the following recipe:

One Part Ingredient #1
Two Parts Ingredient #2
Three Parts Ingredient #3
Four Parts Water

Put 100 gallons (total) of ingredients into reactor and then "cook" according to the following heat balance equation:

$$\frac{dTr}{dt} = U_h \, (HTR) \, [T_e - T_r]^2 - 1/10 \, (T_r - T_a)$$

T_r = Reactor temperature (^oC)
U_h = Control signal to turn on heater (0 or 1)
HTR = Heater constant (^oC/min.)
T_e = Heater temperature setting (^oC)
T_a = Ambient temperature

The 100 gallons in the reactor is to be heated up to (140^oC) and then cooled down to (100^oC).

The program is designed to respond to disasters input by the "user" of the program. These disasters are:

(i) A pump stuck off
(ii) The heater stuck off
(iii) The conveyor stuck off
(iv) Running out of empty bottles
(v) Running out of ingredients

The program is designed to produce 12 ounce bottles of product. Every bottle must have at least 12 ounces of product, but, the absolute capacity of each bottle is 14 ounces so that a possible overfill will not spill fluid onto the conveyor. The need for this buffer of space, however, is eliminated by the proper assignment of variables. The final simulation program runs are run with a requested volume of product of 2,000 gallons. This is 2,000 gallons of retail product; therefore, this is (2,000 * 128/12) = (21,334) 12 ounce bottles of product despite the actual amount pumped out of Pump #7. The instantaneous values of time-dependent simulated measured variables will be estimated by the Euler Integration Method discussed in [Section II-2) Differential Equations and Integration]. Other than the heat balance equation, all other differential equations are derived in [Section II-2)]. The analysis of the behavior of the simulation program is based on the system's response to changes of certain "test variables" input by the "user" of this simulation. These responses are evaluated by performance criteria. The coding of this simulation program has been done in "Turbo-Pascal".

The following assumptions are adhered to throughout the simulations:

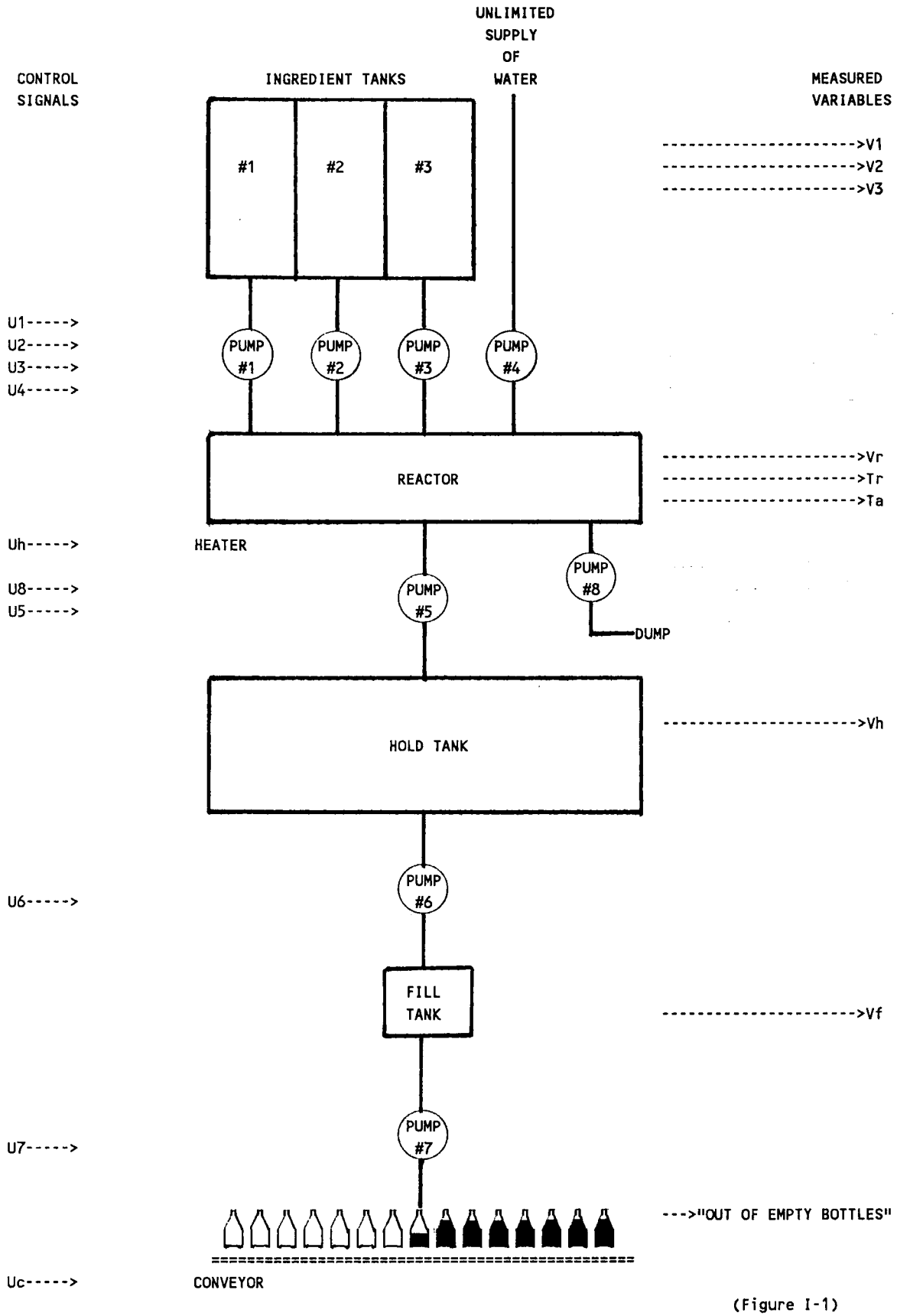
(i) No fluid will ever be left in the reactor.

(ii) The ingredients are mixed by the turbulence of pumping them into the reactor. No mechanical mixing is necessary.

(iii) Turbulence will not effect the volume levels.

(iv) The ingredients enter the reactor at ambient temperature.

(v) The ambient temperature remains constant throughout the production of a requested volume of product.

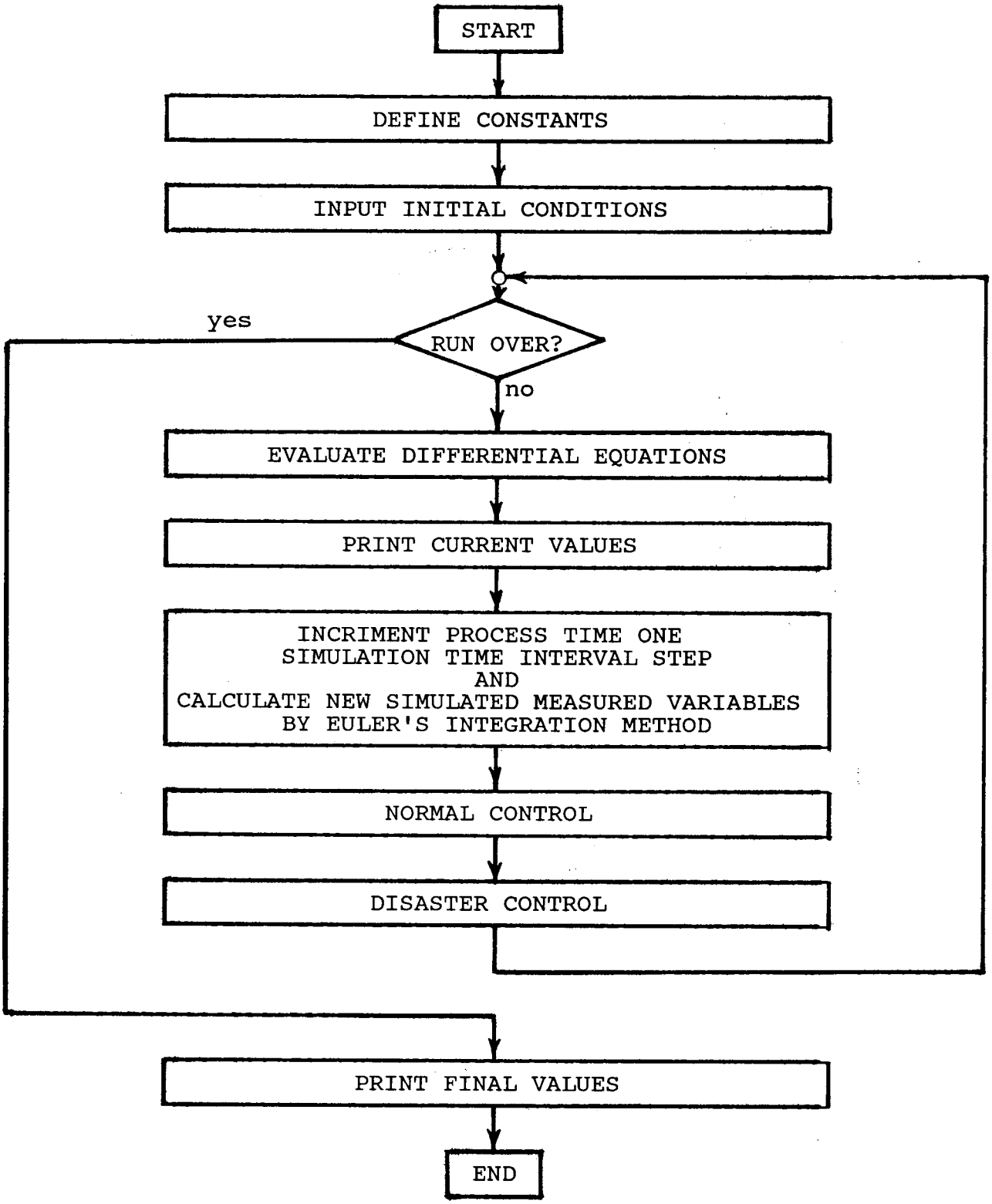

UNLIMITED
SUPPLY
OF
WATER
CONTROL
SIGNALS
INGREDIENT TANKS
MEASURED
VARIABLES
#1
#2
#3
----------->V1
----------->V2
----------->V3
U1----->
U2----->
U3----->
U4----->
PUMP
#1
PUMP
#2
PUMP
#3
PUMP
#4
REACTOR
----------->Vr
----------->Tr
----------->Ta
Uh----->
HEATER
U8----->
U5----->
PUMP
#5
PUMP
#8
DUMP
HOLD TANK
----------->Vh
U6----->
PUMP
#6
FILL
TANK
----------->Vf
U7----->
PUMP
#7
--->"OUT OF EMPTY BOTTLES"
Uc----->
CONVEYOR

(Figure I-1)

II. METHODOLOGY AND THEORY

II. A) PROGRAM DESIGN

The overall flow of this simulation program is shown in [Figure II-1]. The disaster control is located after the normal control so that if a disaster occurs, the normal control signals may be superceded by disaster control signals. This must occur before the program loops back to the differential equation part of the program where these control signals are used to calculate the simulated time-dependent differential behavior of this system. The "run over?" decision block has been located immediately after the initial conditions so that the program will terminate immediately if it is run with a requested volume of product of zero gallons.


START
DEFINE CONSTANTS
INPUT INITIAL CONDITIONS
yes
RUN OVER?
no
EVALUATE DIFFERENTIAL EQUATIONS
PRINT CURRENT VALUES
INCRIMENT PROCESS TIME ONE
SIMULATION TIME INTERVAL STEP
AND
CALCULATE NEW SIMULATED MEASURED VARIABLES
BY EULER'S INTEGRATION METHOD
NORMAL CONTROL
DISASTER CONTROL
PRINT FINAL VALUES
END

(Figure II-1)

II.A.1) VARIABLES AND CONSTANTS

For the purposes of this simulation, the terms constant and variable shall be defined as follows:

CONSTANT - Any parameter, dictated by the problem statement, which can not or most likely will not be changed by a user of this simulation program.

VARIABLE - Any parameter not classified as a constant. This includes simulated measured variables, control variables, and any other symbolic representation which may likely be assigned a numeric value initially or intermittently throughout the simulation.

II.A.1.a) CONSTANTS

One of the assumptions made in the design of this simulation is that no equipment (e.g.: tanks, heaters, etc.) have yet been purchased. Therefore, the only "constants" of this simulation are:

(i) The recipe for the product:
(One part ingredient #1, two parts ingredient #2, three parts ingredient #3 and four parts water)

(ii) The related reactor heat balance target heat (140°C) and target cool (100°C) which the product will be heated, then cooled to respectively.

The reactor heat balance equation itself is also fixed since this is part of the given recipe. [See Section II.A.2) DIFFERENTIAL EQUATIONS AND INTEGRATION.] And this equation is only valid for cooking a volume of 100 gallons. Therefore, the reactor volume capacity could also be considered somewhat of constant of approximately 110 gallons. However, it has been classified as a variable since it can in fact be varied.

II.A.1.b) VARIABLES

Variables have been grouped into seven classifications:

(i) "NON-FLUCTUATING VARIABLES" - These variables do not change value during a simulation run. This includes presets of pumps, tank limits, simulation time interval step (h), etc. [See Table II-2a.] However, these variables are sometimes given different values for successive program runs to determine an optimal value for them. [See Section III) DESIGN OF TEST DATA]

(ii) "SIMULATED MEASURED VARIABLES" - These variables take on instantaneous values as a function of time. These values represent what would be actual measured sensor readings, from the physical process, if a system (process) was actually being controlled (as opposed to the simulation which is being performed here). These simulated measured variable values are approximated by a calculus method described in [Section II.A.2) DIFFERENTIAL EQUATIONS AND INTEGRATION] and are listed in [Table II-2b].

(iii) "SOFTWARE FLAGS" - These variables, when "set", affect the subsequent processing of code. They may be set internally by the program ("Internal Software Flags") or initially by the user ("Disaster Flags"). They play a major role in the "disaster" part of this simulation. [See Section II.A.3) CONTROLS] and [Table II-2d].

(iv) "SIMULATED CONTROL VARIABLES" - These variables change value intermittently throughout the simulation depending on the results of conditional program statements which test other variables. These variables take on values which simulate actual control pulses to activate the reactor heater, pumps, and the conveyor. [See Section II.A.3) CONTROLS] and [Table II-2d].

(v) TIME - There are two types of time [See Table II-1d] considered in this simulation:

(i) Simulated Process Time: This is to be advanced by one simulation time interval step (h), once for each iteration of the main program loop, until the desired quantity of product has been produced or some disaster terminates the simulation.

(ii) Program Run Time: This is the "real" computing time required for the simulation program to execute.

(vi) "PERFORMANCE CRITERIA VARIABLES" - These variables will be used to evaluate the system's performance. They will take on values which indicate productivity (bottling rate) and the percentage waste. [See Sections II.B) CODING and III) DESIGN OF TEST DATA] and [Table II-2d]

(vii) "STATE VARIABLES AND OTHER INTERNAL VARIABLES" - These variables allow the modular structuring of program code. [See Section II.B) CODING] and [Table II-2c]

(TABLE II-2a)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
CONSTANTS:				
TARGET HEAT	T_TrgtHEAT	degrees C	given constant	constant of cook recipe
TARGET COOL	T_TrgtCOOL	degrees C	given constant	constant of cook recipe
NONFLUCTUATING VARIABLES:				
Volumes:				
REACTOR CAPACITY	Vrcap	gal.	set by user	related to 100 gallon cook recipe
HOLD TANK CAPACITY	Vhcap	gal.	set by user	a buffer of approx. 10 reactor volumes
FILL TANK CAPACITY	Vfcap	gal.	set by user	a buffer of approx. 30 bottle volumes
BOTTLE CAPACITY	Vbcap	ounces	set by user	12oz. bottles
DESIRED PRODUCT VOLUME	Vrequest	gal.	set by user	final runs at 2000 gallons
INGEDIENT #1 TANK MIN.	V1min	gal.	=P1*h/60	prevent pumping beyond tank limits
INGEDIENT #2 TANK MIN.	V2min	gal.	=P2*h/60	prevent pumping beyond tank limits
INGEDIENT #3 TANK MIN.	V3min	gal.	=P3*h/60	prevent pumping beyond tank limits
REACTOR MIN.	Vrmin	gal.	=P5*h/60	prevent pumping beyond tank limits
HOLD TANK MIN	Vhmin	gal.	=P6*h/60	prevent pumping beyond tank limits
FILL TANK MIN	Vfmin	gal.	=P7*h/60	prevent pumping beyond tank limits
FILL TANK LOW	Vflow	gal.	=3*Vbcap/128	always keep 3 bottle volumes in fill tank
REACTOR MAX.	Vrmax	gal.	=(Vrcap-10)-2.5*P4*h/60	prevent pumping beyond tank limits
HOLD TANK MAX.	Vhmax	gal.	=Vhcap-1.1*Vrcap	always save room for a reactor batch
FILL TANK MAX.	Vfmax	gal.	=Vfcap-P6*h/60	prevent pumping beyond tank limits
Pump Rates:				
PUMP RATE #1	P1	gal./min.	=0.25*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #2	P2	gal./min.	=0.50*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #3	P3	gal./min.	=0.75*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #4	P4	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #5	P5	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #6	P6	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #7	P7	gal./min.	set by user	fast enough to keep up with P7
PUMP RATE #8	P8	gal./min.	set by user	dumps reactor (for disasters only!)
Heater Variables:				
AMBIENT TEMP.	Ta	degrees C	set by user	likely TEST VARIABLE to be analized
HEATER TEMP. SETTING	Te	degrees C	set by user	likely TEST VARIABLE to be analized
HEATER RATING	HTR	C/min	set by user	likely TEST VARIABLE to be analized
Integration Variable:				
SIMULATION TIME STEP	h	seconds	set by user	likely TEST VARIABLE to be analized

(TABLE II-2b)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
SIMULATED MEASURED VARIABLES:				
Initial Volumes:				
INGEDIENT #1	V1	gal.	set by user	avoid type-3 disaster (out ofingrediant)
INGEDIENT #2	V2	gal.	set by user	avoid type-3 disaster (out ofingrediant)
INGEDIENT #3	V3	gal.	set by user	avoid type-3 disaster (out ofingrediant)
REACTOR	Vr	gal.	set by user(should=0)	system designed so no product left in reactor
HOLD TANK	Vh	gal.	set by user	range= 0 to Vhcap
FILL TANK	Vf	gal.	set by user	range= 0 to Vfcap
Initial Temperature:				
REACTOR	Tr	degrees C	set by user	ingredients propably would enter reactor at T=Ta

(TABLE II-2c)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
FLAGS:				
Internal:				
FLAG #1	Ui	0/1	initially set=0	shutdown ingrediants part of system
FLAG #2	Ur	0/1	initially set=0	shutdown reactor part of system
FLAG #3	Us	0/1	initially set=0	shutdown "lower" part of system
FLAG #4	Um	0/1	initially set=0	shutdown entire system
External:				
DISASTER FLAG #1 (P1)	#1	0/1	set by user	pump #1 stuck off (type-1 disaster)
DISASTER FLAG #2 (P2)	#2	0/1	set by user	pump #2 stuck off (type-1 disaster)
DISASTER FLAG #3 (P3)	#3	0/1	set by user	pump #3 stuck off (type-1 disaster)
DISASTER FLAG #4 (P4)	#4	0/1	set by user	pump #4 stuck off (type-1 disaster)
DISASTER FLAG #5 (P5)	#5	0/1	set by user	pump #5 stuck off (type-1 disaster)
DISASTER FLAG #6 (P6)	#6	0/1	set by user	pump #6 stuck off (type-2 disaster)
DISASTER FLAG #7 (P7)	#7	0/1	set by user	pump #7 stuck off (type-2 disaster)
DISASTER FLAG #8 (HEATER)	#8	0/1	set by user	heater stuck off (type-1 disaster)
DISASTER FLAG #9 (CONVYR)	#9	0/1	set by user	conveyor stuck off (type-2 disaster)
DISASTER FLAG #10(noBTLS)	#10	0/1	set by user	out of bottles (type-2 disaster)

(TABLE II-2d)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
CONTROLS:				
ACTIVATE PUMPS	U1 to U8	0/1	initially set=0	
ACTIVATE CONVEYOR	Uc	0/1	initially set=0	
ACTIVATE HEATER	Uh	0/1	initially set=0	
TIME:				
PROCESS TIME	CUMTIME	seconds	initially set=0	
PROGRAM EXECUTION TIME	REALTIME	seconds	initially set=0	
PERFORMANCE CRITERIA:				
BOTTLING RATE	BOTLNGRATE	#/min	=#BOTTLES/CUMTIME	[see section II.B) CODING]
PERCENTAGE WASTE	%OVERFILL	percent	---------->	[see section II.B) CODING]
STATE VARIABLES:				
"FILL"	"FILL"	none	initially set	[see section II.B) CODING]
"HEAT"	"HEAT"	none	initially not set	[see section II.B) CODING]
"COOL"	"COOL"	none	initially not set	[see section II.B) CODING]
"DRAIN"	"DRAIN"	none	initially not set	[see section II.B) CODING]
OTHER INTERNAL VARIABLES:				
PRINT INDEX	PRINTINDEX	none	set by user	defines frequency of data writes
Misc. Internal Variables	-	none	initially set=0	[see section II.B) CODING]

II.A.2) DIFFERENTIAL EQUATIONS AND INTEGRATION

The purpose of the differential equations and integration parts of this simulation is to simulate actual measured process variable values read from sensors. Since all the simulated measured variables vary in time, they can be represented by a differential equation. The rate of change of any tank volume is equal to (the flow rate in) less (the flow rate out). Thus;

$$dv = (P_{in} - P_{out})$$

Where P is a pump rate. Since the pumps must be turned on to have an effect on this equation, the control signals (U's) to turn on the pumps must be included in this equation as follows:

$$\frac{dv}{dt} = [[(U_{in} * P_{in}) - (U_{out} * P_{out})]] * K$$

K = conversion factor (1 min/60 sec)

The differential equation describing each tank volume change rate is shown in [Figure II-2].

In addition to the volume change rates, the reactor temperature changes with respect to time according to a heat balance equation given as part of the product recipe:

$$\frac{dTr}{dt} = [[Uh * HTR * (Te - Tr)^2] - [1/10 * (Tr - Ta)]] * K$$

Uh = Control signal to turn on heater
HTR = Heater rating (°C/min)
Te = Heater Temperature Setting
Tr = Reactor Temperature
K = conversion factor (1 min/60 sec)

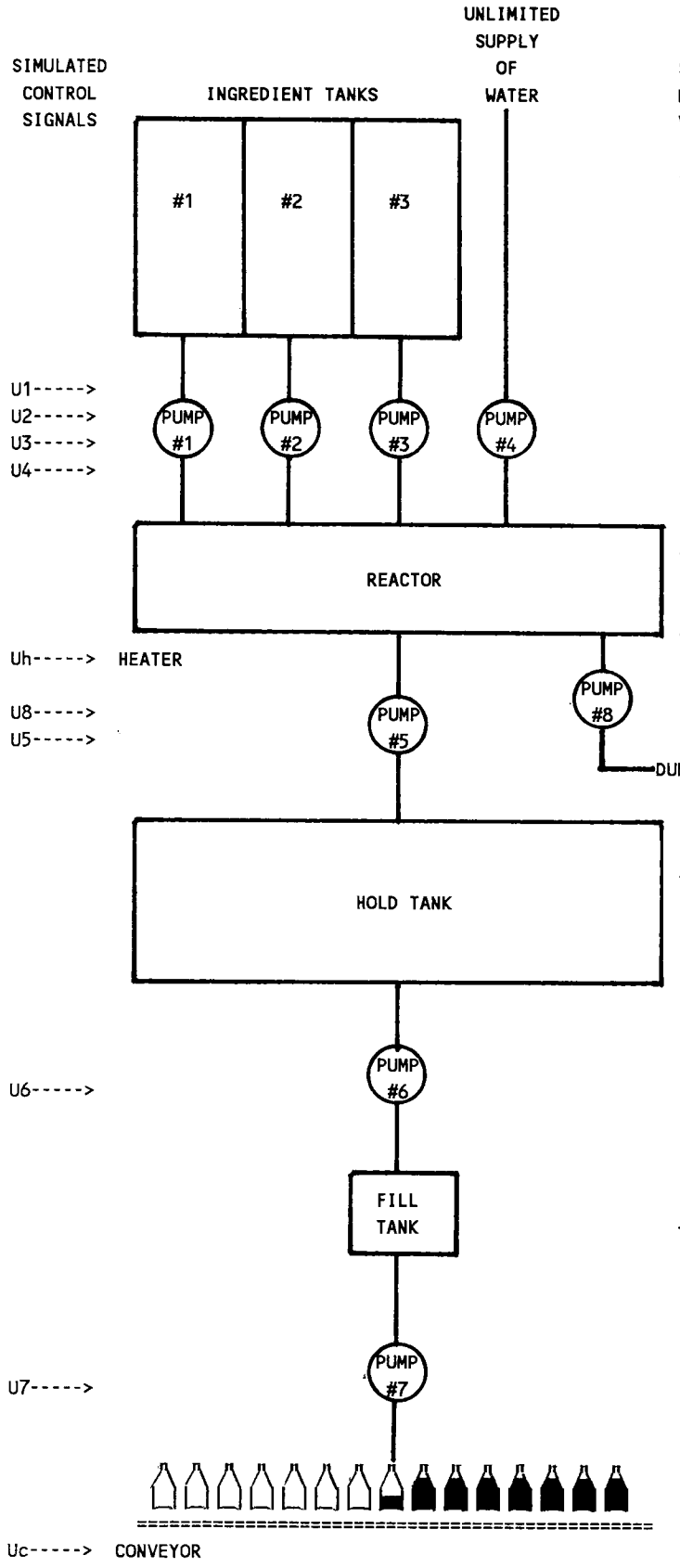
Once the time-dependent differential equations are defined, the

instantaneous slope of the function that describes the time-dependent variable is known (i.e. $\underline{dVariable}$) and the value of the
dt
time-dependent variable can be estimated at (h) seconds later by using the Euler Method of Integration:

$$V_{t+h} = [V_t + h * (\underline{dv})]_{dt}$$

$$TR_{t+h} = [TR_t + h * (\underline{dTr})]_{dt}$$

The initial tank volumes and reactor temperature are defined by the user. The equation for each simulated measured variable is shown in [Figure II-2].



SIMULATED MEASURED VARIABLES	RATE OF CHANGE	APPROXIMATE INSTANTANEOUS VALUE
----->V1	dV1/dt= -P1(U1)	V1new= V1old+ h(dV1/dt)
----->V2	dV2/dt= -P2(U2)	V2new= V2old+ h(dV2/dt)
----->V3	dV3/dt= -P3(U3)	V3new= V3old+ h(dV3/dt)
----->Vr	dVr/dt= -P5(U5)-P8(U8)+P1(U1) +P2(U2)+P3(U3)+P4(U4)	Vrnew= Vrold+ h(dVr/dt)
----->Tr	dTr/dt= Uh(HTR)(Te-Tr)**2 -(1/10)(Tr-Ta) T_TrgtHEAT= 140 degrees C T_TrgtCOOL= 100 degrees C	Trnew= Trold+ h(dTr/dt)
----->Vh	dVh/dt= -P6(U6)+P5(U5)	Vhnew= Vhold+ h(dVh/dt)
----->Vf	dVf/dt= -P7(U7)+P6(U6)	Vfnew= Vfold+ h(dVf/dt)

(Figure II-2)

II.A.3) CONTROLS

The simulated control signals (U1 to U8, Uh, and Uc) are given values (0 or 1) depending on conditional program statements within the control sections of the simulation program. These two sections are:

(a) Normal Control

(b) Disaster Control

II.A.3.a) NORMAL CONTROL

The normal control section contains two main subsections; "Reactor Control" to control everything "above" Pump #5 (including P5); and a lower control subsection to control all "below" Pump #5.

The reactor control program flow is shown in [Figures II-3]. This part of the program is organized into four states; "fill", "heat", "cool" and "drain". Using a program "case statement" described in [Section II.B) CODING], the program structure has been made such that the "fill" state is always entered at the beginning of any program run. The "fill" state has the capability of stopping ingredients from being pumped into the reactor, if, in the previous cycle of the program, Flag #1 (see below) has been set by either the disaster controls or the previous "fill" state. The previous "fill" state will set Flag #1 if the hold tank has enough volume to finish producing the requested volume of product ($V_{togo} < V_h$ + projected loss) or if there aren't enough ingredients to completely fill the reactor

(($V1_{ref} < 1/10V_{rmax}$) or ($V2_{ref} < 2/10V_{rmax}$) or ($V3_{ref} < 3/10V_{rmax}$)). The variables V_{togo}, $V1_{ref}$, $V2_{ref}$ and $V3_{ref}$, defined in [Section II.B) CODING], are defined such that volume will never be left in the reactor when the system must be shut down under normal control. The flow of the lower control subsection is shown in [Figure II-4].

Both the reactor control and lower control parts of the normal controls react to previously set internal software flags generally defined as follows:

Flag #1 = Stop ingredients from being pumped
Flag #2 = "Shut-down" system Pump #5 and "above"
Flag #3 = "Shut-down" system "below" Pump #5
Flag #4 = "Shut-down" entire system

II.A.3.b) DISASTER CONTROL

The disaster control section [Figure II-5] creates control signals and sets software flags (Flag #1, #2, #3, #4) only if the user of the simulation program sets the following software disaster flags:

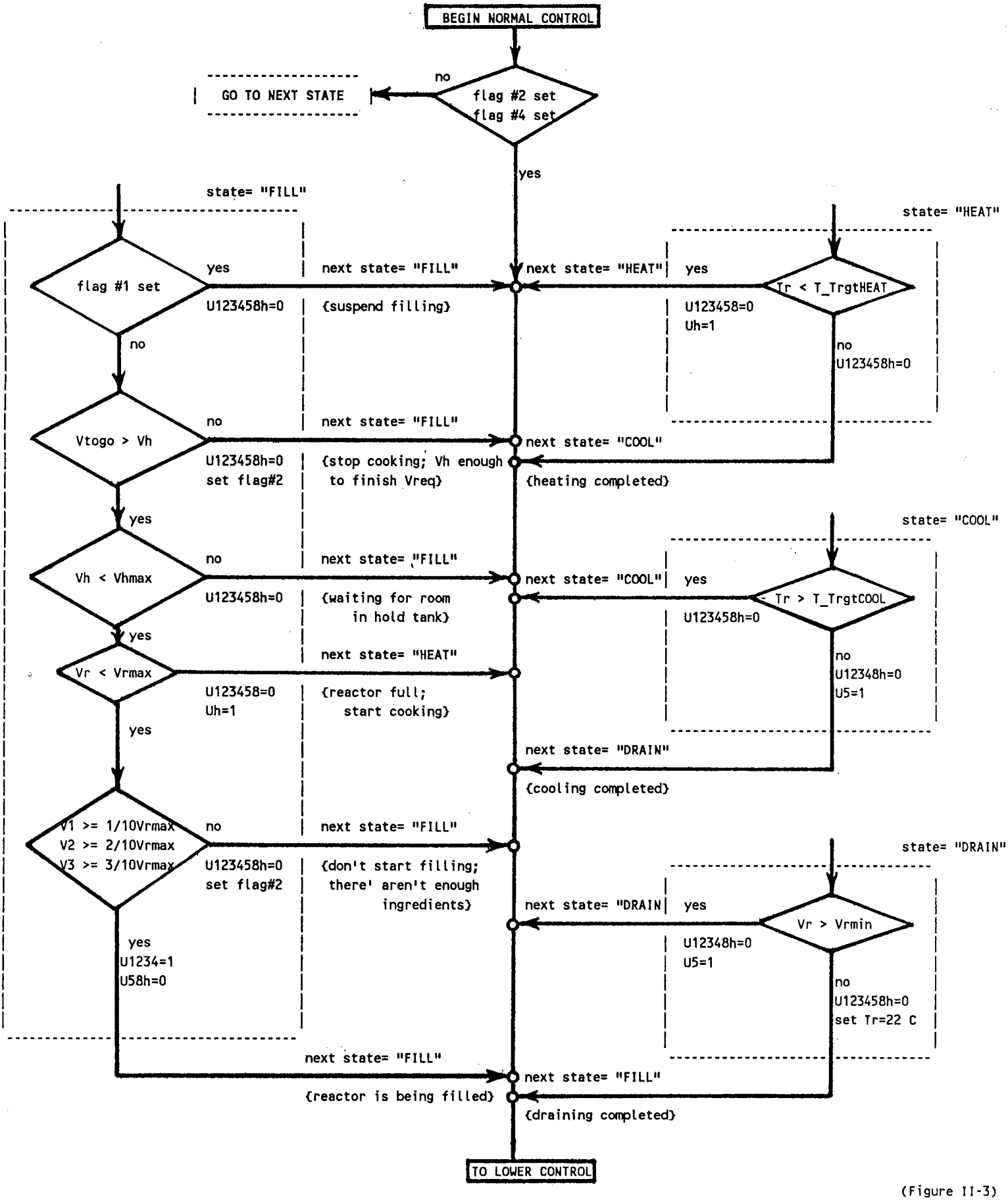
(i) Disaster Flag #1 to #7 (one of Pumps #1 to #7 stuck off)
(ii) Disaster Flag #8 (heater stuck off)
(iii) Disaster Flag #9 (conveyor stuck off)
(iv) Disaster Flag #10 (out of bottles)

These disaster flags simulate what would be actual sensor inputs from a mechanism indicating that it is broken (i.e.: stuck off). The disaster controls have been designed so that the system maintains operation as long as possible once a disaster is detected. For example, if a disaster occurs somewhere in the reactor part of the system, the system will keep filling bottles by using the volumes in the fill tank and hold tank. A disaster

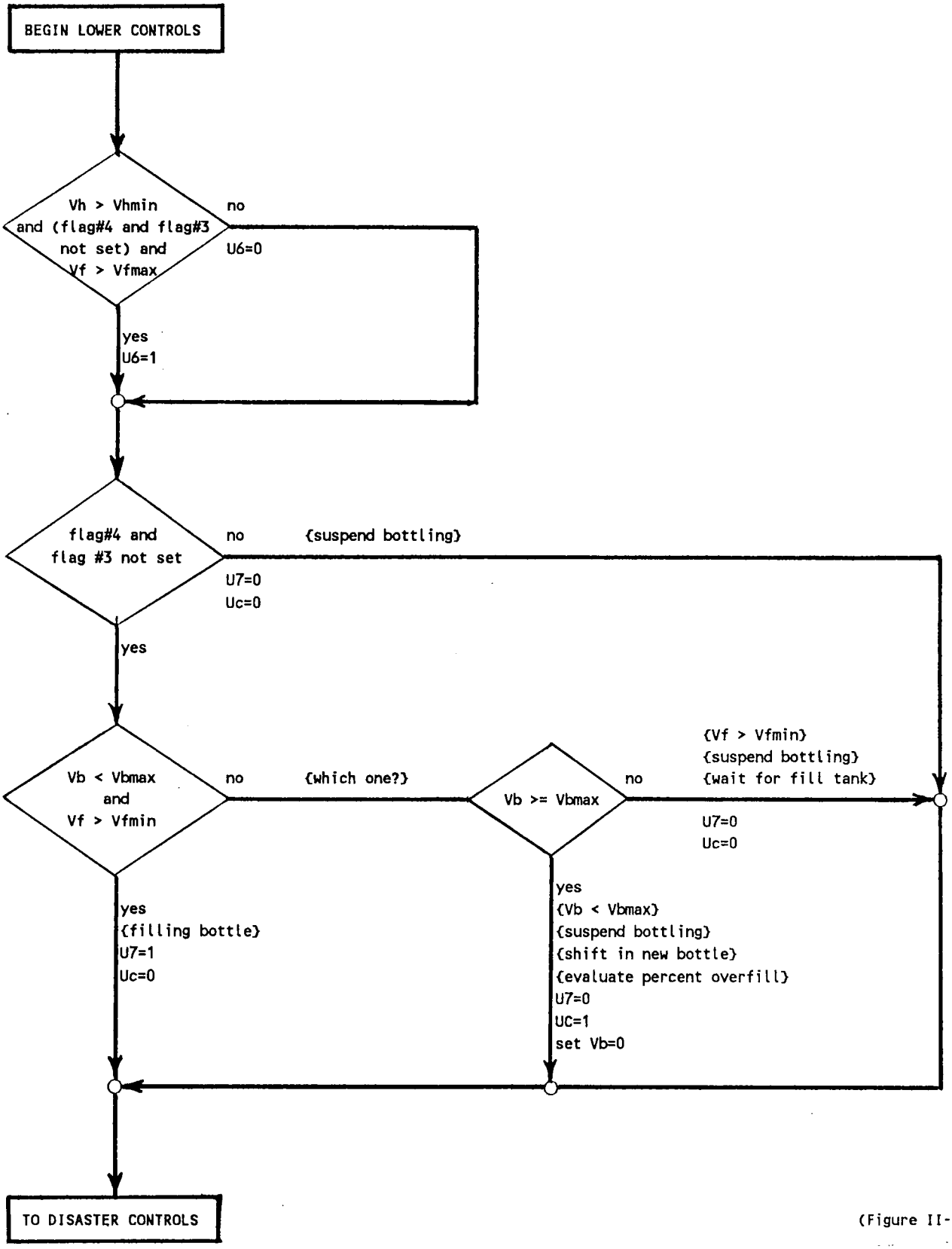
is considered a Type-1 disaster when it causes reactor operations to stop while lower controls keep on bottling. Failure of P1, P2, P3, P4, P5 or the HTR are all of this type. A Type-2 disaster causes the lower bottling cycle to stop (P6 and below) while not affecting the normal operation of the reactor. In this case of disaster the reactor is brought to a stop gently and shut off, to allow repairs with minimal volume in the system. Failure of P6, P7, the conveyor, or "out of bottles" are Type-2 disasters.

Running out of ingredients is a Type-3 disaster. Normal controls will handle this type of problem and prevent the reactor tank from being filled if there is an insufficient volume of ingredients on hand.

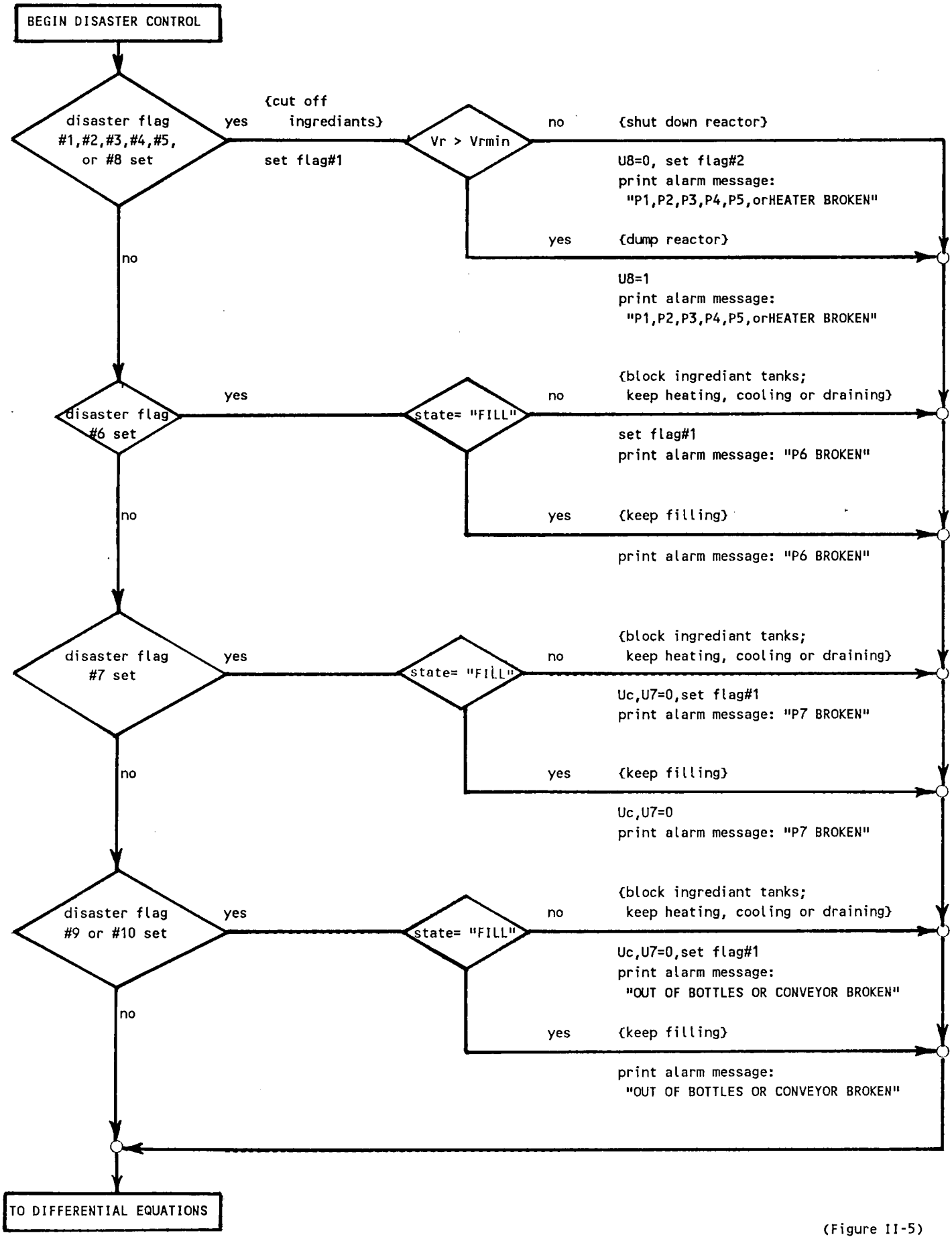
Disaster simulations in this program are limited to "stuck-off" equipment (pumps, heater, conveyor), low ingredients conditions, and running out of bottles. For each situation, appropriate alarms are sounded.


BEGIN NORMAL CONTROL
flag #2 set
flag #4 set
no
GO TO NEXT STATE
yes
state= "FILL"
flag #1 set
yes
next state= "FILL"
U123458h=0
{suspend filling}
no
Vtogo > Vh
no
next state= "FILL"
U123458h=0
set flag#2
{stop cooking; Vh enough to finish Vreq}
yes
Vh < Vhmax
no
next state= "FILL"
U123458h=0
{waiting for room in hold tank}
yes
Vr < Vrmax
next state= "HEAT"
U123458=0
Uh=1
{reactor full; start cooking}
yes
V1 >= 1/10Vrmax
V2 >= 2/10Vrmax
V3 >= 3/10Vrmax
no
next state= "FILL"
U123458h=0
set flag#2
{don't start filling; there' aren't enough ingredients}
yes
U1234=1
U58h=0
next state= "FILL"
{reactor is being filled}
next state= "HEAT"
state= "HEAT"
yes
Tr < T_TrgtHEAT
U123458=0
Uh=1
no
U123458h=0
next state= "COOL"
{heating completed}
state= "COOL"
yes
Tr > T_TrgtCOOL
U123458h=0
no
U12348h=0
U5=1
next state= "DRAIN"
{cooling completed}
state= "DRAIN"
yes
Vr > Vrmin
U12348h=0
U5=1
no
U123458h=0
set Tr=22 C
next state= "FILL"
{draining completed}
TO LOWER CONTROL

(Figure II-3)


BEGIN LOWER CONTROLS
Vh > Vhmin
and (flag#4 and flag#3
not set) and
Vf > Vfmax
no
U6=0
yes
U6=1
flag#4 and
flag #3 not set
no
{suspend bottling}
U7=0
Uc=0
yes
Vb < Vbmax
and
Vf > Vfmin
no
{which one?}
Vb >= Vbmax
no
{Vf > Vfmin}
{suspend bottling}
{wait for fill tank}
U7=0
Uc=0
yes
{filling bottle}
U7=1
Uc=0
yes
{Vb < Vbmax}
{suspend bottling}
{shift in new bottle}
{evaluate percent overfill}
U7=0
UC=1
set Vb=0
TO DISASTER CONTROLS

(Figure II-4)


BEGIN DISASTER CONTROL
disaster flag #1,#2,#3,#4,#5, or #8 set
yes
{cut off ingrediants}
set flag#1
Vr > Vrmin
no
{shut down reactor}
U8=0, set flag#2
print alarm message: "P1,P2,P3,P4,P5,orHEATER BROKEN"
yes
{dump reactor}
no
U8=1
print alarm message: "P1,P2,P3,P4,P5,orHEATER BROKEN"
disaster flag #6 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
set flag#1
print alarm message: "P6 BROKEN"
yes
{keep filling}
no
print alarm message: "P6 BROKEN"
disaster flag #7 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
Uc,U7=0,set flag#1
print alarm message: "P7 BROKEN"
yes
{keep filling}
no
Uc,U7=0
print alarm message: "P7 BROKEN"
disaster flag #9 or #10 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
Uc,U7=0,set flag#1
print alarm message: "OUT OF BOTTLES OR CONVEYOR BROKEN"
yes
{keep filling}
no
print alarm message: "OUT OF BOTTLES OR CONVEYOR BROKEN"
TO DIFFERENTIAL EQUATIONS

(Figure II-5)

II-B) CODING

The code for this project was produced and compiled using Turbo Pascal 5.5 on an INTEL 286 based micro computer. The physical layout of the code follows to a large degree, the logical layout of the operation/control cycle. First, types, constants, and variables are declared. Next, subroutines and functions are defined. Then, comes the main program, in which initial conditions are first set and real-time clock is checked, then the system loops until it is directed to stop by completion of the batch or by some disaster occurring. Finally, on exiting the main loop, the last system configuration is printed out, the real-time clock is checked as the program terminates.

II.B.1) VARIABLE TYPES, CONSTANTS AND VARIABLES

Four user-defined types are used. "Volumes" is used as an array parameter so that all the volume variables can be referred to via an array. This makes it clear which variable is being used at a given time and also makes it easier to use a loop for the integration subroutine. Possible values of this type are V1, V2, V3, V4, V_h, V_r, V_b. "Vol-Values" is an array of the above type, therefore, particular variables, such as "V" or "dVdt4" can be declared as this type of array. Therefore, "V" can only have array parameters of those given for "Volumes" (ex. V[V1] or V[V_h]). "I Vol-Values" is the same as Vol-Values, except that it is an array of integers not reals. It is used only for output format purposes. The type "state" provides four discrete values for the cooking cycle status, fill, heat, cool, drain. The variable "state", then, can only have a value of one of these, and it is clear in which state the cooker is at a given time.

Constants, as defined in [Section II.A.1.a) Constants], are fixed by the specified recipe.

Variables that are strictly internal and not seen by the user of this simulation program are explained in [Section II.A.1.b) Variable]. The user does not normally have access to these.

Other variables, whether user adjustable or just observable to the user, are explained in [Section II.A.1.b) Variables].

II.B.2) SUBROUTINES AND FUNCTIONS

Two subroutines and one function are used. The "Disaster Procedure" (subroutine) received values from the main program and returns the same values or an erroneous one if a disaster has been directed to occur. This procedure also dictates which disaster, if any, is to occur. For example, if Disaster Flag is set to 1, then Pump #1 has failed, and the system must respond. If a Disaster Flag is set to 8 then the heater has failed. See [Table II-1b] for a list of possible disasters. Note, that this Disaster Flag does not cause a failure; it only indicates a failure. So, to simulate a failure of P1, Disaster Flag #1 must be set to 1. There is considerable room for expansion of the Disaster Procedure both in detecting of failures and in directing of failures (causing).

The procedure "Get Time Diff" reads in two sets of real-time (the earlier time then the later time) and returns in the earlier's slot the difference between the two times in like format. It does not account for times which cross over the PM to AM change.

The function "Integration Function" contains the Euler's integration equation. It receives X, dXdt and h, and returns as its value X at the next time. To change the integration method of this project, one need only change this function.

II.B.3) MAIN PROGRAM

The structure of the main program is as follows. The start real-time is recorded, the data output file is opened, user-changeable variables are set to initial conditions, other variables are set to initial conditions, and then the main loop is entered from then on, the program cycles within the main loop until something causes the master cut off to be set, such as completion of the requested volume of product or some disaster.

Users of this simulation program may change the values of many of the variables controlling the system. For example, (h) can be adjusted to obtain better results in relation to (P7) and other variables. (Print Index) can be changed to have the program print out more or fewer of the cycles of the system. Pump rates can be changed to make certain design specs. For a complete list of user changeable variables, see the variable description tables [Table II-1a, b, c, d].

Other variables in the system vary, but are not directly adjustable by the user. For example, pump rates P1, P2 and P3 are directly proportional to P4, and the cooking recipe. (P1 = 1/4 P4), (P2 = 1/2 P4) and (P3 = 3/4 P4). The program also sets tank upper and lower volume limits internally, based on each tank's capacity and the rate at which the attached pumps will change its volume. For example, the reactor tank, with a capacity of (V_{rcap} = 110). (V_{rmax}) is determined by taking

the maximum desired volume in it, V_{rcap} - 10 (for a cook volume of 100 gallons), and subtracting a volume equal to the largest amount of liquid that can enter the tank before the next control sequence can adjust for it. That is,in one cycle (h) seconds long, pump P1 will pump into the reactor [P1 * h * (1/60)] gallons. Pumps P2, P3 and P4 will follow the same way. Since they are all actually derived from P4, we can see that the maximum volume entering the tank in any cycle is [2.5 * P4 * h * (1/60)]. Therefore, if V_{rmax}, the upper limit which will trigger shut off at all input pumps, is set to (V_{rcap}-10) - 2.5 * P4 * h * (1/60) then the reactor will not overfill but may stop filling at just under the (V_{rcap}-10) desired level. All other tank level limits are derived in a similar fashion. Note that as (h) or the pump rates are decreased, the $V_{max\ or\ min}$ shut off limit gets closer to the precise target value. V_{flow}, the turn-on limit for P6 is set to maintain a (V_f) of at least three bottles of volume.

The main loop has six general sections as shown in [Figures II-3, 4, 5], which are executed once each cycle. The cycle is simulated to happen every (h) seconds. First, the differential equations are evaluated. Next, all results are printed out to screen; and files also if the PrintCount is up to the Print Index, or if a change of state has occurred. Integration of all volumes and the temperature equation is accomplished next using a subroutine call to the Euler's equation subroutine. Reactor

controls are next, followed by lower controls. The reactor controls section included control of P1-P5, P8 and the heater using U_1 - U_5, U_8 and U_h. The lower control section includes control of U_6, U_7 and U_c. The disaster section is last, and includes evaluation of some possible disaster conditions as well as a call to the disaster subroutine which identifies which, if any, disaster has occurred.

There are four master control switches (U_i, U_r, U_s and U_m) and three master control flags (U_{k1}, U_{k2} and U_{k3}) [see Table II-1d]. In a physical plant, the four master control switches would be installed as redundant switches since they are located in all equations in parallel with the normal controls. So, if a given master control cuts off pumps (P1-P4) and the (U1-U4) do not shut off due to a fault, the master controls would still prevent (P1-P4) from operating. (U_i) is used to shut off (P1-P4) (parallel to U1-U4). (U_r) controls (U1-U4) in the same way but also controls U_5, U_8 and U_h. Therefore, if (U_r = 1), the entire reactor section is disabled. U_5, similarly, controls the entire lower control section. U_m controls everything. So for general reactor shutdown, U_m, is set allowing the reactor, if it is in process, to finish while disabling any father injection of ingredients. The U_k flags are used to avoid the repetitive use of many U's. The $U_{k's}$ also negate their signal so (U_{k1} = 0) is set on.

The main loop is exited by setting U_m, either from a disaster control or from completion of the batch. Once enough product has

been cooked, U_r is set, while the bottling process continues until the desired volume is made. The program anticipates the amount of product needed in the various tanks to complete the batch including extra needed for losses.

On exiting the main loop, the last condition of the system is printed out, the real-time clock is checked, output file closed and the program terminated.

III-C) HARDWARE/SOFTWARE

The code for this project was produced with Turbo Pascal 5.5. Tests were run by changing appropriate values within the Pascal source code and recompiling the code.

Test results were sent to output files and/or recorded by hand, depending on the test. Results were then imported, or typed in, to Quattro Pro for analysis and production of charts.

Variable charts and spreadsheets were produced using Lotus. The body of this report was produced using Word Perfect 5.0.

The coding and tests were all run on a Leading Edge D2 with an 80286 microprocessor running at 10 MHz. Files were stored on a 40 MB 28 msec hard drive.

For this system, (h) and (P7) are closely tied together in determining how much apparent waste is produced by the bottler. In general, though, as (h) is made smaller the waste is reduced.

(h) also plays a large part in the Euler's method of integration used. Again, it is beneficial to reduce (h) to minimize errors in the integration. There is one major limit, however, on how small (h) can go, and that is the actual real-time needed to make all the computations using (h). As (h) gets smaller, the real-time needed to run a simulation increases, as shown on [Figures II-6 and 7]. This real-time is very much hardware dependent. A faster computer would permit smaller values of (h) for the same actual run time.

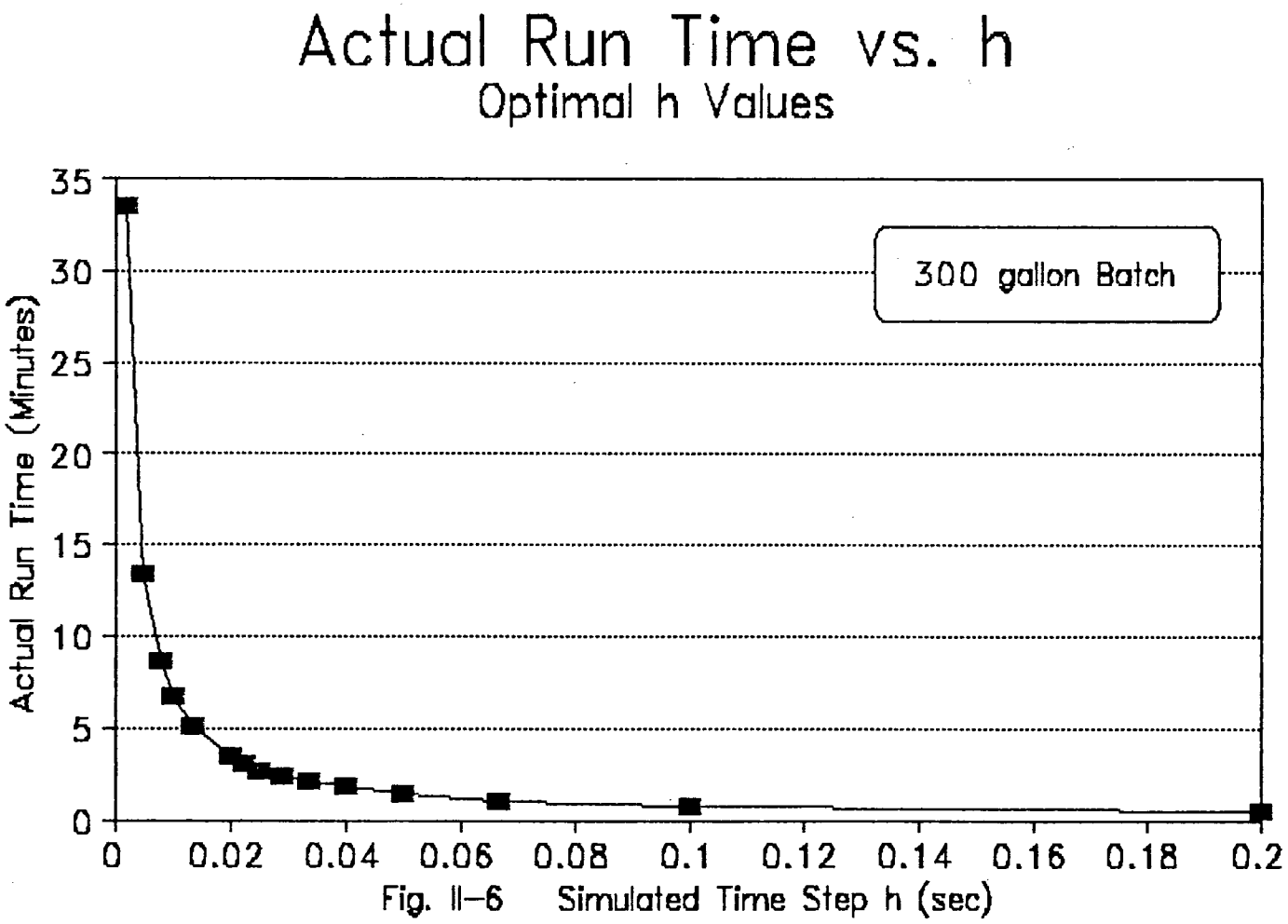

Actual Run Time vs. h
Optimal h Values
300 gallon Batch
Actual Run Time (Minutes)
35
30
25
20
15
10
5
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Simulated Time Step h (sec)

Fig. II–6

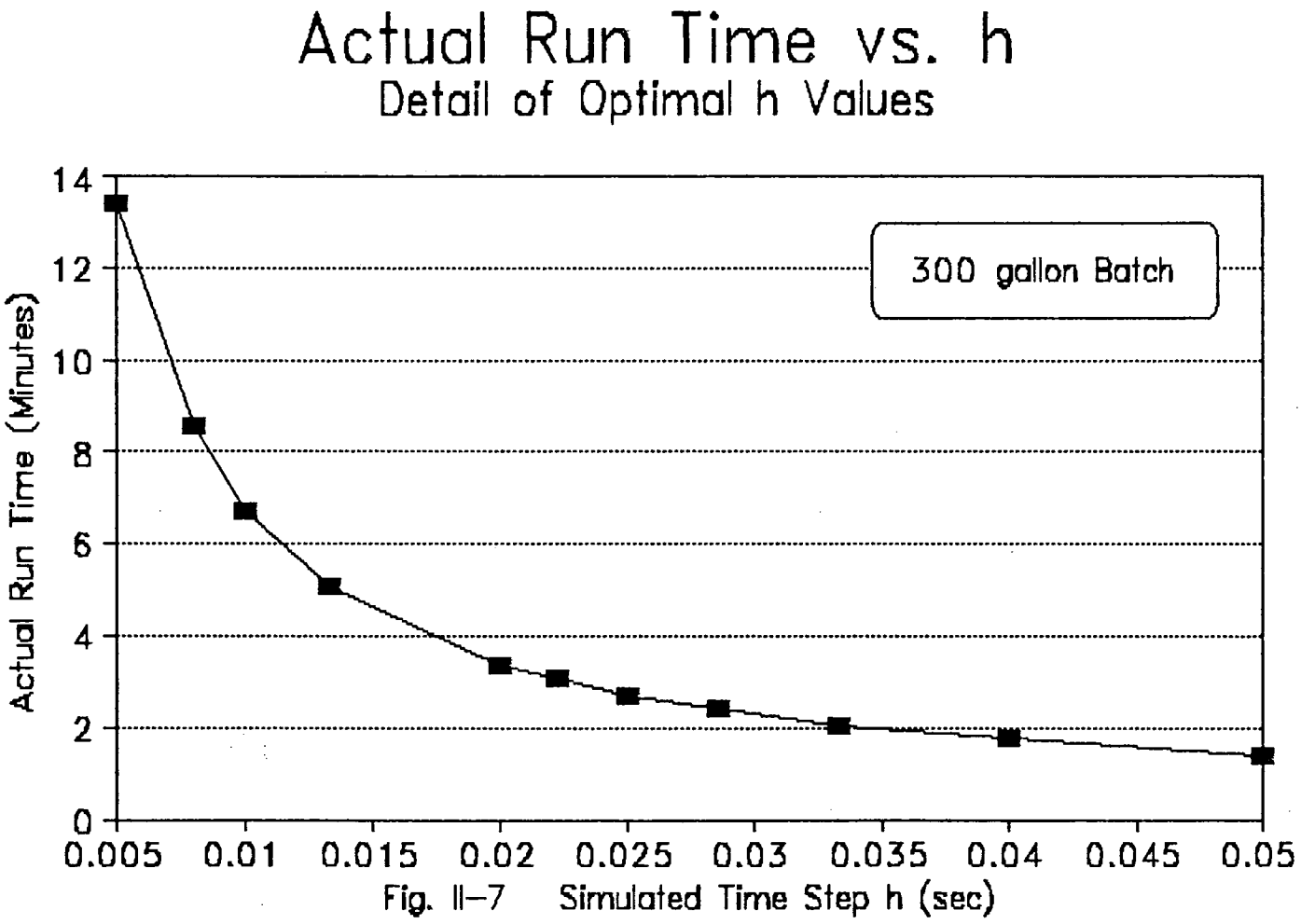

Actual Run Time vs. h
Detail of Optimal h Values
300 gallon Batch
Actual Run Time (Minutes)
14
12
10
8
6
4
2
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
Simulated Time Step h (sec)

Fig. II–7

Internal Variables

Variable	Derivation	Use
CoolingConstant	=0.1	used in the reactor heat equation, has 4
State	internal	possible values: Fill, Heat, Cool, and
		Drain, used to identify stages of cooking
Previous State	by previous state	for detecting when printout is needed, if a change of state has occurred
PrintIndex	input	says how often to print some results
PrintCount	reset after each print	counts number of cycles since last print to the screen and output files
In_file	user input	internal name for the user input file
Out_file	result file	internal name for the results output file
StateInd	internal	internal status messages
MsgInd	internal	internal status messages
Msg1-3	by process	contain msg about process for screen view
ch	dummy	for input at end to delay screen clear
Vrequested	user input	gallons to be made
NumberOfBottles	by process	starts at 0, is incremented each time a bottle is filled
Vdone	#bottles* 12/128	total volume done this batch
Vtogo	Vrequested-Vdone	tells how much more to go
Vfinished_prod	Vdone+V in all tanks	total product made thus far
Vinitial	Vr+Vh+Vf	total product in tanks at start of batch
Vingredients	V1-3 + Vinitial	total product into system
V_[ ]	internal	integer value of tank volumes for output
Vblost	Vb-Vbcap	cumalative loss from over filling bottles
PercentOverfill	Vblost/Vbcap /#bottles*%	average percent overfill of bottles
Vmaxbatchlost	Vtogo*P7*h* (0.1778)	maximum possible wasted product for current batch, used to estimate Vtogo
CumTime	count h's	simulation rum time, cycles*h
BottlingRate	# bottles/ CumTime	running ratio of bottles done to run time
Hour1, etc	internal	used to get real time
DisasterFlag	internal	used to identify a particular disaster
Ui	internal	overrides set of U1-U4
Ur	internal	overrides set of U1-U5, U8, Uh
Us	internal	overrides set of U6, U7, Uc
Um	internal	overrides set of all controls
Uk1	Um, Ur or Ui	master control flags, are inversed OR's
Uk2	Um or Ui	of master controls
Uk3	Um or Us	

III DESIGN OF TEST DATA

The design of test data involves the choosing of certain variables to be changed (test variables) which could significantly affect the overall system performance and general system behavior. The first step in this process of choosing is to identify all variables which can be changed by the "user" of this simulation program (i.e.: any variable which is not purely a function of other variables such as internal software flags, case variables, tank volume minimums and maximums, etc.). The variables which will be allowed to be changed by the "user" of this simulation program shall be as follows:

(a) Tank capacities and bottle capacity (V_{rCAP}, V_{hCAP}, V_{FCAP}, V_{BCAP})
(b) Desired volume of product requested (V_{req})
(c) Pump rates (P4, P5, P6, P7, P8) (Note: P1, P2, P3 are a function of P4)
(d) Ambient temperature (Ta)
(e) Heater temperature setting (Te)
(f) Heater rating (HTR)
(g) Simulation time interval step (h)
(h) Initial tank volumes and reactor temperature (V_1, V_2, V_3, V_r, V_h, V_f, Tr)
(i) Disaster flags

Of these variables, "Test Variables" have been selected to answer two questions:

(1) Which variables will probably have a significant effect on the performance criteria (eg: bottling rate and percentage waste)?

(2) How will the system volume levels react to a disaster?

III.A) PERFORMANCE CRITERIA

The performance criteria that have been chosen to evaluate the performance of this system are as follows:

(a) The bottling rate (bottles/minute) produced [See Section II.B CODING].

(b) The percentage waste due to overfilling bottles [See Section II.B CODING].

III.B) SELECTED TEST VARIABLES

The initial tank volumes were not chosen as test variables because: 1) Initial fill tank or ingredient tank volumes would only effect the performance criteria if initially empty (i.e. an empty ingredient tank would simply cause a termination of the simulation; an empty fill tank would merely cause a short delay in turning on P7). 2) The system is designed so that no volume is ever left in the reactor; therefor (V_r) initially must be zero. 3) The initial hold tank volume does have an effect on performance criteria, however, an analysis of this was deferred until after the primary and secondary analysis (run at V_{req} = 300 gal.) and the final analysis (run at V_{req} = 2,000 gal.) were completed. [See Section V) DISCUSSION OF RESULTS]

Tank capacities should have values based on the rationale previously described in [Table II-2a] and therefore have not been chosen as test variables.

The bottle size (12 ozs.) and the desired volume of product (2,000 gallons) shall remain fixed for the final program runs as per the given specifications. However, smaller desired volumes of product will be used for initial test runs.

Pump Rate #8 (which dumps the reactor) will only be activated if certain disasters occur; and since the entire system will always be shutting down in these instances P8 will have no affect on the performance criteria under normal operating conditions. Therefore, P8 has not been chosen as a test variable. However, Pump Rates (P4, P5, P6 and P7) may have a significant affect on the performance criteria and have therefore been chosen as test variables.

The ambient temperature (Ta), heater temperature setting (Te), heater rating (HTR), and simulation time interval step (h) are all believed to have a significant affect on the performance criteria and therefore have also been chosen as test variables.

IV RESULTS SUMMARY

The following results describe the system's performance and behavior when subjected to varying values of test variables (P4, P5, P6, P7, Ta, Te, HTR and h) and disaster flags. The values of (h) and (P7) used in [Graph #3, #4, #5, #6, #7 and #8] have been obtained from a primary analysis of [Graph #1, #2 and #3]. The values of all of the test variables used in [Graph #9, 10, 11 and 12] have been obtained from a secondary analysis of [Graph #1, 2, 3, 4, 5, 5, 6, 7 and 8]. These two phases of analysis and a final analysis are discussed in [Section V DISCUSSION OF RESULTS]. The tables on the following two pages list all of the initial values for each graph. In the first four tables [Tables IV-1a,b,c,d] "user changeable" variables are identified by the "set by user" entry in the "value assigned" column [See Section III DESIGN OF TEST DATA]. The next four tables [Tables IV-2a,b,c,d] contain all of the initial values corresponding to each graph.

(TABLE IV-1a)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
CONSTANTS:				
TARGET HEAT	T_TrgtHEAT	degrees C	given constant	constant of cook recipe
TARGET COOL	T_TrgtCOOL	degrees C	given constant	constant of cook recipe
NONFLUCTUATING VARIABLES:				
Volumes:				
REACTOR CAPACITY	Vrcap	gal.	set by user	related to 100 gallon cook recipe
HOLD TANK CAPACITY	Vhcap	gal.	set by user	a buffer of approx. 10 reactor volumes
FILL TANK CAPACITY	Vfcap	gal.	set by user	a buffer of approx. 30 bottle volumes
BOTTLE CAPACITY	Vbcap	ounces	set by user	12oz. bottles
DESIRED PRODUCT VOLUME	Vrequest	gal.	set by user	final runs at 2000 gallons
INGEDIENT #1 TANK MIN.	V1min	gal.	=P1*h/60	prevent pumping beyond tank limits
INGEDIENT #2 TANK MIN.	V2min	gal.	=P2*h/60	prevent pumping beyond tank limits
INGEDIENT #3 TANK MIN.	V3min	gal.	=P3*h/60	prevent pumping beyond tank limits
REACTOR MIN.	Vrmin	gal.	=P5*h/60	prevent pumping beyond tank limits
HOLD TANK MIN	Vhmin	gal.	=P6*h/60	prevent pumping beyond tank limits
FILL TANK MIN	Vfmin	gal.	=P7*h/60	prevent pumping beyond tank limits
FILL TANK LOW	Vflow	gal.	=3*Vbcap/128	always keep 3 bottle volumes in fill tank
REACTOR MAX.	Vrmax	gal.	=(Vrcap-10)-2.5*P4*h/60	prevent pumping beyond tank limits
HOLD TANK MAX.	Vhmax	gal.	=Vhcap-1.1*Vrcap	always save room for a reactor batch
FILL TANK MAX.	Vfmax	gal.	=Vfcap-P6*h/60	prevent pumping beyond tank limits
Pump Rates:				
PUMP RATE #1	P1	gal./min.	=0.25*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #2	P2	gal./min.	=0.50*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #3	P3	gal./min.	=0.75*P4	recipe ingrediant proportions (1/2/3/4)
PUMP RATE #4	P4	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #5	P5	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #6	P6	gal./min.	set by user	likely TEST VARIABLE to be analized
PUMP RATE #7	P7	gal./min.	set by user	fast enough to keep up with P7
PUMP RATE #8	P8	gal./min.	set by user	dumps reactor (for disasters only!)
Heater Variables:				
AMBIENT TEMP.	Ta	degrees C	set by user	likely TEST VARIABLE to be analized
HEATER TEMP. SETTING	Te	degrees C	set by user	likely TEST VARIABLE to be analized
HEATER RATING	HTR	C/min	set by user	likely TEST VARIABLE to be analized
Integration Variable:				
SIMULATION TIME STEP	h	seconds	set by user	likely TEST VARIABLE to be analized

(TABLE IV-1b)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
SIMULATED MEASURED VARIABLES:				
Initial Volumes:				
INGEDIENT #1	V1	gal.	set by user	avoid type-3 disaster (out ofingrediant)
INGEDIENT #2	V2	gal.	set by user	avoid type-3 disaster (out ofingrediant)
INGEDIENT #3	V3	gal.	set by user	avoid type-3 disaster (out ofingrediant)
REACTOR	Vr	gal.	set by user(should=0)	system designed so no product left in reactor
HOLD TANK	Vh	gal.	set by user	range= 0 to Vhcap
FILL TANK	Vf	gal.	set by user	range= 0 to Vfcap
Initial Temperature:				
REACTOR	Tr	degrees C	set by user	ingredients propably would enter reactor at T=Ta

(TABLE IV-1c)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
FLAGS:				
Internal:				
FLAG #1	Ui	0/1	initially set=0	shutdown ingrediants part of system
FLAG #2	Ur	0/1	initially set=0	shutdown reactor part of system
FLAG #3	Us	0/1	initially set=0	shutdown "lower" part of system
FLAG #4	Um	0/1	initially set=0	shutdown entire system
External:				
DISASTER FLAG #1 (P1)	#1	0/1	set by user	pump #1 stuck off (type-1 disaster)
DISASTER FLAG #2 (P2)	#2	0/1	set by user	pump #2 stuck off (type-1 disaster)
DISASTER FLAG #3 (P3)	#3	0/1	set by user	pump #3 stuck off (type-1 disaster)
DISASTER FLAG #4 (P4)	#4	0/1	set by user	pump #4 stuck off (type-1 disaster)
DISASTER FLAG #5 (P5)	#5	0/1	set by user	pump #5 stuck off (type-1 disaster)
DISASTER FLAG #6 (P6)	#6	0/1	set by user	pump #6 stuck off (type-2 disaster)
DISASTER FLAG #7 (P7)	#7	0/1	set by user	pump #7 stuck off (type-2 disaster)
DISASTER FLAG #8 (HEATER)	#8	0/1	set by user	heater stuck off (type-1 disaster)
DISASTER FLAG #9 (CONVYR)	#9	0/1	set by user	conveyor stuck off (type-2 disaster)
DISASTER FLAG #10(noBTLS)	#10	0/1	set by user	out of bottles (type-2 disaster)

(TABLE IV-1d)

VARIABLE	SYMBOL	UNITS	VALUE ASSIGNED	RATIONALE FOR VALUE ASSIGNED
CONTROLS:				
ACTIVATE PUMPS	U1 to U8	0/1	initially set=0	
ACTIVATE CONVEYOR	Uc	0/1	initially set=0	
ACTIVATE HEATER	Uh	0/1	initially set=0	
TIME:				
PROCESS TIME	CUMTIME	seconds	initially set=0	
PROGRAM EXECUTION TIME	REALTIME	seconds	initially set=0	
PERFORMANCE CRITERIA:				
BOTTLING RATE	BOTLNGRATE	#/min	=#BOTTLES/CUMTIME	[see section II.B) CODING]
PERCENTAGE WASTE	%OVERFILL	percent	---------->	[see section II.B) CODING]
STATE VARIABLES:				
"FILL"	"FILL"	none	initially set	[see section II.B) CODING]
"HEAT"	"HEAT"	none	initially not set	[see section II.B) CODING]
"COOL"	"COOL"	none	initially not set	[see section II.B) CODING]
"DRAIN"	"DRAIN"	none	initially not set	[see section II.B) CODING]
OTHER INTERNAL VARIABLES:				
PRINT INDEX	PRINTINDEX	none	set by user	defines frequency of data writes
Misc. Internal Variables	-	none	initially set=0	[see section II.B) CODING]

(TABLE IV-2a) INITIAL CONDITIONS

				PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS		% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
				GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
CONSTANTS:															
TARGET HEAT	T_TrgtHEAT	degrees C		140	140	140	140	140	140	140	140	140	140	140	140
TARGET COOL	T_TrgtCOOL	degrees C		100	100	100	100	100	100	100	100	140	140	140	140
NONFLUCTUATING VARIABLES:															
Volumes:															
REACTOR CAPACITY	Vrcap	gal.	*	110	110	110	110	110	110	110	110	110	110	110	110
HOLD TANK CAPACITY	Vhcap	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
FILL TANK CAPACITY	Vfcap	gal.	*	3	3	3	3	3	3	3	3	3	3	3	3
BOTTLE CAPACITY	Vbcap	ounces	*	12	12	12	12	12	12	12	12	12	12	12	12
DESIRED PRODUCT VOLUME	Vrequest	gal.	*	300	300	300	300	300	300	300	300	2000	2000	2000	2000
INGEDIENT #1 TANK MIN.	V1min	gal.		f(h)	0.17	f(h)	0.17	f(P4)	0.17	0.17	0.17	0.04	0.04	0.04	0.04
INGEDIENT #2 TANK MIN.	V2min	gal.		f(h)	0.33	f(h)	0.33	f(P4)	0.33	0.33	0.33	0.08	0.08	0.08	0.08
INGEDIENT #3 TANK MIN.	V3min	gal.		f(h)	0.50	f(h)	0.50	f(P4)	0.50	0.50	0.50	0.13	0.13	0.13	0.13
REACTOR MIN.	Vrmin	gal.		f(h)	0.33	f(h)	0.33	f(P5)	0.33	0.33	0.33	0.17	0.17	0.17	0.17
HOLD TANK MIN	Vhmin	gal.		f(h)	0.13	f(h)	0.13	0.13	0.13	0.13	0.13	0.02	0.02	0.02	0.02
FILL TANK MIN	Vfmin	gal.		f(h)	f(P7)	f(h,P7)	f(P7)	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
FILL TANK LOW	Vflow	gal.		0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
REACTOR MAX.	Vrmax	gal.		f(h)	98.33	f(h)	98.33	f(P4)	98.33	98.33	98.33	99.58	99.58	99.58	99.58
HOLD TANK MAX.	Vhmax	gal.		f(h)	879.00	f(h)	879.00	f(P4)	879.00	879.00	879.00	879.00	879.00	879.00	879.00
FILL TANK MAX.	Vfmax	gal.		f(h)	2.87	f(h)	2.87	2.87	2.87	2.87	2.87	2.98	2.98	2.98	2.98
Pump Rates:															
PUMP RATE #1	P1	gal./min.		250	250	250	250	f(P4)	250	250	250	125	125	125	125
PUMP RATE #2	P2	gal./min.		500	500	500	500	f(P4)	500	500	500	250	250	250	250
PUMP RATE #3	P3	gal./min.		750	750	750	750	f(P4)	750	750	750	375	375	375	375
PUMP RATE #4	P4	gal./min.	*	1000	1000	1000	1000	TEST	1000	1000	1000	500	500	500	500
PUMP RATE #5	P5	gal./min.	*	500	500	500	500	TEST	500	500	500	500	500	500	500
PUMP RATE #6	P6	gal./min.	*	200	200	200	200	200	200	200	200	50	50	50	50
PUMP RATE #7	P7	gal./min.	*	28.125	TEST	TEST	TEST	28.125	28.125	28.125	28.125	28.125	28.125	28.125	28.125
PUMP RATE #8	P8	gal./min.	*	500	500	500	500	500	500	500	500	500	500	500	500
Heater Variables:															
AMBIENT TEMP.	Ta	degrees C	*	22	22	22	TEST	22	22	22	22	22	22	22	22
HEATER TEMP. SETTING	Te	degrees C	*	200	200	200	200	200	TEST	TEST	200	180	180	180	180
HEATER RATING	HTR	C/min	*	0.50	0.50	0.50	0.50	0.50	0.50	0.50	TEST	0.30	0.30	0.30	0.30
Integration Variable:															
SIMULATION TIME STEP	h	seconds	*	TEST	0.04	TEST	0.04	0.04	0.04	0.04	0.04	0.02	0.02	0.02	0.02

* = a variable which can be "set by user"

TEST = independent variable on graph (ie.: the "TEST" variable)

f(X) = initial value of variable is a function of the value of the "TEST" variable

(TABLE IV-2b) INITIAL CONDITIONS

				PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS		% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
				GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
SIMULATED MEASURED VARIABLES:															
Initial Volumes:															
INGEDIENT #1	V1	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	150
INGEDIENT #2	V2	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
INGEDIENT #3	V3	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
REACTOR	Vr	gal.	*	0	0	0	0	0	0	0	0	0	0	0	0
HOLD TANK	Vh	gal.	*	100	100	100	100	100	100	100	100	100	100	100	100
FILL TANK	Vf	gal.	*	1	1	1	1	1	1	1	1	1	1	1	1
Initial Temperature:															
REACTOR	Tr	degrees C	*	22	22	22	f(Ta)	22	22	22	22	22	22	22	22

* = a variable which can be "set by user"
TEST = independent variable on graph (ie.: the "TEST" variable)
f(X) = initial value of variable is a function of the value of the "TEST" variable

(TABLE IV-2c) INITIAL CONDITIONS

			PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS	% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
			GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
FLAGS:														
Internal:														
FLAG #1	Ui	0/1	0	0	0	0	0	0	0	0	0	0	0	0
FLAG #2	Ur	0/1	0	0	0	0	0	0	0	0	0	0	0	0
FLAG #3	Us	0/1	0	0	0	0	0	0	0	0	0	0	0	0
FLAG #4	Um	0/1	0	0	0	0	0	0	0	0	0	0	0	0
External:														
DISASTER FLAG #1 (P1)	#1	0/1 *	0	0	0	0	0	0	0	0	0	1	0	0
DISASTER FLAG #2 (P2)	#2	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #3 (P3)	#3	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #4 (P4)	#4	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #5 (P5)	#5	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #6 (P6)	#6	0/1 *	0	0	0	0	0	0	0	0	0	0	1	0
DISASTER FLAG #7 (P7)	#7	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #8 (HTER)	#8	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #9 (CONV)	#9	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #10(BTLS)	#10	0/1 *	0	0	0	0	0	0	0	0	0	0	0	0

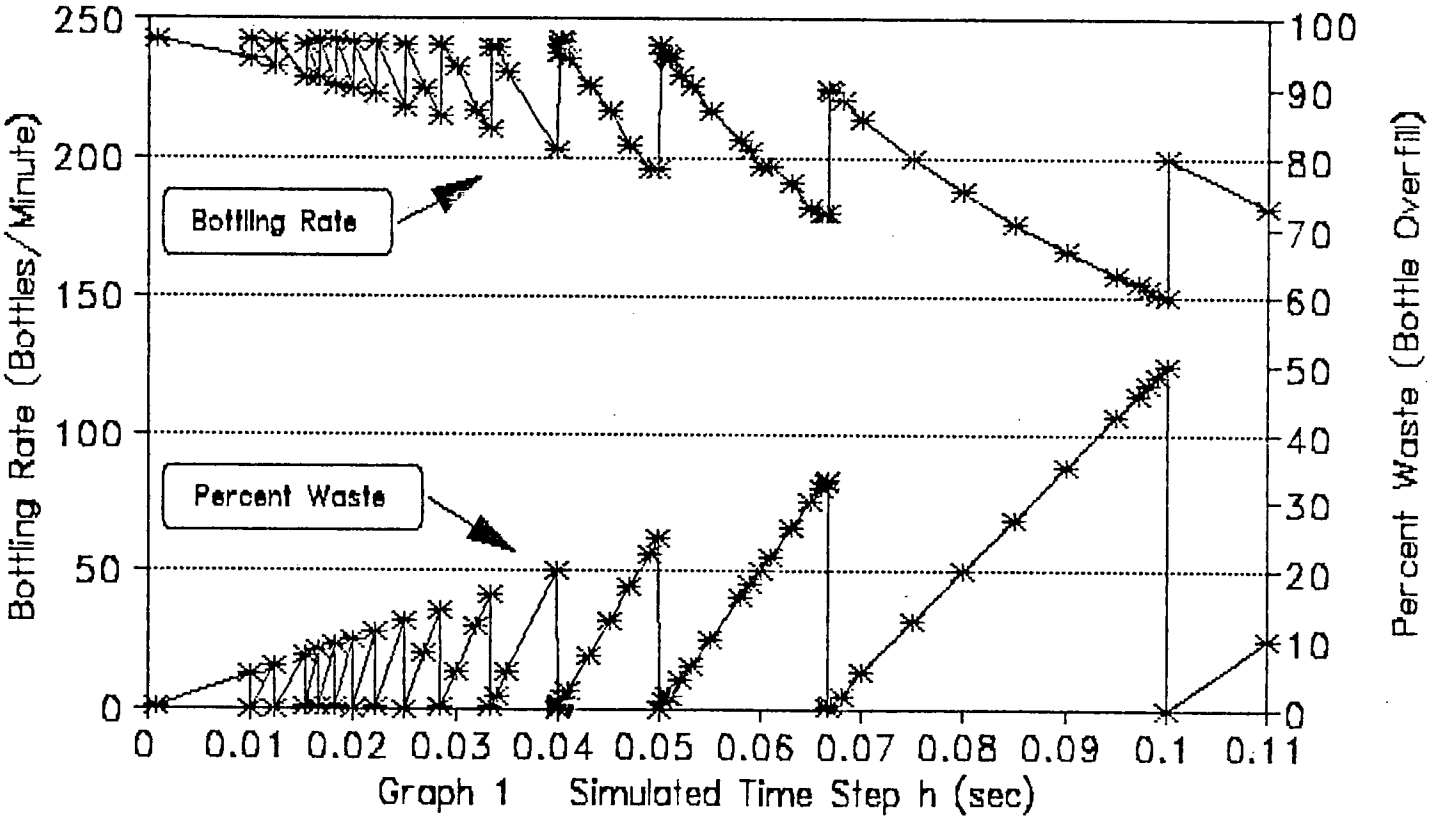
* = a variable which can be "set by user"

(TABLE IV-2d) INITIAL CONDITIONS

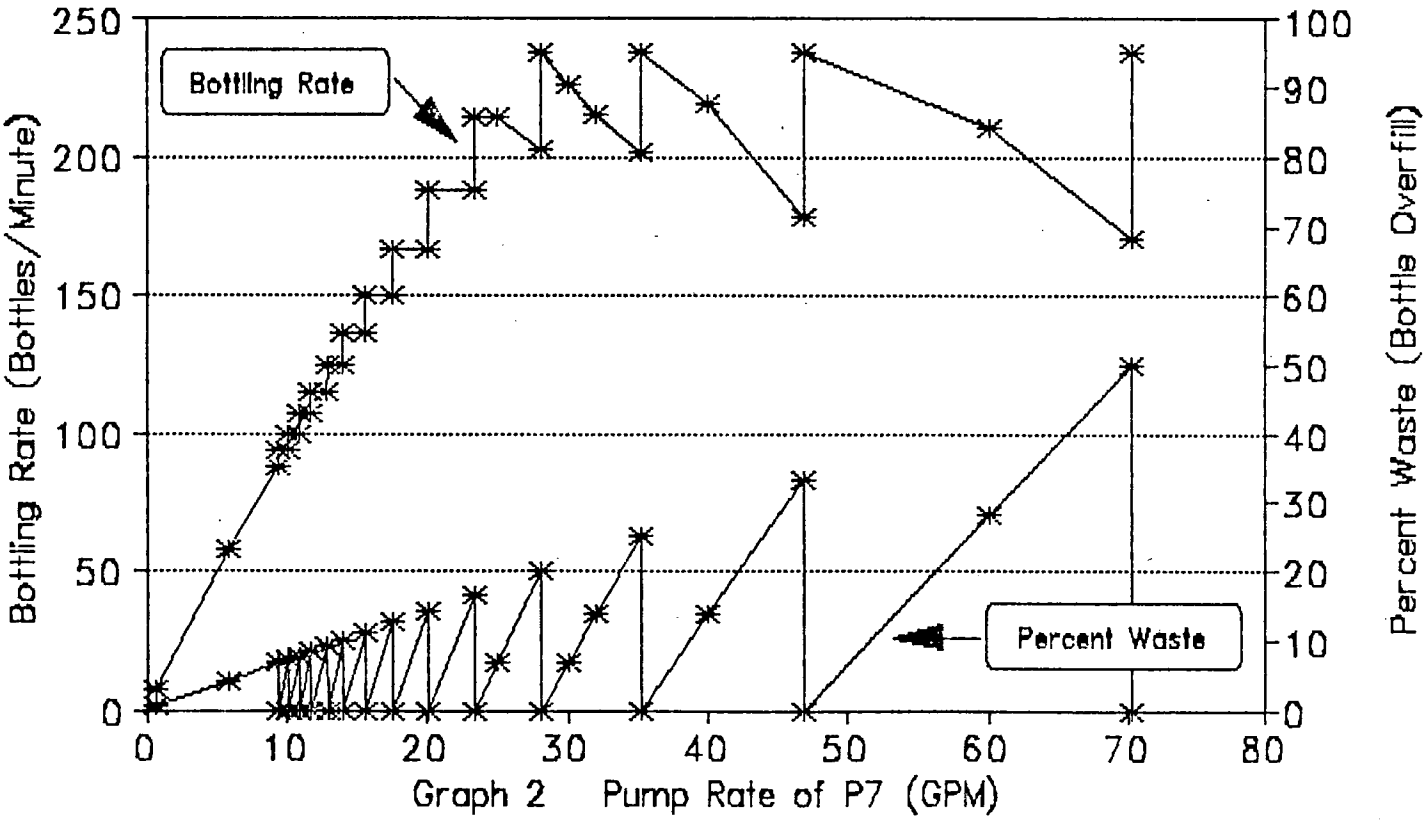
			PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS	% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
			GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
CONTROLS:														
ACTIVATE PUMPS	U1 to U8	0/1	0	0	0	0	0	0	0	0	0	0	0	0
ACTIVATE CONVEYOR	Uc	0/1	0	0	0	0	0	0	0	0	0	0	0	0
ACTIVATE HEATER	Uh	0/1	0	0	0	0	0	0	0	0	0	0	0	0
TIME:														
PROCESS TIME	CUMTIME	seconds	0	0	0	0	0	0	0	0	0	0	0	0
PROGRAM EXECUTION TIME	REALTIME	seconds	0	0	0	0	0	0	0	0	0	0	0	0
PERFORMANCE CRITERIA:														
BOTTLING RATE	BOTLNGRATE	#/min	0	0	0	0	0	0	0	0	0	0	0	0
PERCENTAGE WASTE	%OVERFILL	percent	0	0	0	0	0	0	0	0	0	0	0	0
STATE VARIABLES:														
"FILL"	"FILL"	none	0	0	0	0	0	0	0	0	0	0	0	0
"HEAT"	"HEAT"	none	0	0	0	0	0	0	0	0	0	0	0	0
"COOL"	"COOL"	none	0	0	0	0	0	0	0	0	0	0	0	0
"DRAIN"	"DRAIN"	none	0	0	0	0	0	0	0	0	0	0	0	0
OTHER INTERNAL VARIABLES:														
PRINT INDEX	PRINTINDEX	none	20	20	20	20	20	20	20	20	200	20	20	20
Misc. Internal Variable	-	none	-	-	-	-	-	-	-	-	-	-	-	-

* = a variable which can be "set by user"

Percent Waste & Bottling Rate vs. h



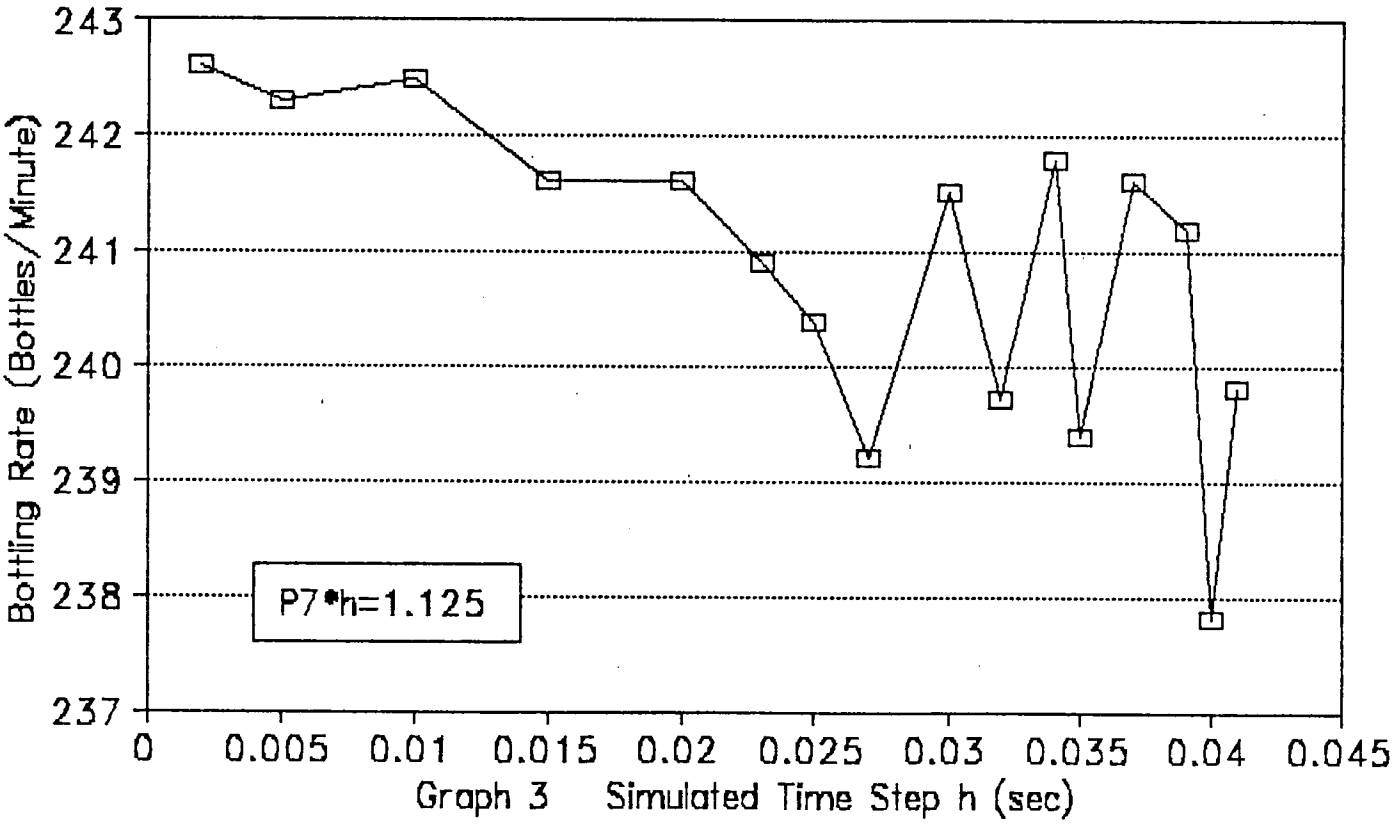
Percent Waste & Bottling Rate vs. P7



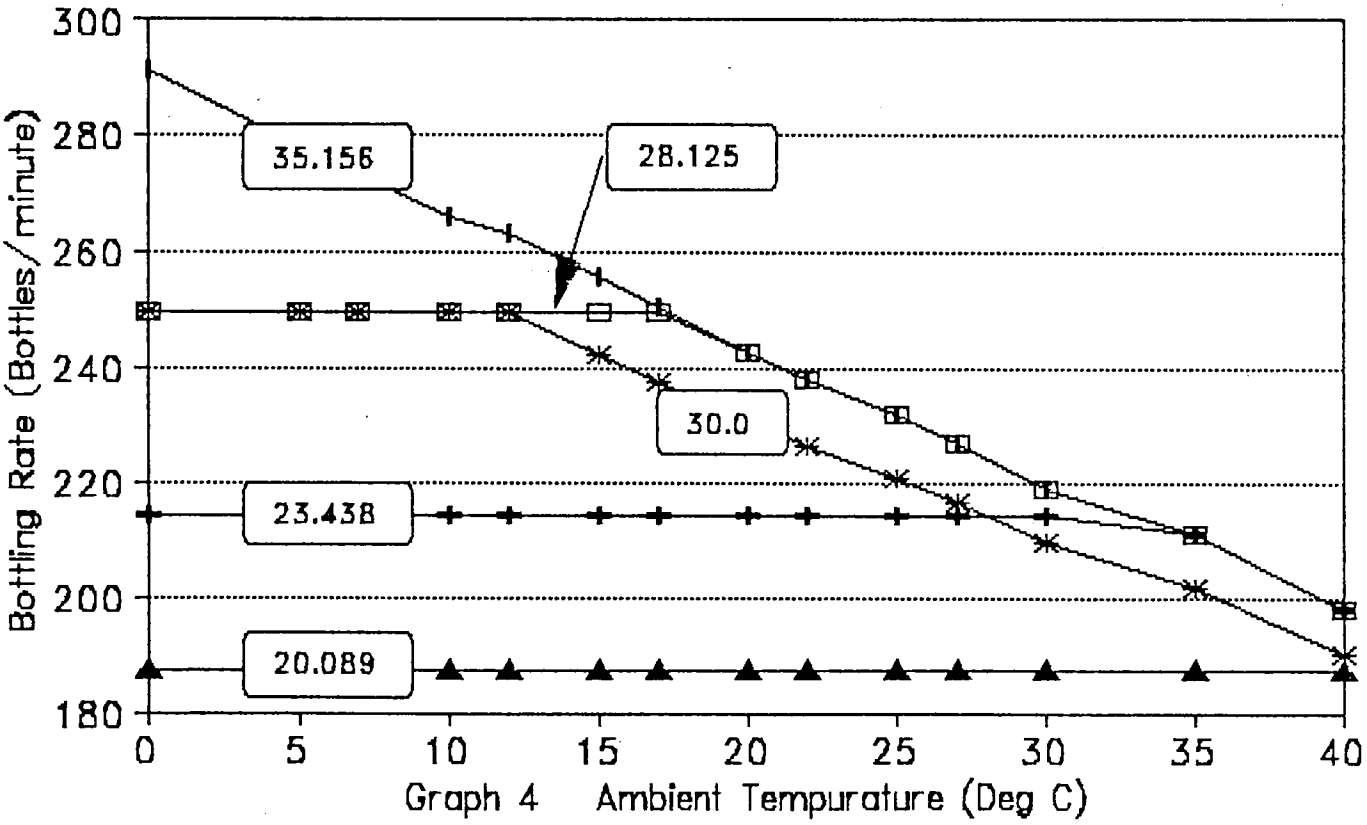
Graph 2

Bottling Rate vs. h

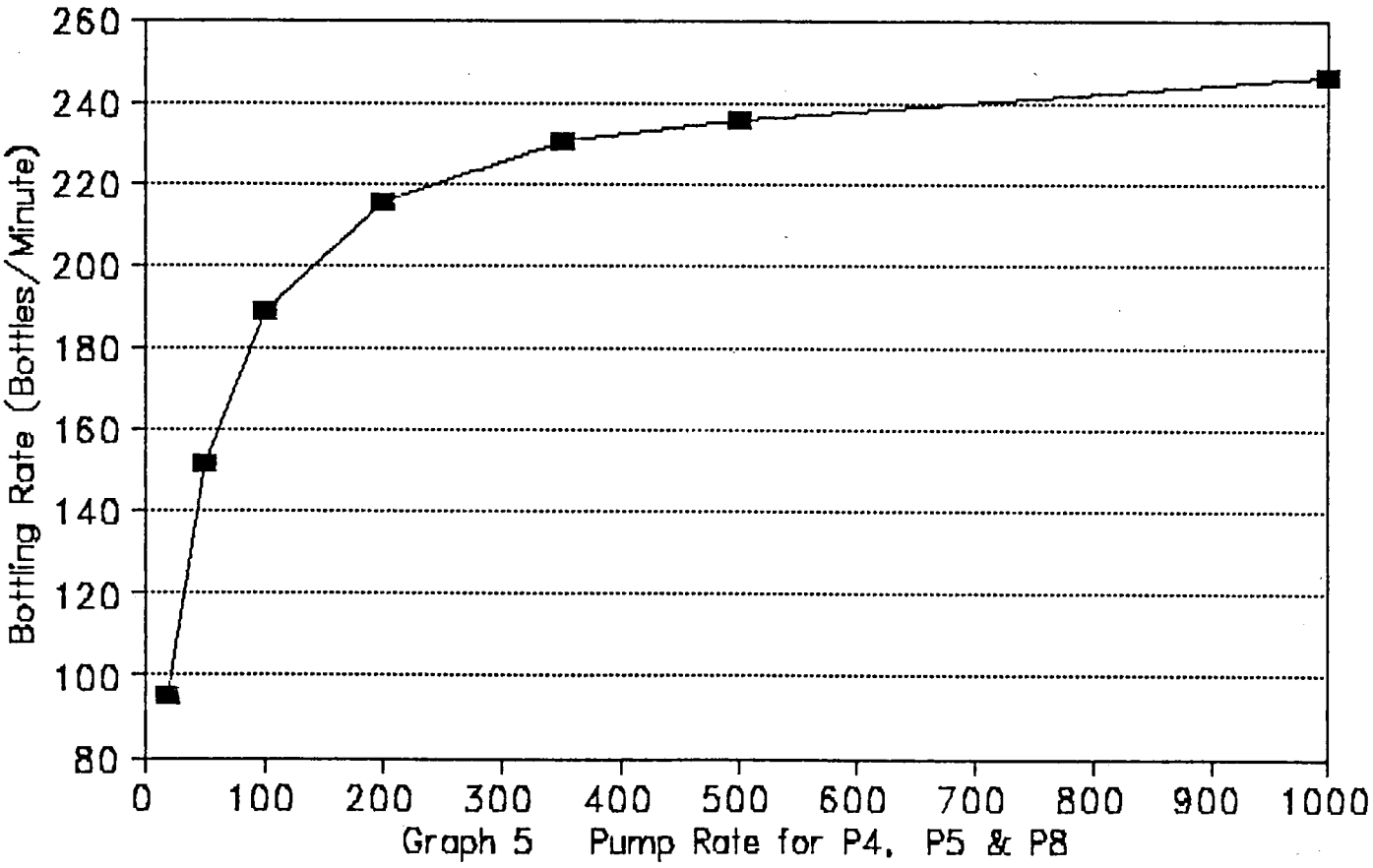
With P7*h Held Constant



Bottling Rate vs. Ambient Temperature



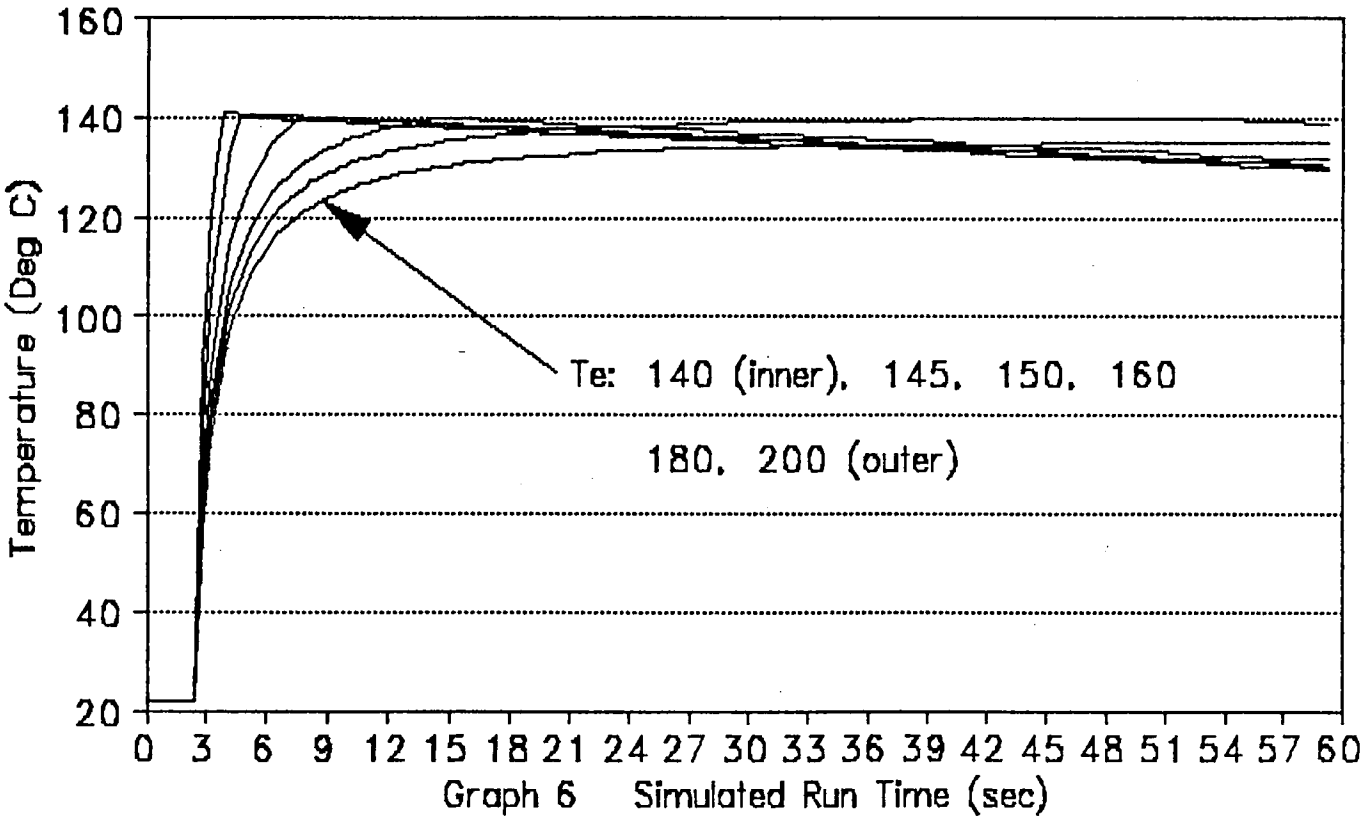
Bottling Rate vs. Max Pump Rates



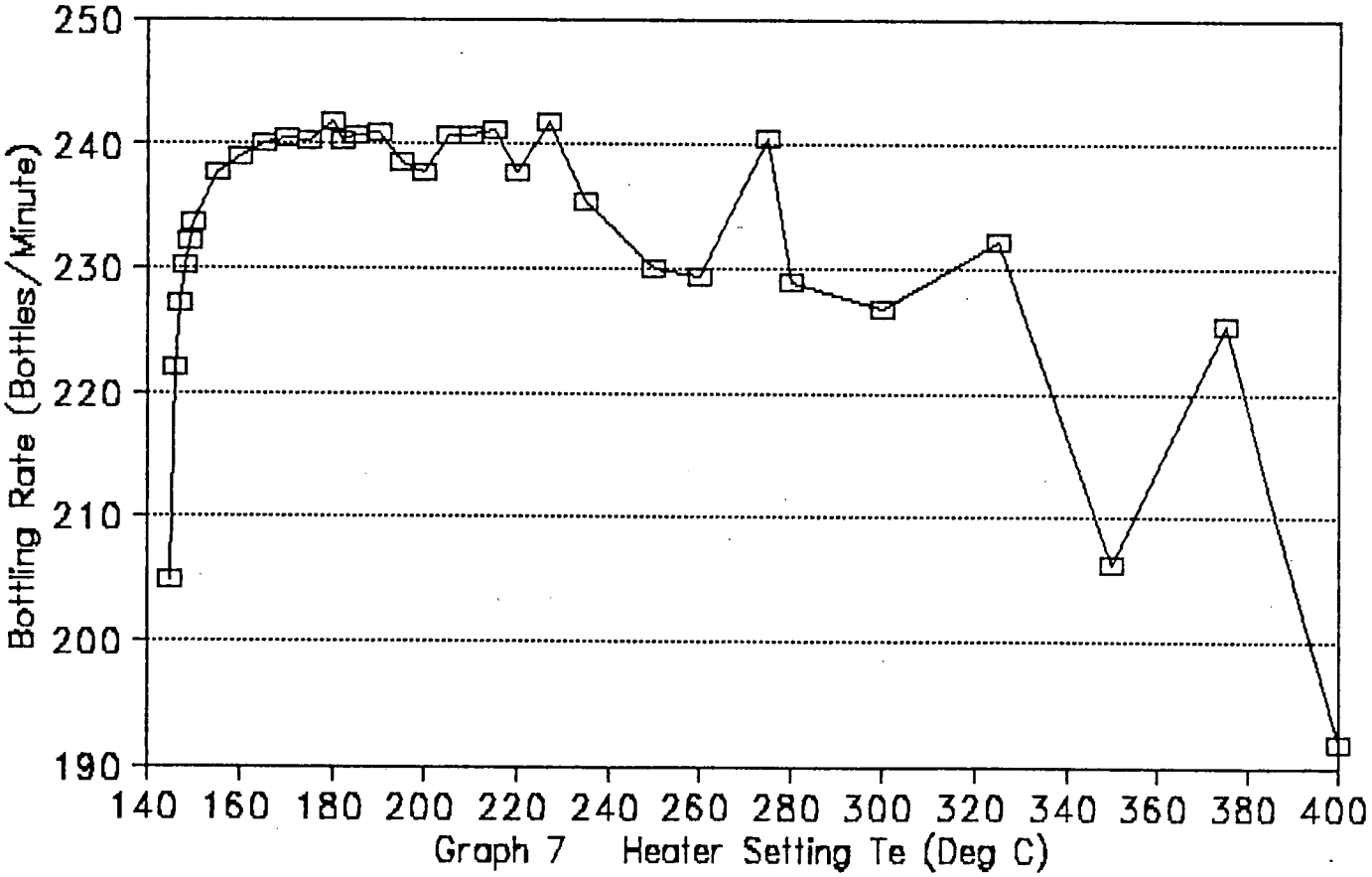
Graph 5 Pump Rate for P4, P5 & P8

Reactor Temperature vs. Time

For Various Values of Te

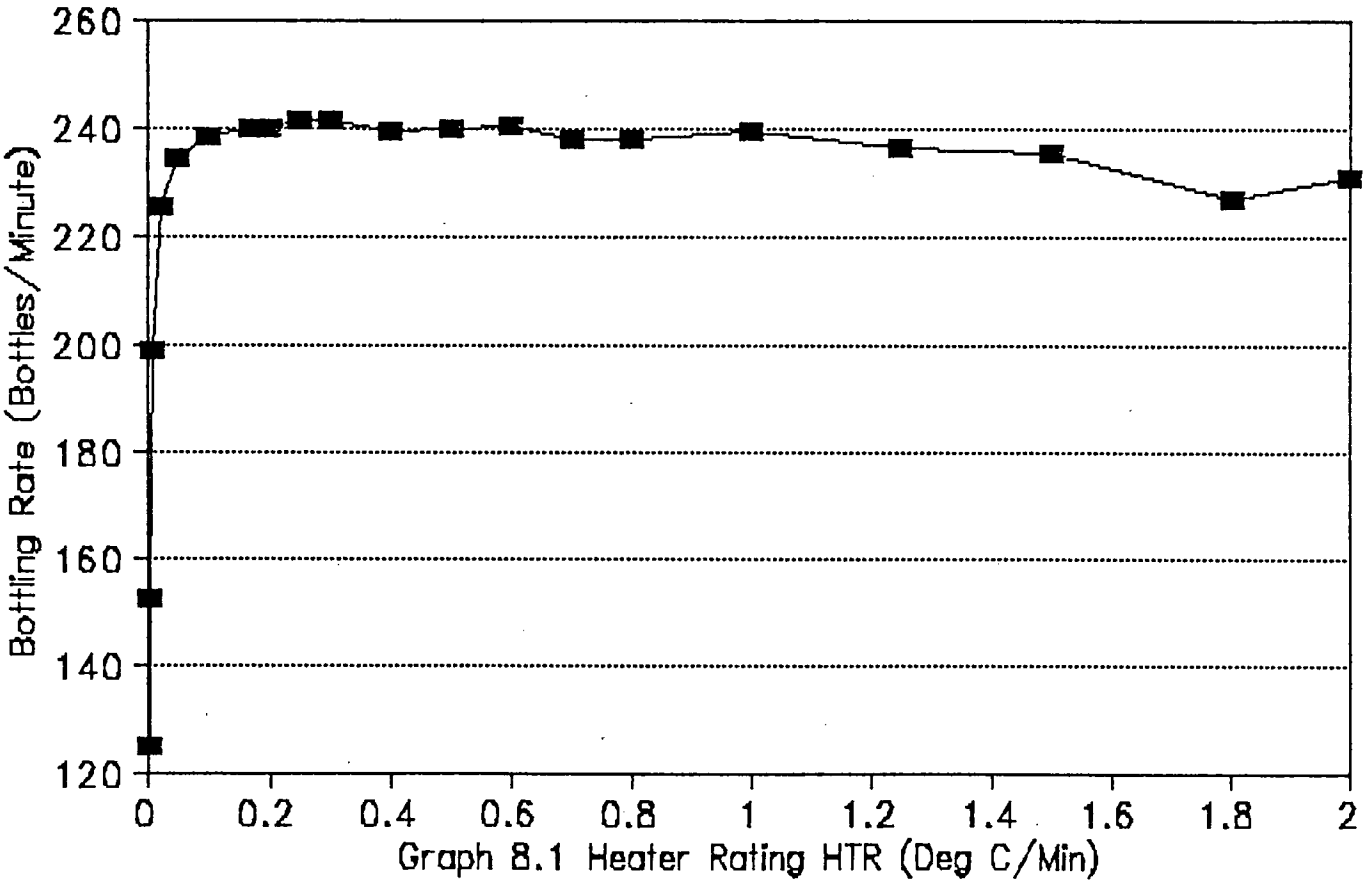


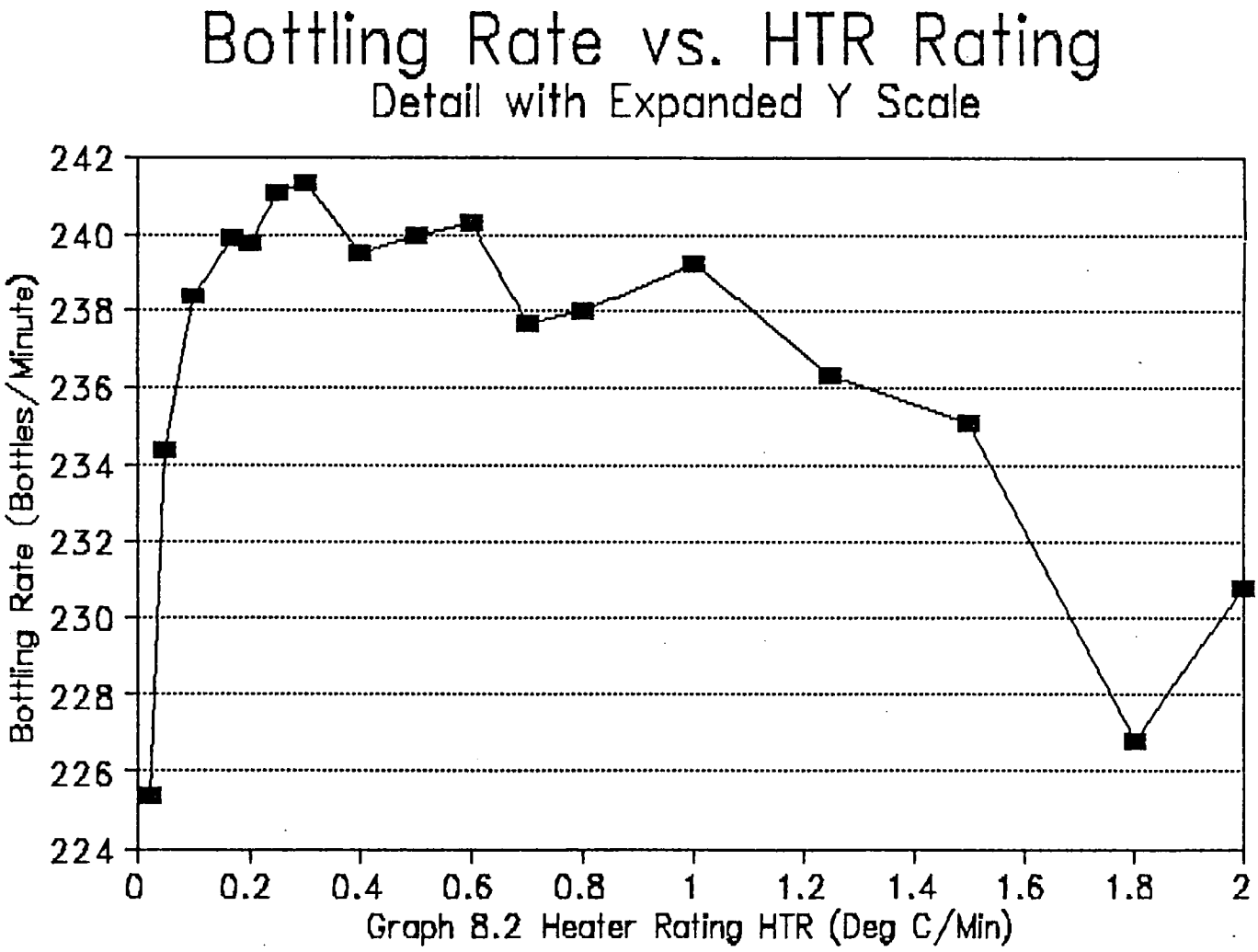
Bottling Rate vs. Te



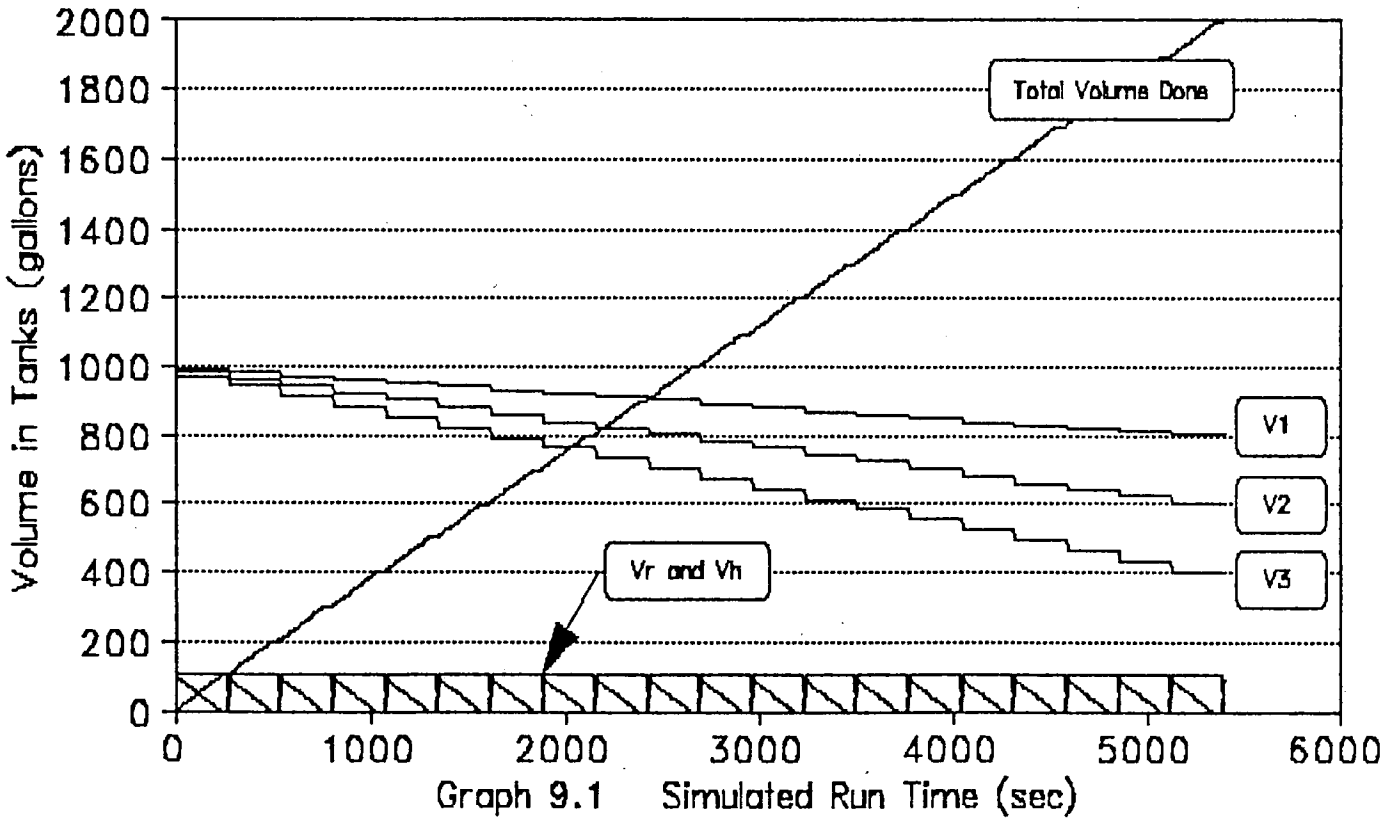
Graph 7

Bottling Rate vs. HTR Rating




Bottling Rate vs. HTR Rating
Detail with Expanded Y Scale
Bottling Rate (Bottles/Minute)
242
240
238
236
234
232
230
228
226
224
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Graph 8.2 Heater Rating HTR (Deg C/Min)

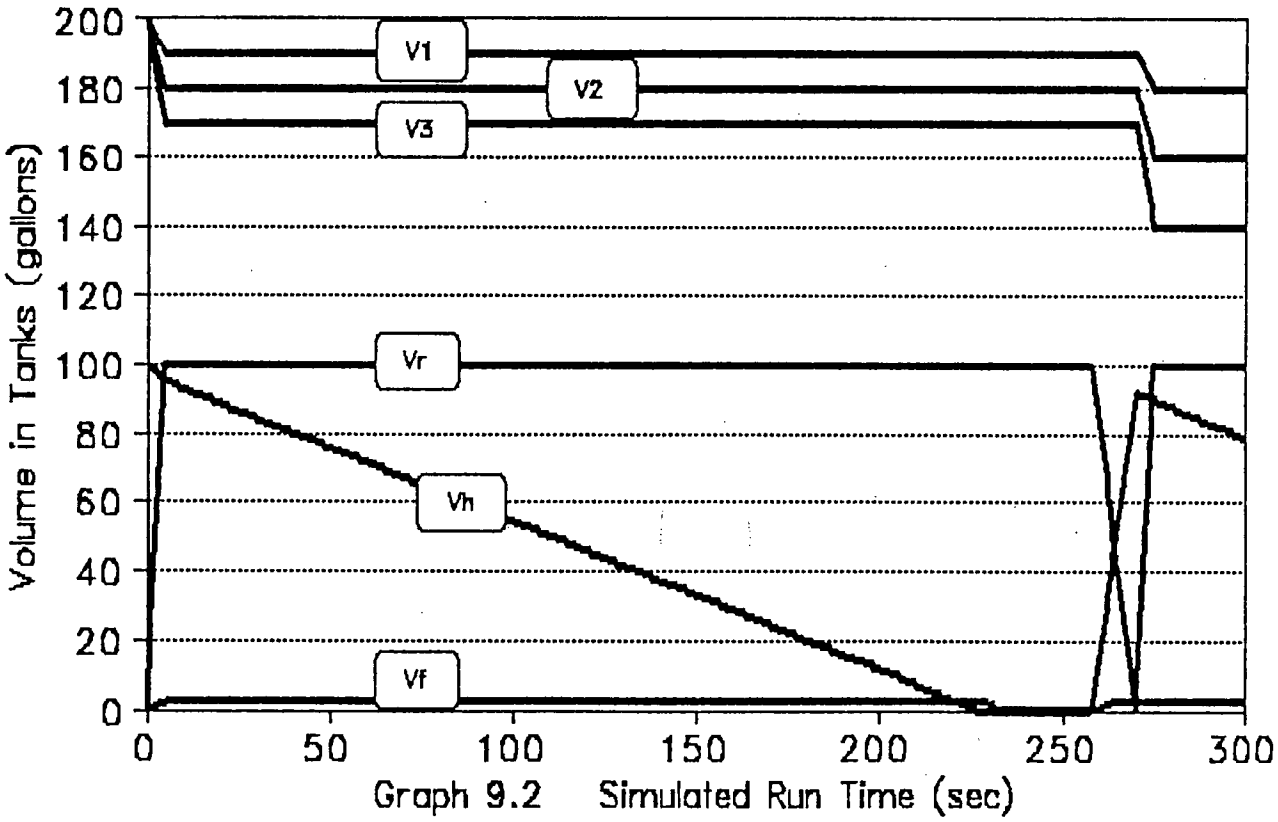
Volume in Tanks vs. Run Time



Graph 9.1

Volume in Tanks vs. Run Time

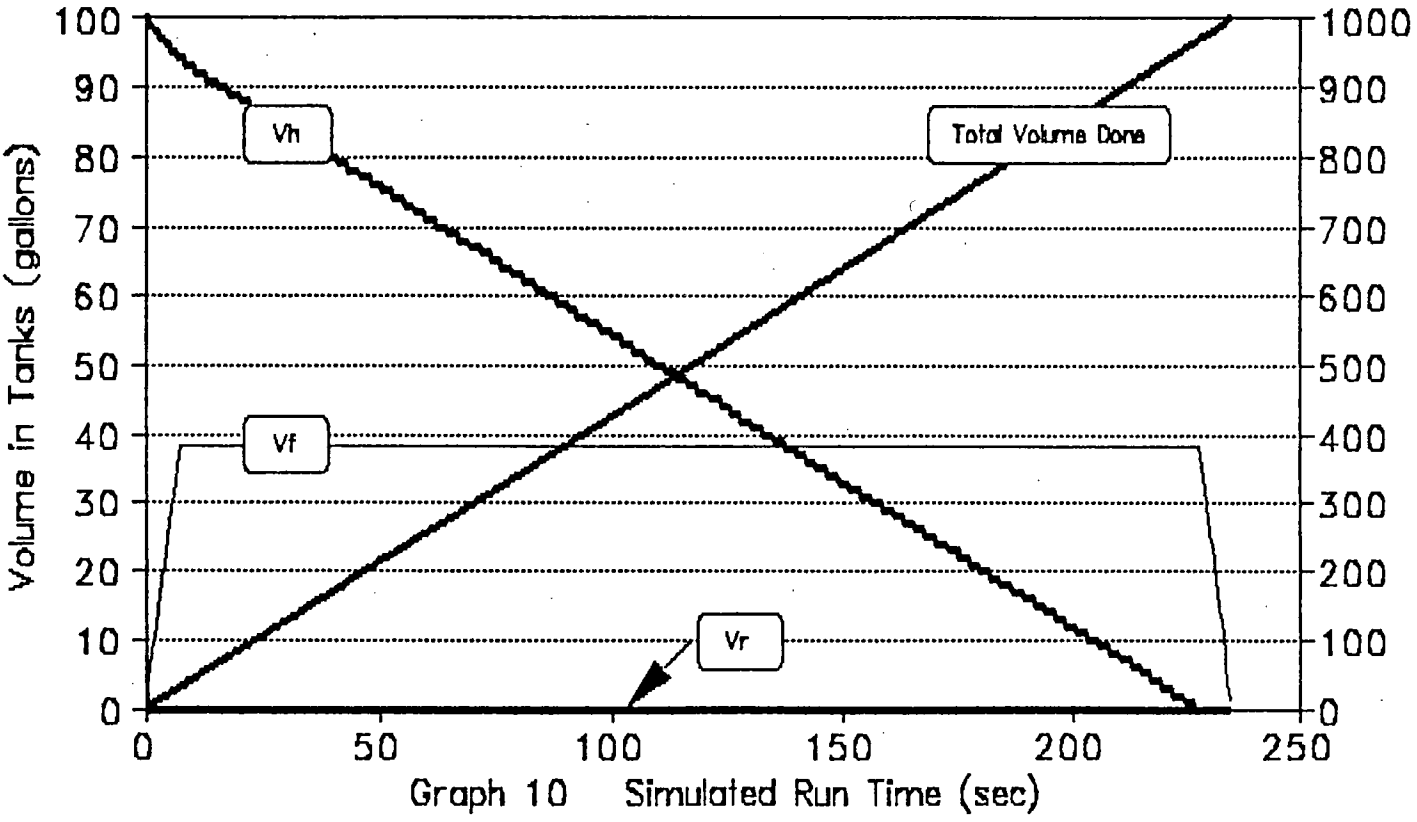
One Cook Cycle



Graph 9.2

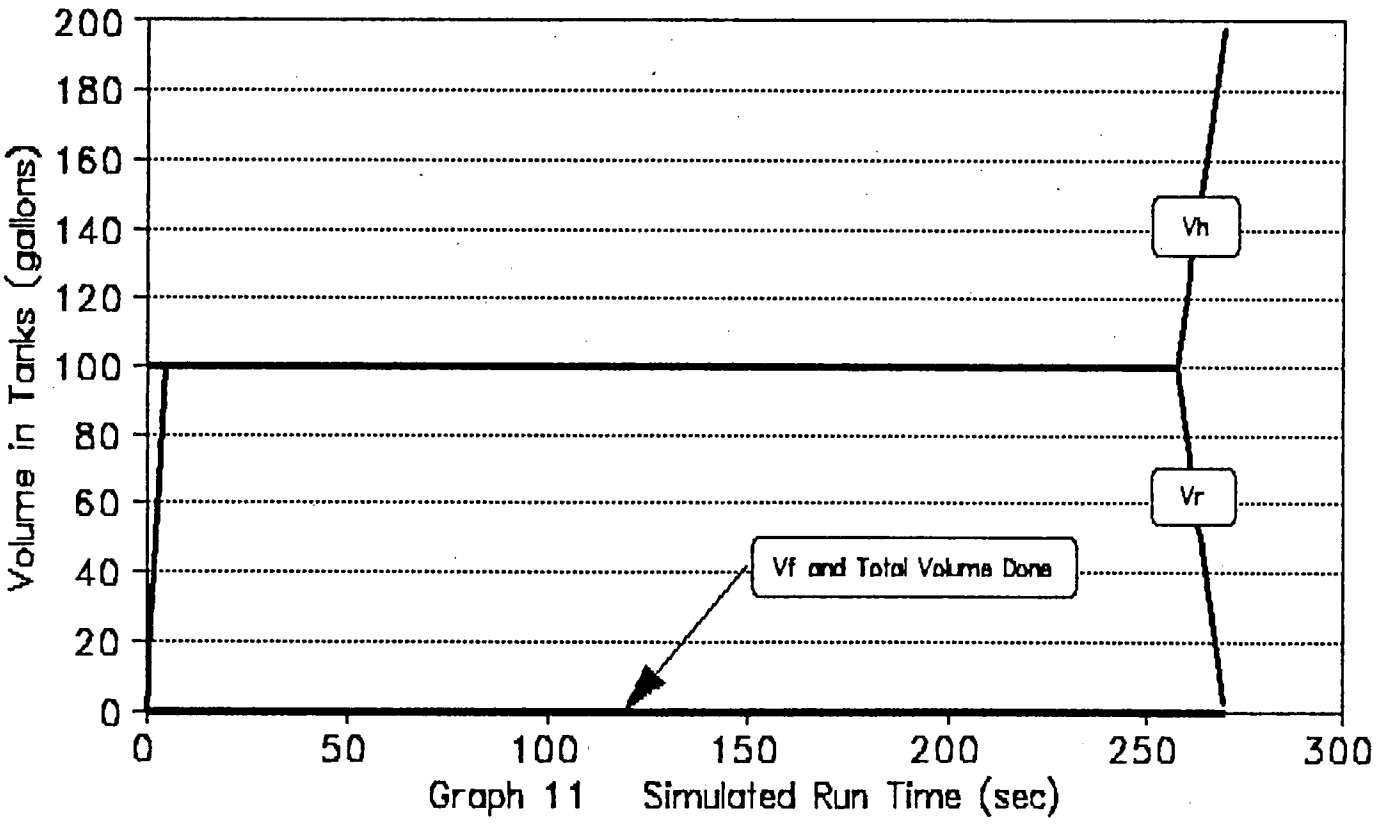
Volume in Tanks vs. Run Time

P1 Stuck at Off



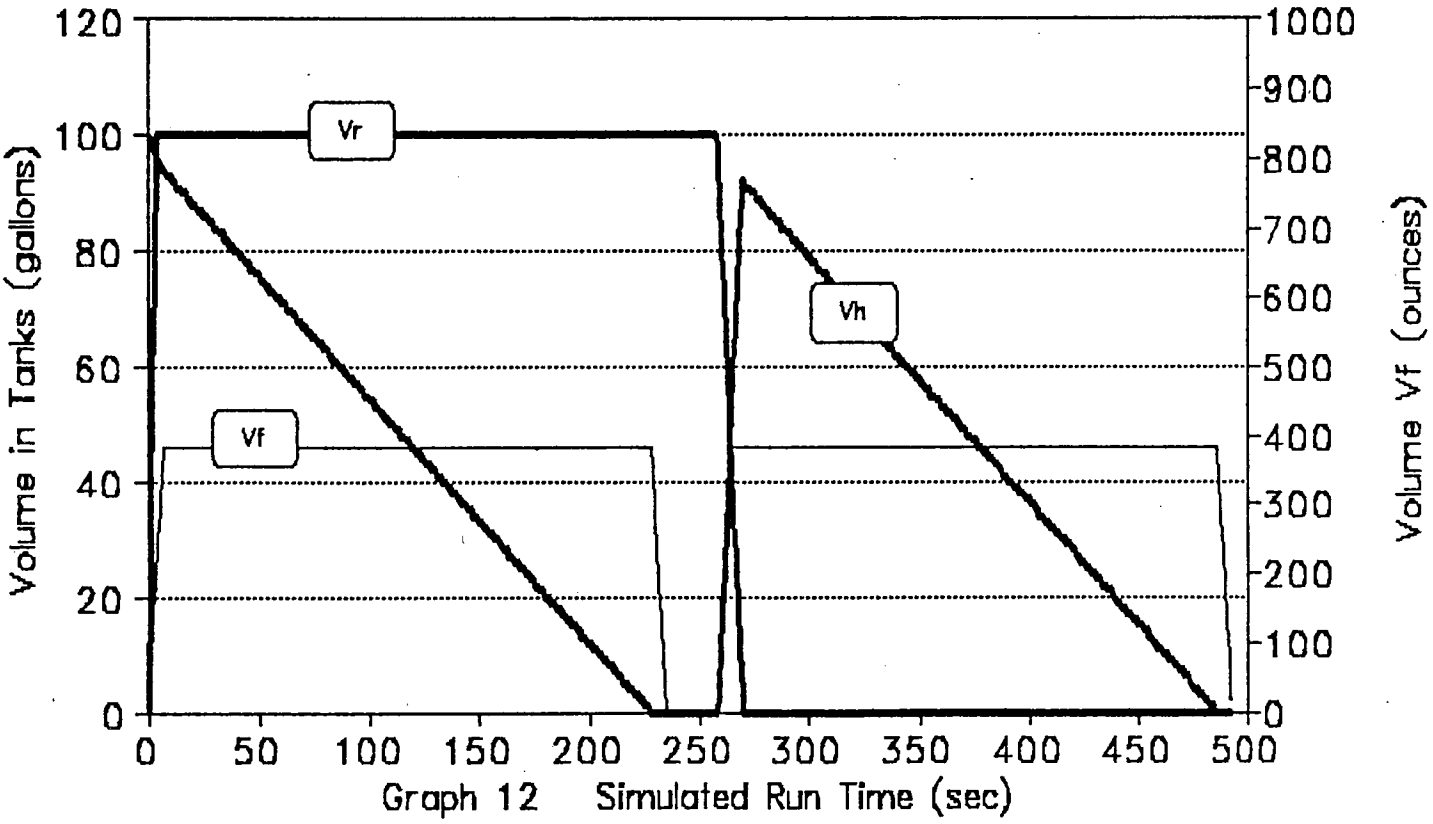
Volume in Tanks vs. Run Time

P6 Stuck at Off



Volume in Tanks vs. Run Time

Low V1 Volume After One Cook Cycle



Graph 12

V. DISCUSSION OF RESULTS

The testing of this simulation consisted of a three phase process of analysis. The primary and secondary analysis led to a set of system parameters (user changeable inputs) used as initial values for the final analysis. [See highlighted values on Tables V-1a, b, c, d.] The desired volume of product used for the primary and secondary analyses was 300 gallons; 2,000 gallons was used for the final analysis. This did cause decreases in the bottling rate under certain conditions (e.g. initial tank volumes had more significance when V_{req} = 300 gallons). However, the decreases in the simulation program run time (as much as 30 minutes for one data point) lead to a much larger set of data points for analysis then would have otherwise been feasible. Also, the objective of testing the simulation was to observe the relative significance and interrelation of each of the test variables, (i.e., finding exact optimal values for initial inputs was not as important).

The following analysis will refer to the graphs at the end of this section; these graphs are identical to the graphs in [Section IV] except that the graphs in this section have been highlighted to emphasize certain data points and functional behavior.

V-A) PRIMARY ANALYSIS

The primary analysis of this simulation involved the concurrent evaluation of the effects of the simulation time step (h) and Pump Rate #7 (P7) on the systems performance (bottling rate and percentage waste). The optimization of the system's performance consists of maximizing the bottling rate and minimizing the percentage waste. One of the first considerations towards fulfilling this objective was to attempt to find unique combinations of (P7) and (h) that would result in exactly 12 ounces being put into each bottle. This mathematical phenomenon can be compared to the influence of a large sampling rate in an actual process control program. Since the mathematical influence of h is inherent in using Euler's approximation method of integration, the idea of "overfill" due to h must be considered. The curves of Graph #1 and #2 take on saw-tooth shapes mainly because of the overfill due to h. However, the inaccuracy of Euler's method at higher values of the independent variable also contribute to large errors as the graph is traced to the right [see Section VI) Conclusions]. For this reason an (h) of 0.05 is shown on Graph #1 to be the largest value that (h) may be before affects, other than being in synch with (P7), begin having a major significance [i.e., the high points of the saw tooth curve (bottling rate vs. h) cease to remain at a relatively constant bottling rate beyond h = 0.05]. An (h) value of 0.04 was used

for the remaining graphs of the primary and secondary analysis merely because of the relatively short simulation run time associated with h = 0.04. However, a smaller h was used for the final run. An example of the critical relationship between (h) and (P7) is as follows:

Case-1) If h = .04 and P7 = 28.125

$$F7 * h = \left(\frac{28.125 \text{ gpm}}{\text{min.}}\right)\left(\frac{.04 \text{ sec.}}{\text{Cycle}}\right)\left(\frac{1 \text{ min.}}{60 \text{ sec.}}\right)\left(\frac{128 \text{ ozs.}}{1 \text{ gal.}}\right) = 2.400 \text{ ozs.}$$

Case-2) If h = .04 and P7 = 28.124

F7 * h = 2.399 ozs.

In the first case (h) and (P7) form what we have termed as a "critical-pair" which cause an exact filling of the bottle. This occurs just before the conditional program statement which shifts in a new bottle if the volume just put into a bottle is calculated to be greater than or equal to 12 ounces. In Case-2, the bottle does not fill in exactly five cycles of (h) as in Case-1; here the program must loop one more time before the conditional statement is satisfied; the bottle is overfilled by [(2.399 * 6 cycles) - 12] = 2.399 ounces. This corresponds to 19.9% waste for producing this bottle.

The "critical-pairs" for h = 0.04 are illustrated in Graph #2 (i.e., each value of P7 for which percent waste is zero and bottling rate locally peaks). The straight line drawn on Graph #2 (orange color) represents the true linear behavior of (P7) vs. bottling rate if:

(a) Pump #7 never turned off during a program run.
(b) Every bottle filled exactly to 12 ounces.
(c) The integration method used resulted in true solutions.

This is demonstrated by considering (P7=28.125) (circled on Graph #2). For this value of (P7), the bottle exactly fills in five cycles of h; then one cycle of h is used to shift the bottles. The difference (on Graph #2) between the bottling rate peak on the sawtooth curve (237.8) and that on the straight line curve (28.125 gal./min. * 178/60 = 300.0) can be mostly accounted for by factoring out the time waiting for the bottles to shift:

$$\frac{6}{5} * 237.8 = 285.36$$

However, there is a (14.64) bottle discrepancy; most of this can be contributed to (P7) shutting down; waiting for a cook batch to cool as will be discussed in the secondary analysis. However, there is also a two bottle error which can be contributed to Euler's method of integration. All of the (P7) "critical points" below 28.125 (i.e., 23.438, 20.839, etc.) can be completely accounted for by the time delay in shifting bottles. As (P7) is increased above 28.125, the rest of the system increasingly can't

maintain enough volume in the fill tank; bottling must be suspended during the program run and consequently the bottling rate decreases even further. Also, as (P7) increases above (28.125), the inaccuracies of Euler's method become greatly exaggerated.

Graph #3 illustrates the change in bottling rate as (P7) and (h) are changed concurrently while maintaining the same ratio as that shown to be a "critical pair" in Graph #2. From Graph #2 the critical pair of [h = 0.04 and P7 = 28.125] (no waste) was chosen to form a "critical ratio" equal to (h * P7) = (0.04 * 28.125) = 1.125. This ratio was maintained constant in Graph #3 while changing (h). The deep valleys of the sawtoothed bottling rate curves of Graph #1 and Graph #2 seem to have been eliminated as (P7) is varied simultaneously with (h) on Graph #3 through the range: (P7 = 562 @ h = .002) to (P7 = 27.439 @ h = .041). The behavior at this curve seems to imply that the Euler method possibly becomes more significantly inaccurate at (h) values greater than 0.025. Another possibility would be the affect of decreasing the bottling rate due to waiting for the cook cycle while increasing the bottling rate due to greater (P7) as h is decreased on this graph. Whichever the case an (h = 0.02) would be below the (h) values on this graph where erratic data is located. An (h = 0.02) would also be a "critical pair" with (P7 = 28.125). Therefore, (h = 0.02) was used for the final analysis.

V-B SECONDARY ANALYSIS

The secondary analysis of the testing of the simulation was performed somewhat concurrently with the primary analysis; the volume requested was kept at 300 gallons and (h) was kept at (0.04) to speed up the simulation runs. A pump rate of (P7 = 28.125) was taken from the primary analysis and used for the secondary analysis. Since (h = .04, P7 = 28.125) is a "critical pair", as defined previously, the overfill (waste) problem is eliminated from the secondary analysis. Only the bottling rate performance criteria will be considered.

The significance of the ambient temperature of the air (t_a) surrounding the reactor is shown in Graph #4. The curve highlighted in orange is for (P7 = 28.125). It can be seen that the bottling rate remains constant for (P7 = 28.125) at temperatures below $17^{o}C$; this was confirmed down to absolute zero. However, as (T_a) is increased above $17^{o}C$, the bottling rate decreases linearly; this is due to the increased cooling time of the reactor batches. This resulted in a decrease in the bottling rate because bottling was suspended waiting on the reactor to cool. The curve highlighted in green is for (P7 = 30); since this is not a "critical pair" for (h = 0.04), the bottling rate begins to decrease at ($T_a > 12^{o}C$) rather than ($T_a > 17^{o}C$).

Graph #4 would be a helpful aid for making a decision of whether or not to artificially increase the cooling of the reactor or possibly change (P7). However, since artificially increasing the cooling of cook batches is not incorporated into this simulation, the only variable of consideration here is (P7). The (T_A) used for the final analysis was therefore 22°C (room temperature). A (P7 = 28.125) remains the optimal value since even at ambient temperature, the bottling rate is significantly higher (237.8) than that of the next lowest critical pair: [(h = .04, P7 = 23.438) which corresponds to a bottling rate of (214.3)]. The only drawback of (P7 = 28.125) is the injection velocity and resulting forces put on a bottle being filled this rapidly. If the bottle is to be filled by a 1/2" nozzle at (28.125 gal./min.), the velocity of the injection stream would be:

$$\left(\frac{28.125\ \text{gal.}}{\text{min.}}\right)*\left(\frac{1\ \text{ft.}^3}{7.48\ \text{gal.}}\right)*\left(\frac{1}{(.25)^2\ \text{in.}^2}\right)*\left(\frac{144\ \text{in.}^2}{1\ \text{ft.}^2}\right)*\left(\frac{1\ \text{min.}}{60\ \text{sec.}}\right) = 45.9\ \frac{\text{ft.}}{\text{sec.}}$$

This is extremely fast for filling a bottle. But, if the fluid is of a low enough density to reduce the impact forces and if the nozzle is drawn out of the bottle at the exact rate the bottle fills, then (45.9 ft./sec.) may be feasible. Therefore (P7 = 28.125) is used for the final analysis also.

A similar problem of selecting maximum pump rates for (P4) and (P5) was encountered. A range of up to (1,000 gal./min.) was

considered. Flow at (1,000 gal./min.) through (4") diameter pipes would move at (25.53 ft./sec.). This rate seems feasible, however, Graph #5 shows that the increasing of (P4) and (P5) from (500 gal./min.) to (1,000 gal./min.) had relatively little affect on the bottling rate as compared to changes in (P4) and (P5) from (0.0 gal./min.) to (500 gal./min.). Therefore, (P4 = P5 = 500) was chosen for the final analysis.

The other three ingredient pump rates therefore are: (P1 = 1/4 P4 = 125), (P2 = 1/2 P4 = 250), (P3 = 3/4 P4 = 375) to maintain the specified recipe (1/2/3/4). The resulting pumping into the reactor is (P1 + P2 + P3 + P4 = 1,150). This is much larger than the (P5 = 500) pumping out of the reactor, however, the time delay of reactor batches is additive (i.e.: pumping time + heat time + cool time + pump out time) and, therefore, the only consideration in picking (P4) and (P5) is determining a feasible maximum pump rate. The pump rate (P6) need only be approximately the same speed as (P7) to maintain fluid in the fill tank. Therefore (P6 = 50) was used in the final analysis.

The reactor heat balance equation had a major significance on the bottling rate since bottling is suspended if product can't be made in the reactor fast enough. Although the pump rates into and out of the reactor are significant variables, the heating and cooling of the product have the most impact on the system; mostly the cooling. The curve of Graph #7 shows that the heater temperature setting (T_e) must be greater than the target heat

($T\text{-}Tr_g$tHeat = 140°C) or else the reactor temperature (T_r) will never reach (140°C). The curve of Graph #7 confirms this. Graph #7 also shows that if (T_e) is set too high that apparent math errors due to h reduce the bottling rate [see Section VI Conclusions]. The large length of time for cooling could not be shown on Graph #6 because of the relative magnitude of cooling time compared to heating time (i.e.: much longer). This relative difference is because the heat balance equation is such that the reactor heats up at an (X^2) rate and cools down at a (1/10 X) rate. The heater temperature setting (T_e = 180°C) was used in the final analysis and was chosen from Graph #7. This value is large enough (20°C greater than $T\text{-}Tr_g$tHeat) to overcome the asymptotic behavior shown in Graph #6 while being small enough to provide a relatively error-free value.

The affect of the heater rating (HTR) on the bottling rate is shown on Graph #8 and its attached detail. The bottling rate increased rapidly when (HTR) was varied from (0.0) to (0.3), then it leveled off. Therefore, (HTR = 0.03°C/min.) was chosen for the final analysis.

V-C FINAL ANALYSIS

The final analysis consisted of the observations of tank volumes during a simulation run. The initial conditions used were chosen

based on the primary and secondary analysis of certain "test variables". The primary and secondary analysis, as previously discussed, lead to the following values used for the final analysis: (V_{req} = 2,000), (P4 = 500), (P5 = 500), (P6 = 50), (P7 = 28.125), (T_a = 22^{o}C), (T_e = 180^{o}C), (HTR = 0.03) and (h = 0.02) [see Tables V-1a, b, c, d]. All of the other variables either: (1) were not changed by the user for the final analysis or (2) changed on their own because they are a function of other variables (e.g.: (P1 = 125), (P2 = 250), (P3 = 375), etc.).

The internal behavior of the system can be determined by observing the volume changes throughout the simulated time to produce 2,000 gallons of product. This is illustrated in [Graph #9] and [Graph #9 (Detail)]. The fill tank volume (V_F) can not be shown on [Graph #9] because of the small magnitude of this number compared to the volume levels shown on this graph. However, (V_F) can be seen on [Graph #9 (Detail)] which was plotted for (V_{req} = 200) solely for the purpose of viewing all the tank volumes simultaneously. The time that bottling must be suspended is identified on both graphs by the orange highlighting. It can be seen in [Graph #9 (Detail)] that the initial 100 gallons in the hold tank has been used up and the bottling is suspended because the system is waiting for a cook batch from the reactor; the total volume shown on [Graph #9] is a "jagged" line due to this. It should be reiterated here that the

main difference between using (V_{req} = 300 gal.) for the primary and secondary analysis was the affect of the initial tank volumes. Almost all of this affect can be contributed to the initial volume in the hold tank. This is why a small initial hold tank volume (100 gal.) was used for all the analysis of this simulation. By observing the hold tank volume changes on [Graph #9], it can be seen that bottling is suspended one time for (25.14 seconds), then 19 times for (37.14 sec.) each. The total volume of product that could have been produced during these times is [P7 * [(25.14) + 19 (37.14)/60] = 342 gallons. Therefore, an extra 342 gallons initially in the hold tank would have completely eliminated the suspending of bottling during this run; the 2,000 gallons of product would have been produced [(25.14) + 19 (37.14)] = 730 seconds sooner and, therefore, the bottling rate would increase. The bottling rate for the run would change from (237) to (205). This is a (32 bottle/min.) change in the bottling rate. Therefore, the initial hold tank volume is very much a significant variable in effecting the systems performance and should be equally considered when choosing test variables.

The curves of [Graph #10] show the system's response to a "type-1" disaster [i.e.: Pump #1, #2, #3, #4, #5 or the heater stuck off]; any volume in the reactor is dumped and the reactor and ingredients are "cut-off" while the initial 100 gallons in the hold tank is used to produce as many bottles as it can. The

volume in the fill tank has been scaled up on [Graph #10] for "readability".

The curves of [Graph #11] show the system's response to "type-2" disasters [i.e.: Pump #6, #7 or the conveyor stuck off; or "out of bottles"]. The volume in the fill tank remains at the initial one gallon and no volume of product is produced since bottling was suspended immediately. The reactor section, however, will continue an "in-progress" cook batch until it is completed; then the entire system will be shut down. The simulation program was designed this way in anticipation of a possible future version of this program allowing disaster flags being activated at any desired time during the program run. However, at present, the user may only input a disaster at the beginning of the program run. Therefore, since the program is always begun at the "fill state" with no volume in the reactor, the "true present behavior" of the simulation would not look like [Graph #11]. The present response of the simulation to a "type-2" disaster is to immediately shut down the system; all the tank volumes would remain at their initial values.

The curves of [Graph #12] show the system's response to a "type-3" disaster (i.e.: "out of ingredients") the volume of ingredients is detected to be low during the first cook cycle shown; the ingredient tanks are immediately cut off. However, the rest of the system continues until the system essentially runs dry (actually, the hold tank and fill tank will still

contain a small amount of volume due to the values of V_{hmin} and V_{fmin}). The small amount in the reactor due to V_{rmin} is dumped.

(TABLE V-1a) INITIAL CONDITIONS

				PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS		% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
				GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
CONSTANTS:															
TARGET HEAT	T_TrgtHEAT	degrees C		140	140	140	140	140	140	140	140	140	140	140	140
TARGET COOL	T_TrgtCOOL	degrees C		100	100	100	100	100	100	100	100	140	140	140	140
NONFLUCTUATING VARIABLES:															
Volumes:															
REACTOR CAPACITY	Vrcap	gal.	*	110	110	110	110	110	110	110	110	110	110	110	110
HOLD TANK CAPACITY	Vhcap	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
FILL TANK CAPACITY	Vfcap	gal.	*	3	3	3	3	3	3	3	3	3	3	3	3
BOTTLE CAPACITY	Vbcap	ounces	*	12	12	12	12	12	12	12	12	12	12	12	12
DESIRED PRODUCT VOLUME	Vrequest	gal.	*	300	300	300	300	300	300	300	300	2000	2000	2000	2000
INGEDIENT #1 TANK MIN.	V1min	gal.		f(h)	0.17	f(h)	0.17	f(P4)	0.17	0.17	0.17	0.04	0.04	0.04	0.04
INGEDIENT #2 TANK MIN.	V2min	gal.		f(h)	0.33	f(h)	0.33	f(P4)	0.33	0.33	0.33	0.08	0.08	0.08	0.08
INGEDIENT #3 TANK MIN.	V3min	gal.		f(h)	0.50	f(h)	0.50	f(P4)	0.50	0.50	0.50	0.13	0.13	0.13	0.13
REACTOR MIN.	Vrmin	gal.		f(h)	0.33	f(h)	0.33	f(P5)	0.33	0.33	0.33	0.17	0.17	0.17	0.17
HOLD TANK MIN	Vhmin	gal.		f(h)	0.13	f(h)	0.13	0.13	0.13	0.13	0.13	0.02	0.02	0.02	0.02
FILL TANK MIN	Vfmin	gal.		f(h)	f(P7)	f(h,P7)	f(P7)	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
FILL TANK LOW	Vflow	gal.		0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
REACTOR MAX.	Vrmax	gal.		f(h)	98.33	f(h)	98.33	f(P4)	98.33	98.33	98.33	99.58	99.58	99.58	99.58
HOLD TANK MAX.	Vhmax	gal.		f(h)	879.00	f(h)	879.00	f(P4)	879.00	879.00	879.00	879.00	879.00	879.00	879.00
FILL TANK MAX.	Vfmax	gal.		f(h)	2.87	f(h)	2.87	2.87	2.87	2.87	2.87	2.98	2.98	2.98	2.98
Pump Rates:															
PUMP RATE #1	P1	gal./min.		250	250	250	250	f(P4)	250	250	250	125	125	125	125
PUMP RATE #2	P2	gal./min.		500	500	500	500	f(P4)	500	500	500	250	250	250	250
PUMP RATE #3	P3	gal./min.		750	750	750	750	f(P4)	750	750	750	375	375	375	375
PUMP RATE #4	P4	gal./min.	*	1000	1000	1000	1000	TEST	1000	1000	1000	500	500	500	500
PUMP RATE #5	P5	gal./min.	*	500	500	500	500	TEST	500	500	500	500	500	500	500
PUMP RATE #6	P6	gal./min.	*	200	200	200	200	200	200	200	200	50	50	50	50
PUMP RATE #7	P7	gal./min.	*	28.125	TEST	TEST	TEST	28.125	28.125	28.125	28.125	28.125	28.125	28.125	28.125
PUMP RATE #8	P8	gal./min.	*	500	500	500	500	500	500	500	500	500	500	500	500
Heater Variables:															
AMBIENT TEMP.	Ta	degrees C	*	22	22	22	TEST	22	22	22	22	22	22	22	22
HEATER TEMP. SETTING	Te	degrees C	*	200	200	200	200	200	TEST	TEST	200	180	180	180	180
HEATER RATING	HTR	C/min	*	0.50	0.50	0.50	0.50	0.50	0.50	0.50	TEST	0.30	0.30	0.30	0.30
Integration Variable:															
SIMULATION TIME STEP	h	seconds	*	TEST	0.04	TEST	0.04	0.04	0.04	0.04	0.04	0.02	0.02	0.02	0.02

* = a variable which can be "set by user"

TEST = independent variable on graph (ie.: the "TEST" variable)

f(X) = initial value of variable is a function of the value of the "TEST" variable

(TABLE V-1b) INITIAL CONDITIONS

				PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS		% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
				GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
SIMULATED MEASURED VARIABLES:															
Initial Volumes:															
INGEDIENT #1	V1	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	150
INGEDIENT #2	V2	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
INGEDIENT #3	V3	gal.	*	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
REACTOR	Vr	gal.	*	0	0	0	0	0	0	0	0	0	0	0	0
HOLD TANK	Vh	gal.	*	100	100	100	100	100	100	100	100	100	100	100	100
FILL TANK	Vf	gal.	*	1	1	1	1	1	1	1	1	1	1	1	1
Initial Temperature:															
REACTOR	Tr	degrees C	*	22	22	22	f(Ta)	22	22	22	22	22	22	22	22

* = a variable which can be "set by user"
TEST = independent variable on graph (ie.: the "TEST" variable)
f(X) = initial value of variable is a function of the value of the "TEST" variable

(TABLE V-1c) INITIAL CONDITIONS

VARIABLE	SYMBOL	UNITS		PRIMARY ANALYSIS: % WASTE and BOTTLING RATE vs h	PRIMARY ANALYSIS: % WASTE and BOTTLING RATE vs P7	PRIMARY ANALYSIS: BOTTLING RATE vs h@ CRITICAL RATIO	SECONDARY ANALYSIS: BOTTLING RATE vs Ta	SECONDARY ANALYSIS: BOTTLING RATE vs P4,P5	SECONDARY ANALYSIS: Te vs time	SECONDARY ANALYSIS: BOTTLING RATE vs Te	SECONDARY ANALYSIS: BOTTLING RATE vs HTR	FINAL ANALYSIS: VOLUME CHANGES WITH NO DISASTER	FINAL ANALYSIS: VOLUME CHANGES WITH TYPE-1 DISASTER	FINAL ANALYSIS: VOLUME CHANGES WITH TYPE-2 DISASTER	FINAL ANALYSIS: VOLUME CHANGES WITH TYPE-3 DISASTER
				GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
FLAGS:															
Internal:															
FLAG #1	Ui	0/1		0	0	0	0	0	0	0	0	0	0	0	0
FLAG #2	Ur	0/1		0	0	0	0	0	0	0	0	0	0	0	0
FLAG #3	Us	0/1		0	0	0	0	0	0	0	0	0	0	0	0
FLAG #4	Um	0/1		0	0	0	0	0	0	0	0	0	0	0	0
External:															
DISASTER FLAG #1 (P1)	#1	0/1	*	0	0	0	0	0	0	0	0	0	1	0	0
DISASTER FLAG #2 (P2)	#2	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #3 (P3)	#3	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #4 (P4)	#4	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #5 (P5)	#5	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #6 (P6)	#6	0/1	*	0	0	0	0	0	0	0	0	0	0	1	0
DISASTER FLAG #7 (P7)	#7	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #8 (HTER)	#8	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #9 (CONV)	#9	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0
DISASTER FLAG #10(BTLS)	#10	0/1	*	0	0	0	0	0	0	0	0	0	0	0	0

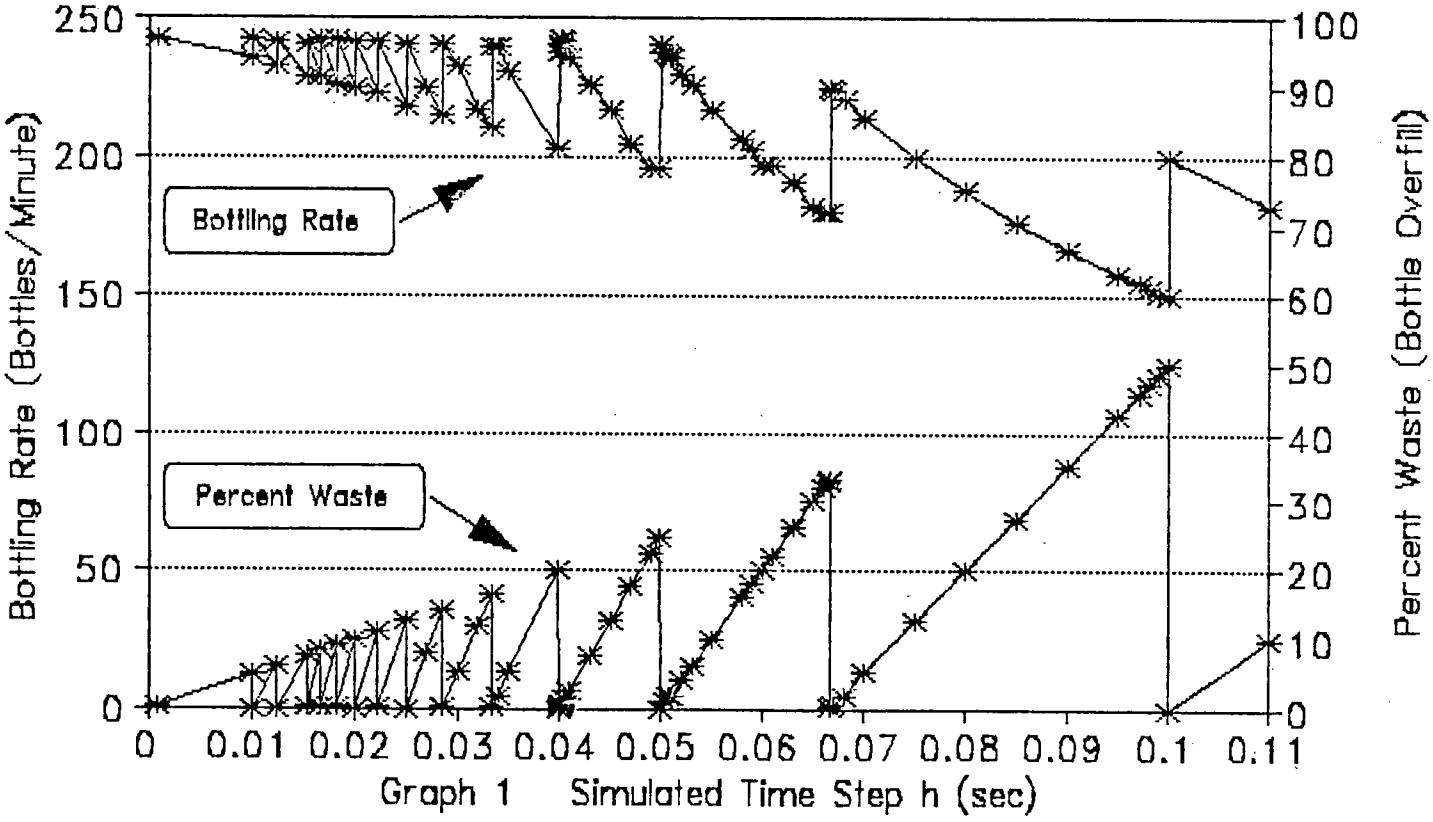
* = a variable which can be "set by user"

(TABLE V-1d) INITIAL CONDITIONS

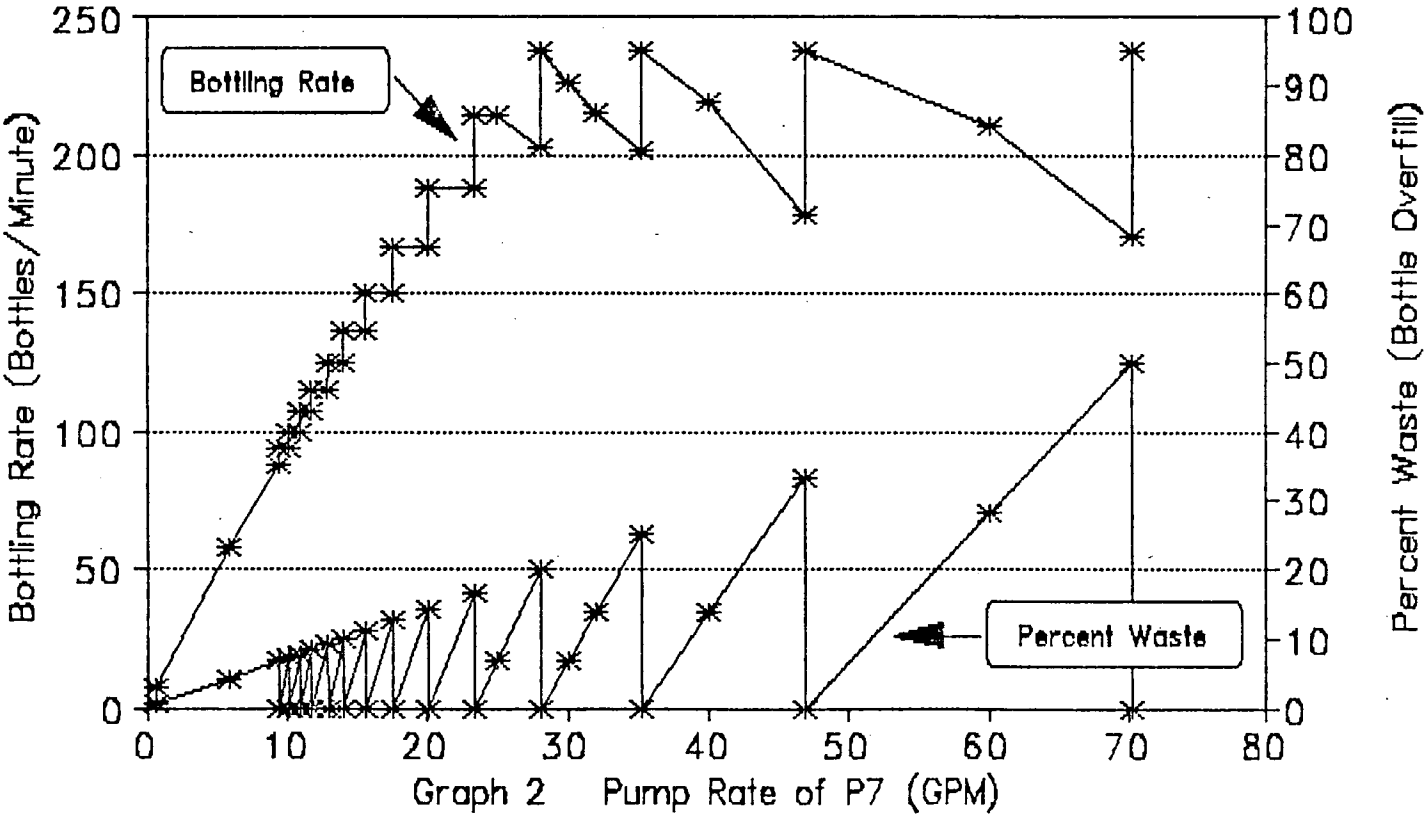
			PRIMARY ANALYSIS			SECONDARY ANALYSIS					FINAL ANALYSIS			
VARIABLE	SYMBOL	UNITS	% WASTE and BOTTLING RATE vs h	% WASTE and BOTTLING RATE vs P7	BOTTLING RATE vs h@ CRITICAL RATIO	BOTTLING RATE vs Ta	BOTTLING RATE vs P4,P5	Te vs time	BOTTLING RATE vs Te	BOTTLING RATE vs HTR	VOLUME CHANGES WITH NO DISASTER	VOLUME CHANGES WITH TYPE-1 DISASTER	VOLUME CHANGES WITH TYPE-2 DISASTER	VOLUME CHANGES WITH TYPE-3 DISASTER
			GRAPH#1	GRAPH#2	GRAPH#3	GRAPH#4	GRAPH#5	GRAPH#6	GRAPH#7	GRAPH#8	GRAPH#9	GRAPH#10	GRAPH#11	GRAPH#12
CONTROLS:														
ACTIVATE PUMPS	U1 to U8	0/1	0	0	0	0	0	0	0	0	0	0	0	0
ACTIVATE CONVEYOR	Uc	0/1	0	0	0	0	0	0	0	0	0	0	0	0
ACTIVATE HEATER	Uh	0/1	0	0	0	0	0	0	0	0	0	0	0	0
TIME:														
PROCESS TIME	CUMTIME	seconds	0	0	0	0	0	0	0	0	0	0	0	0
PROGRAM EXECUTION TIME	REALTIME	seconds	0	0	0	0	0	0	0	0	0	0	0	0
PERFORMANCE CRITERIA:														
BOTTLING RATE	BOTLNGRATE	#/min	0	0	0	0	0	0	0	0	0	0	0	0
PERCENTAGE WASTE	%OVERFILL	percent	0	0	0	0	0	0	0	0	0	0	0	0
STATE VARIABLES:														
"FILL"	"FILL"	none	0	0	0	0	0	0	0	0	0	0	0	0
"HEAT"	"HEAT"	none	0	0	0	0	0	0	0	0	0	0	0	0
"COOL"	"COOL"	none	0	0	0	0	0	0	0	0	0	0	0	0
"DRAIN"	"DRAIN"	none	0	0	0	0	0	0	0	0	0	0	0	0
OTHER INTERNAL VARIABLES:														
PRINT INDEX	PRINTINDEX	none	20	20	20	20	20	20	20	20	200	20	20	20
Misc. Internal Variable	-	none	-	-	-	-	-	-	-	-	-	-	-	-

* = a variable which can be "set by user"

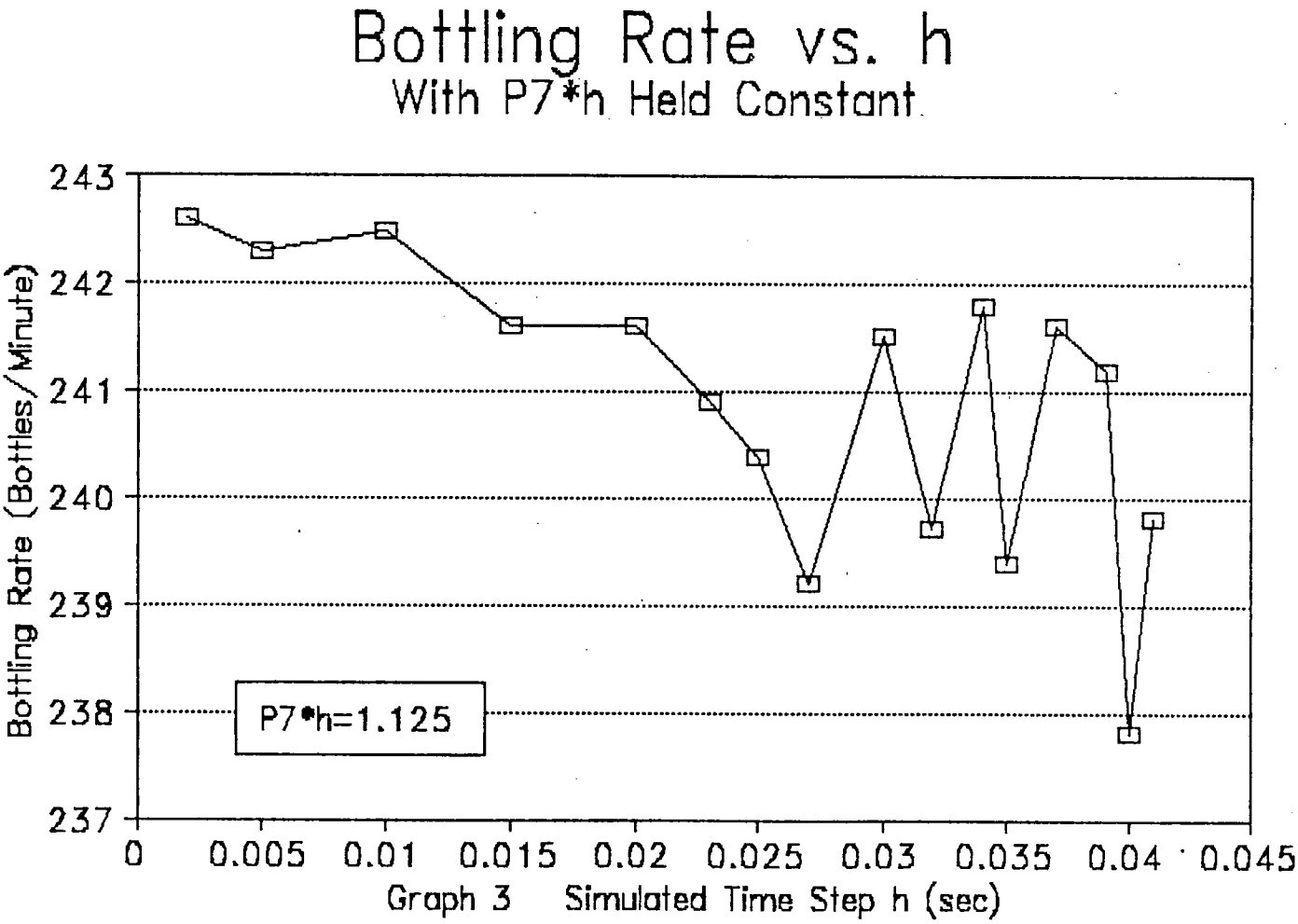
Percent Waste & Bottling Rate vs. h



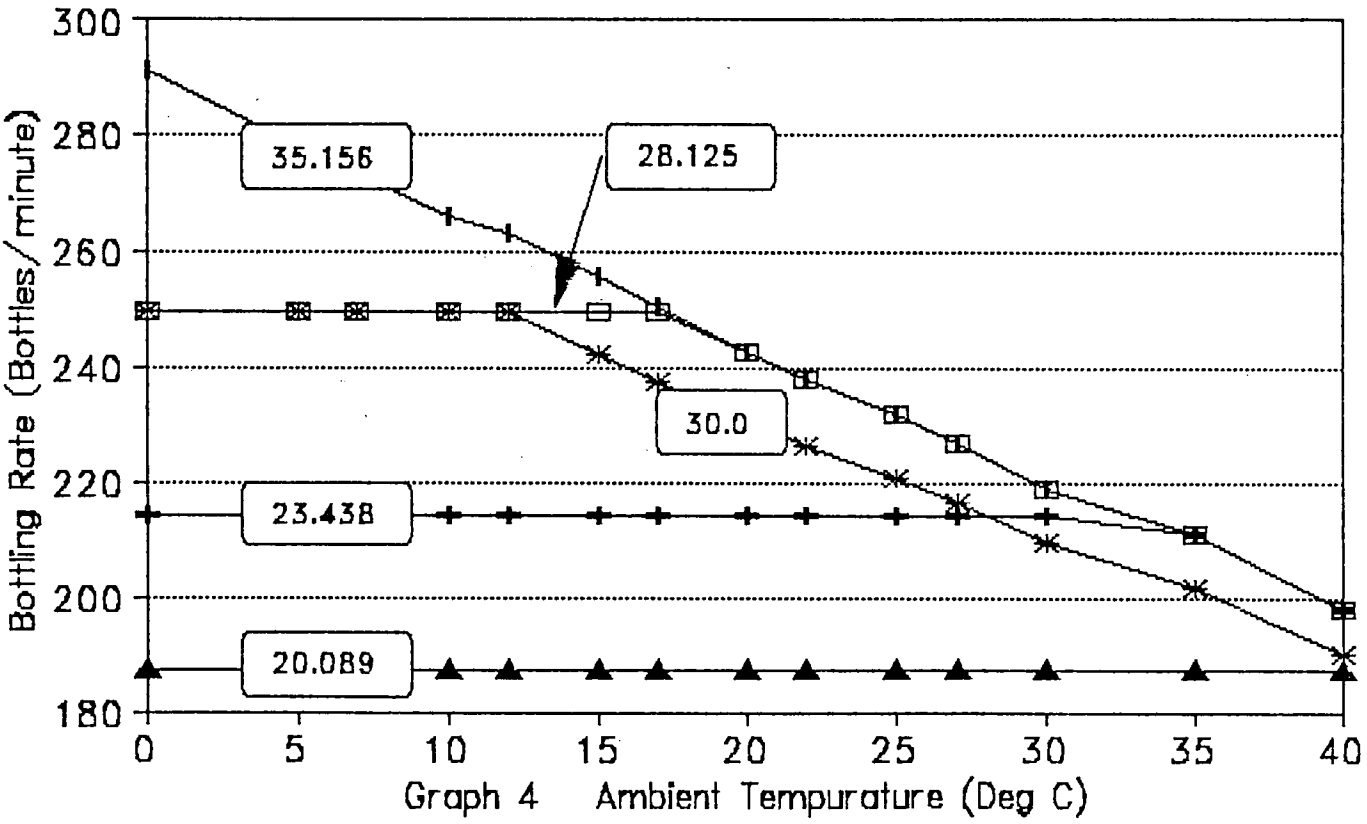
Percent Waste & Bottling Rate vs. P7



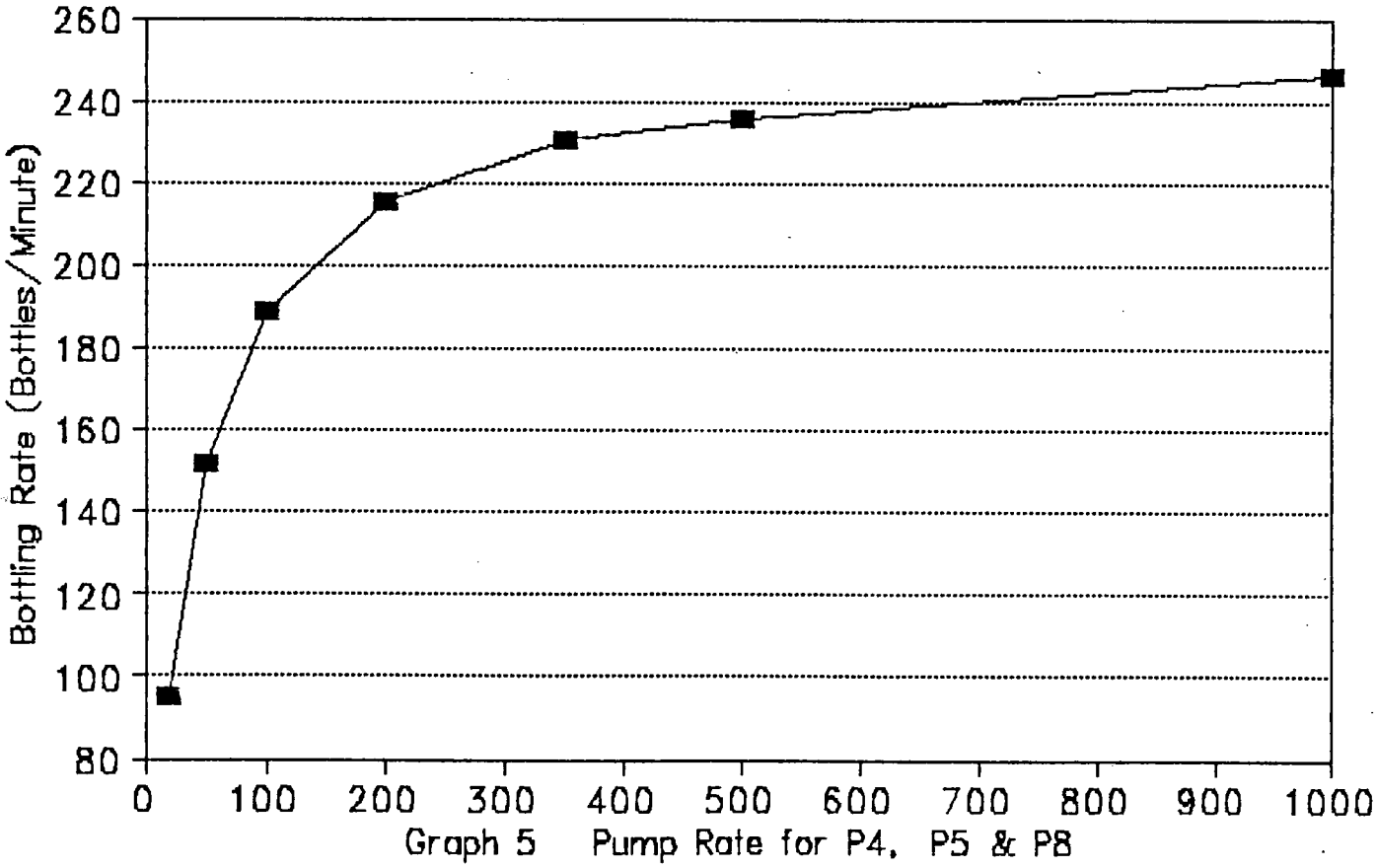
Graph 2


Bottling Rate vs. h
With P7*h Held Constant
Bottling Rate (Bottles/Minute)
243
242
241
240
239
238
237
P7*h=1.125
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
Graph 3 Simulated Time Step h (sec)

Bottling Rate vs. Ambient Temperature



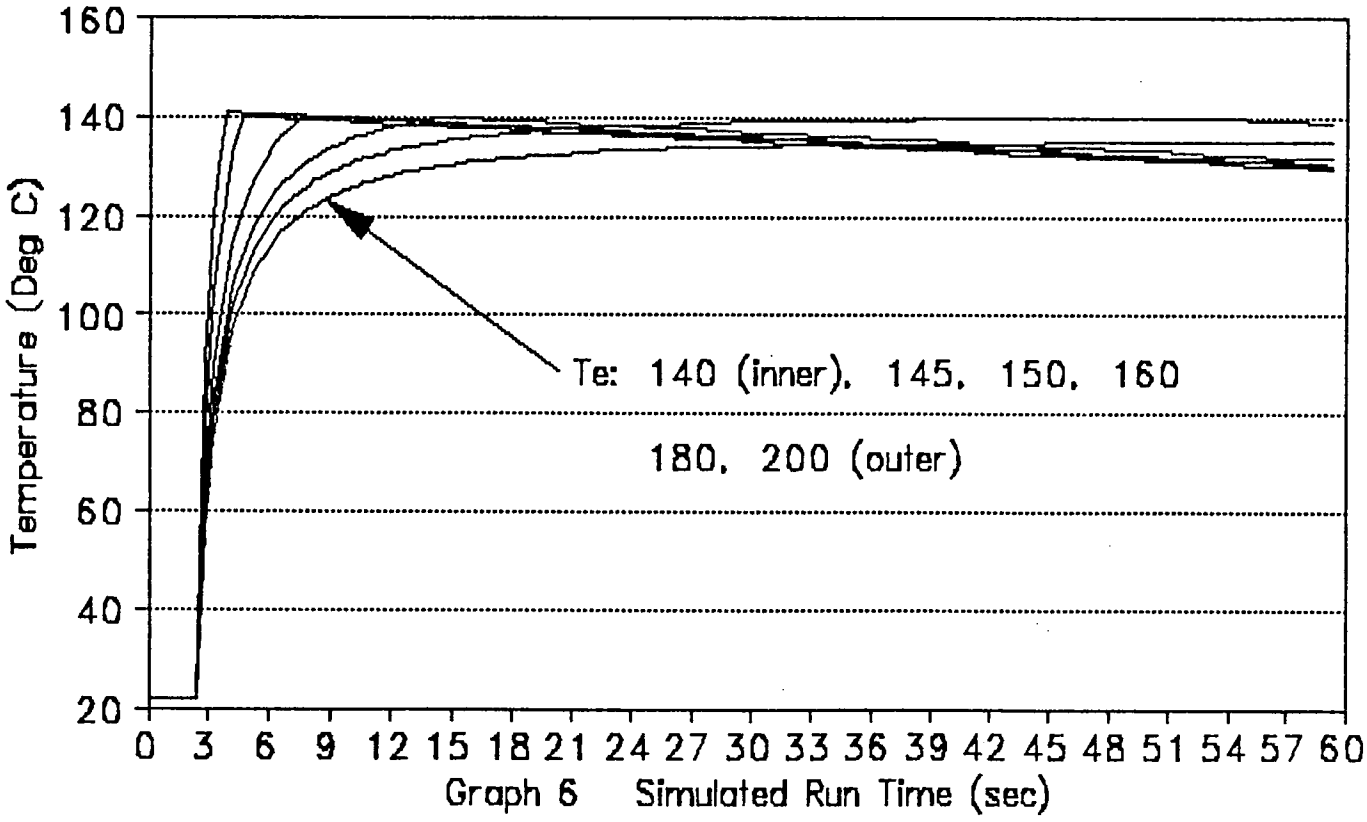
Bottling Rate vs. Max Pump Rates



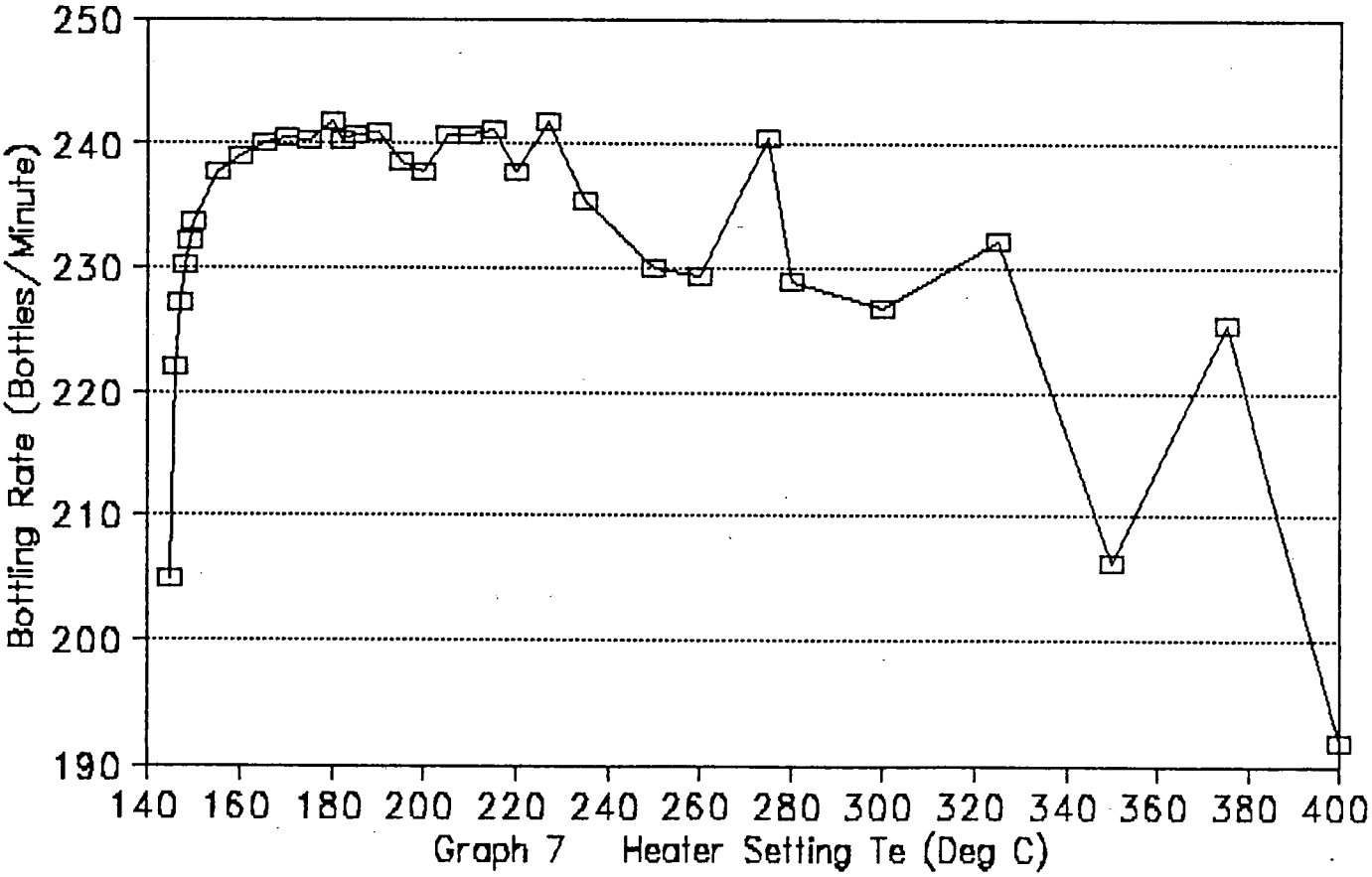
Graph 5 Pump Rate for P4, P5 & P8

Reactor Temperature vs. Time

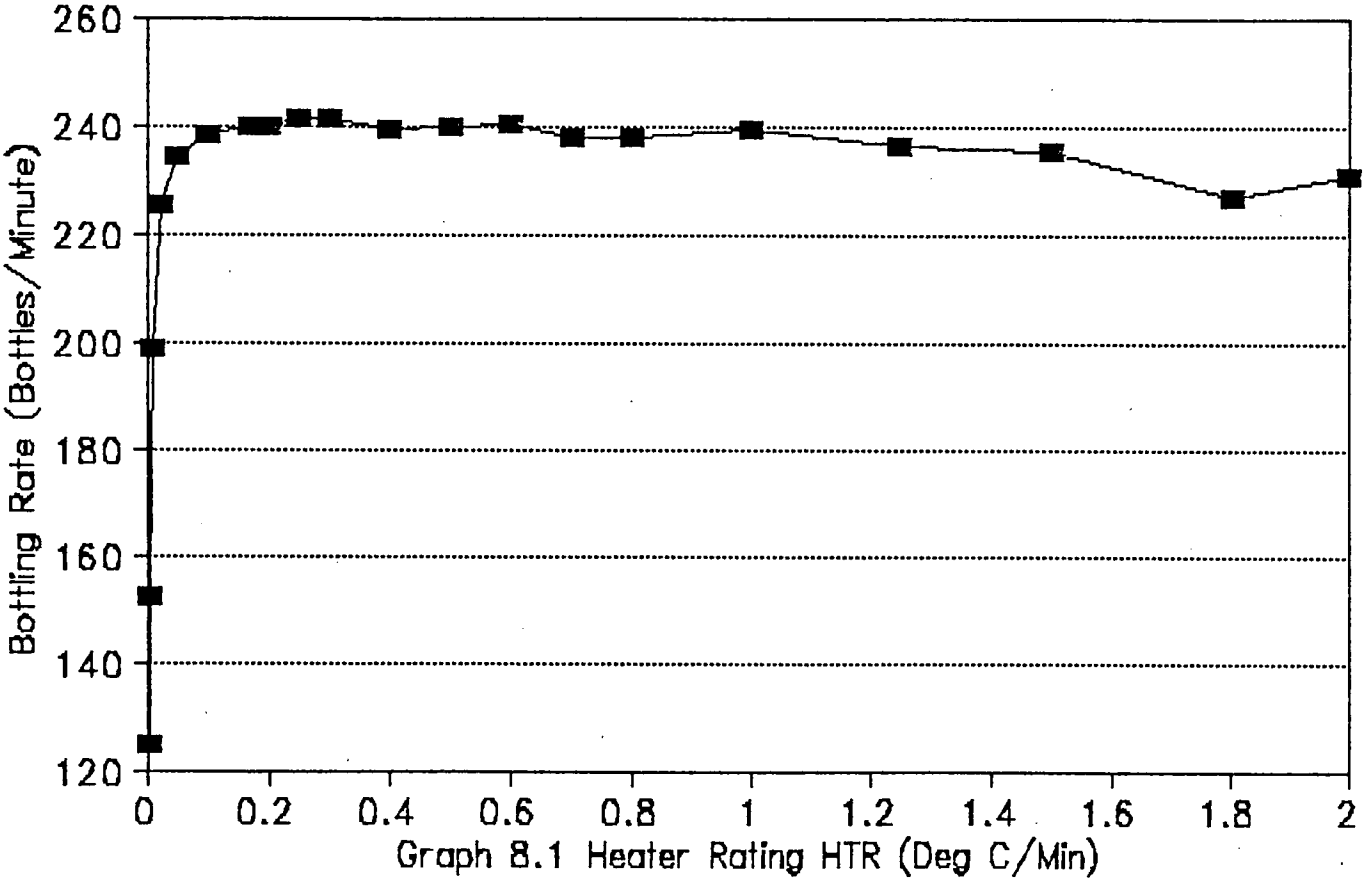
For Various Values of Te



Bottling Rate vs. Te

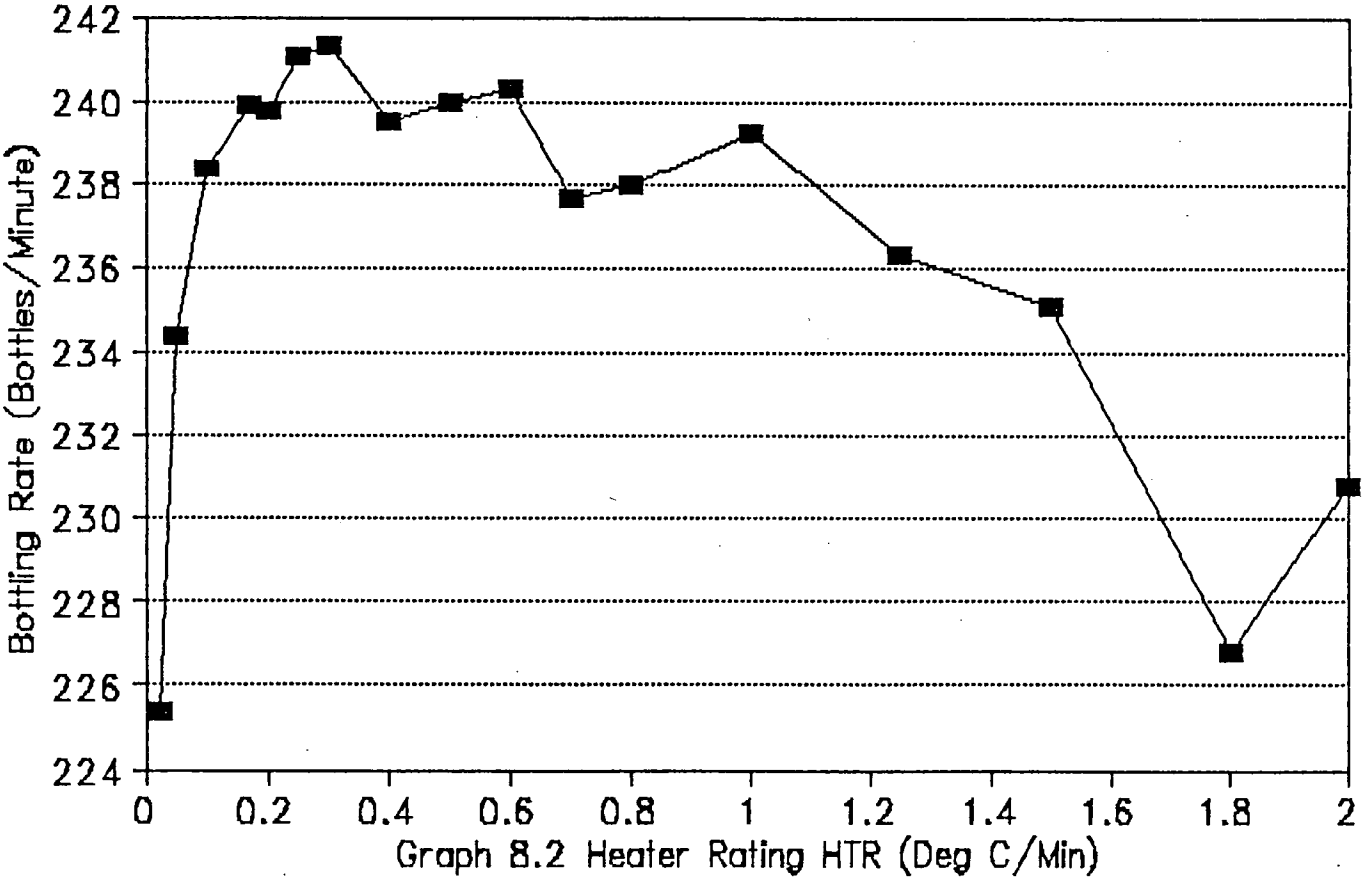


Bottling Rate vs. HTR Rating

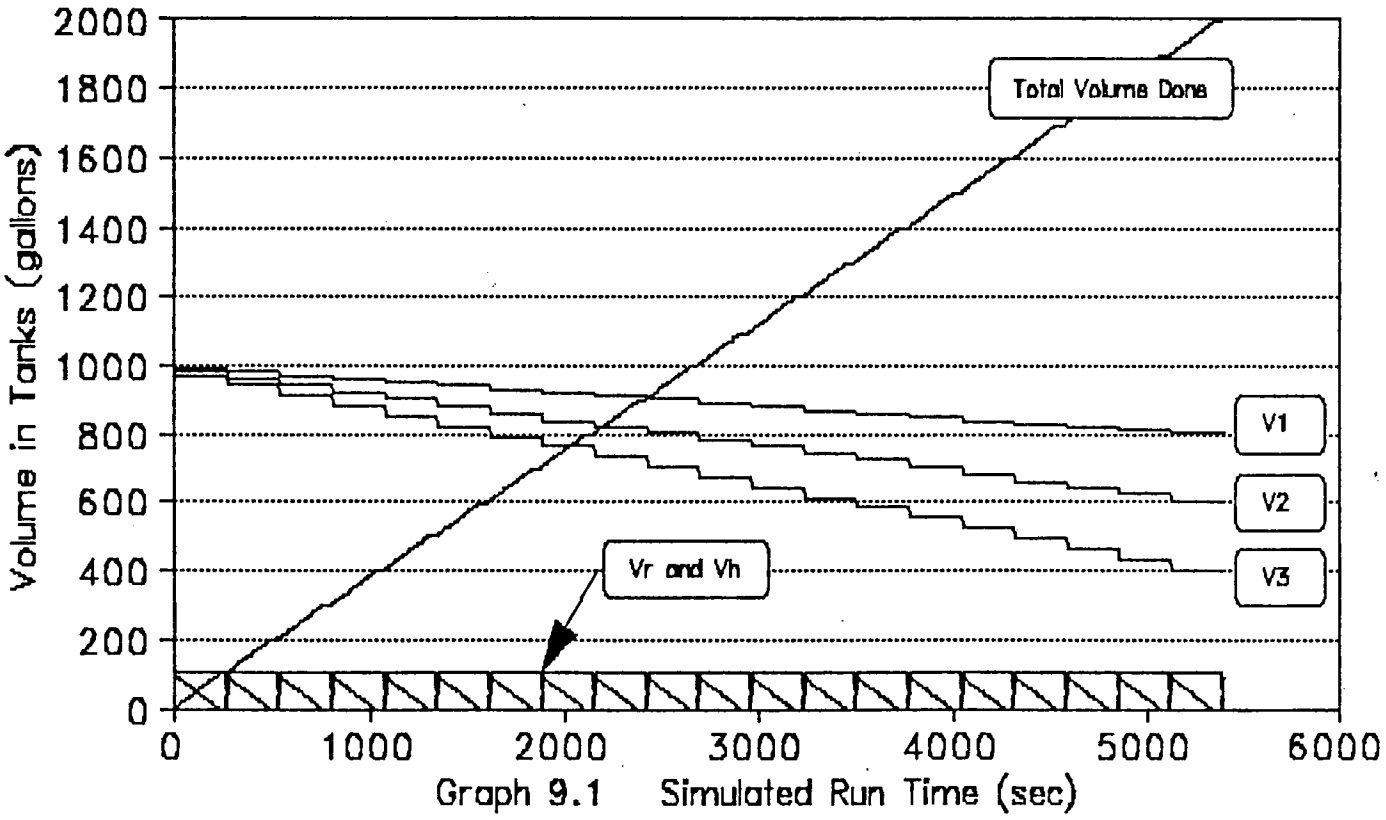


Bottling Rate vs. HTR Rating

Detail with Expanded Y Scale



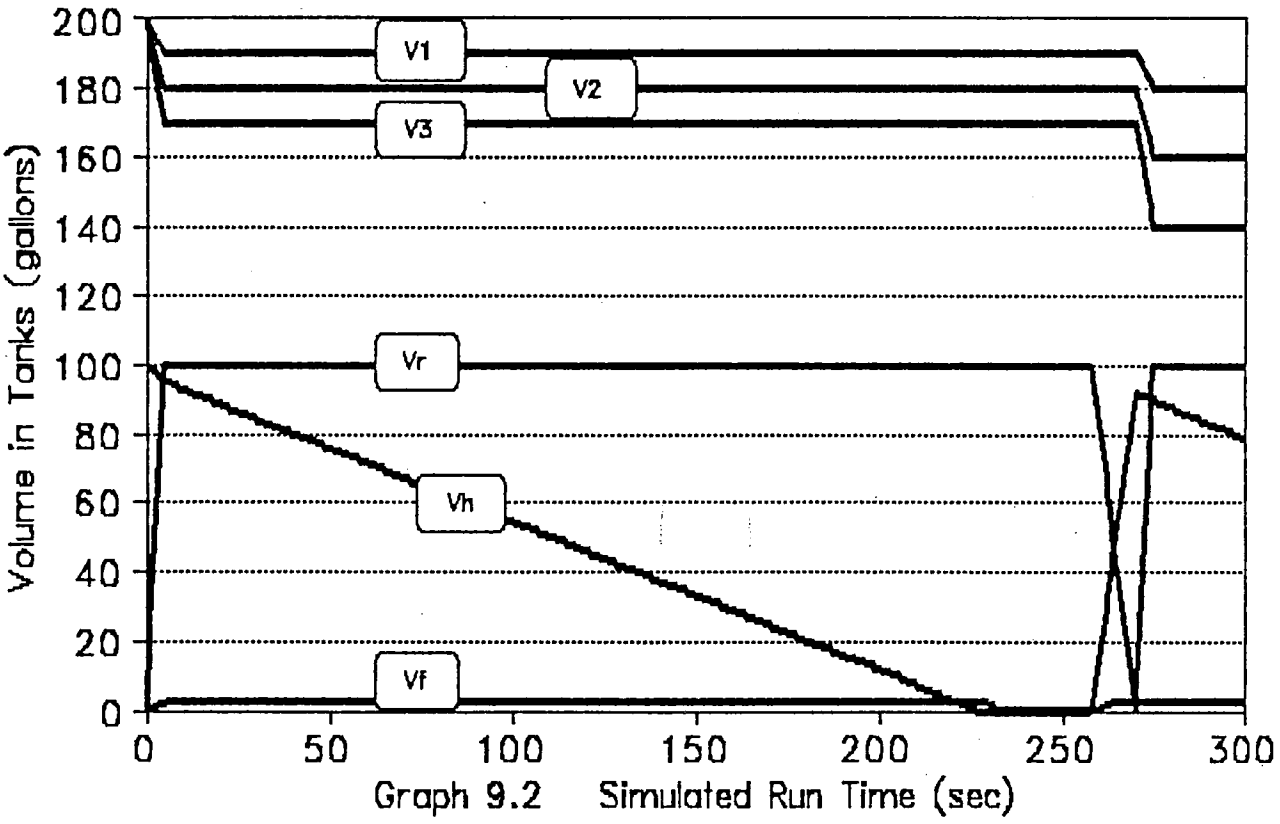
Volume in Tanks vs. Run Time



Graph 9.1

Volume in Tanks vs. Run Time

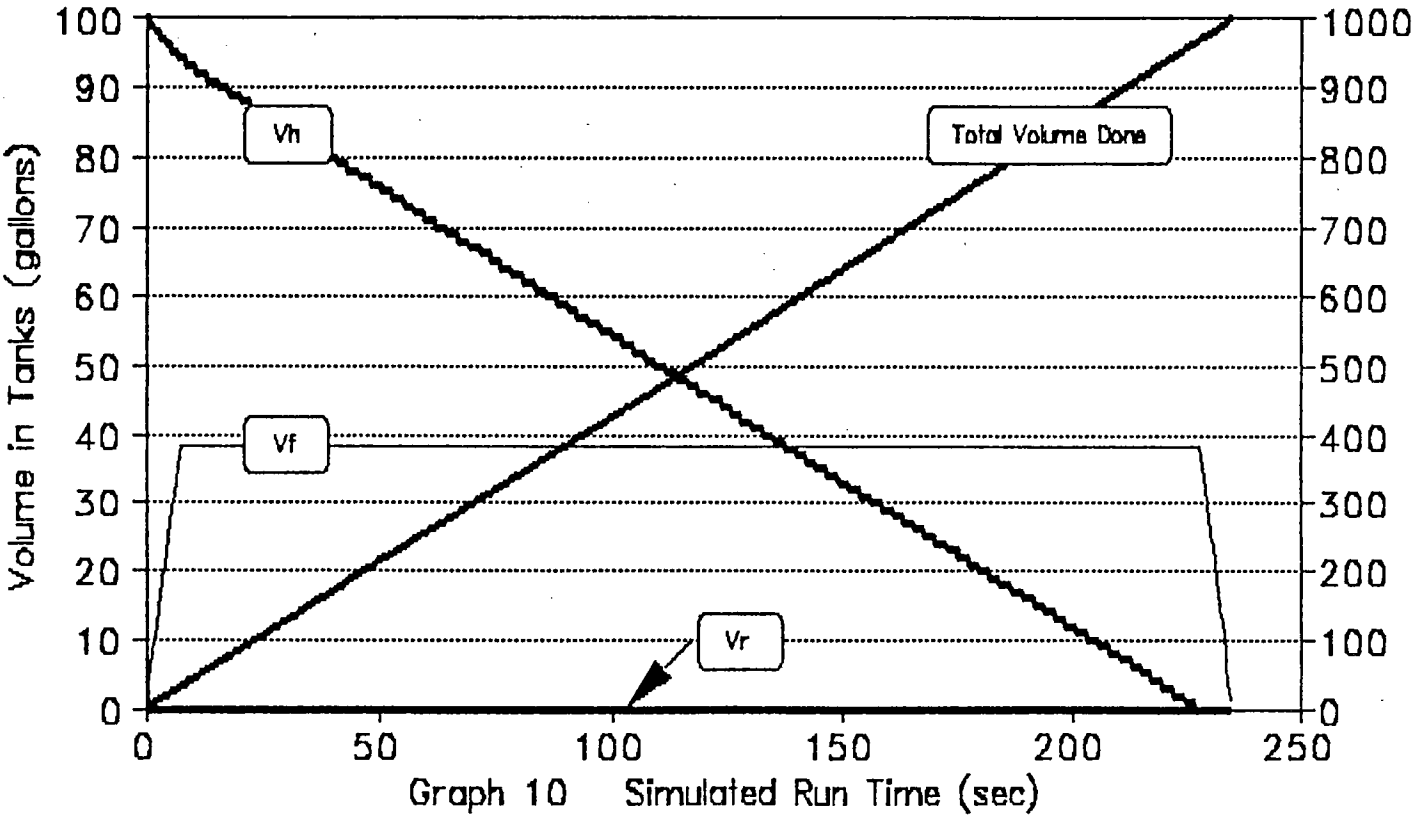
One Cook Cycle



Graph 9.2

Volume in Tanks vs. Run Time

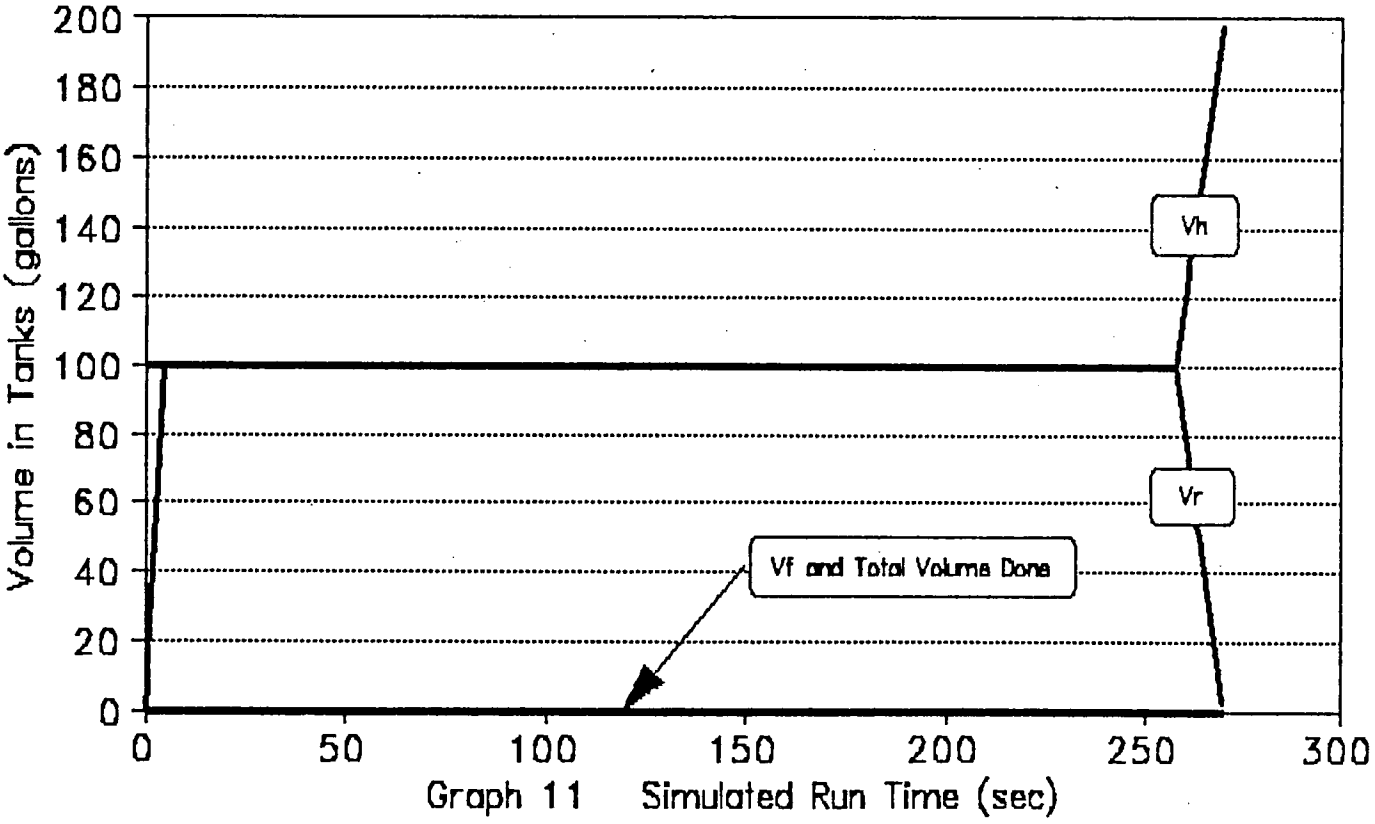
P1 Stuck at Off



Graph 10

Volume in Tanks vs. Run Time

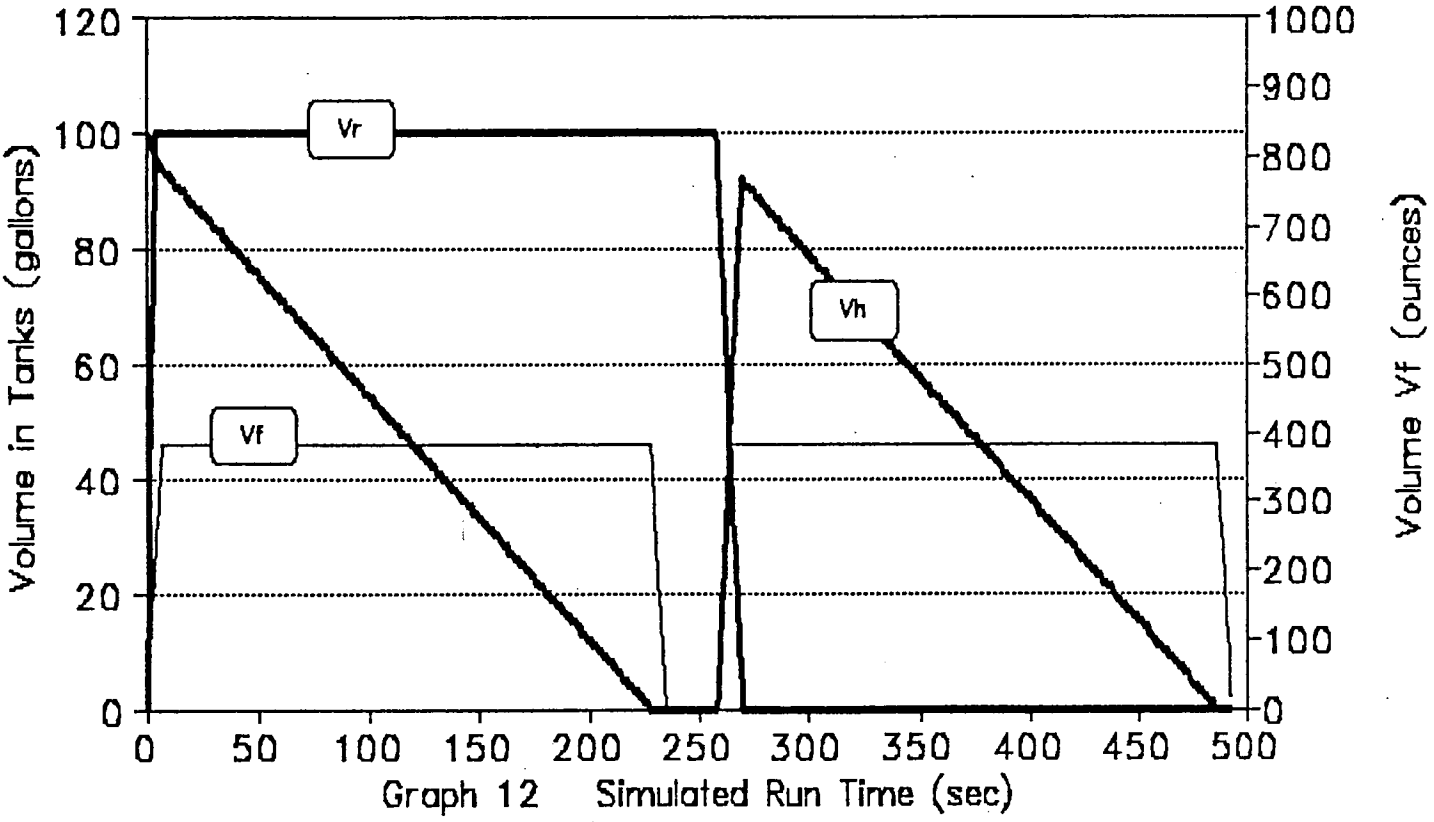
P6 Stuck at Off



Graph 11

Volume in Tanks vs. Run Time

Low V1 Volume After One Cook Cycle



Graph 12

VI CONCLUSIONS

The modeling of a physical system can be accomplished by many different approaches. The approach taken for the simulation program designed for this report was as follows:

(1) Conceptually partition system into a upper and lower section.

(2) Define the behavior and limits of each tank individually within each partition.

(3) Consider the interdependency between tanks of each partition.

(4) Consider the interdependency between partitions.

(5) Test the model created from steps (1) through (4).

(6) Optimize system parameters based on the results of the test.

VI.A) ALTERNATIVE MODELS

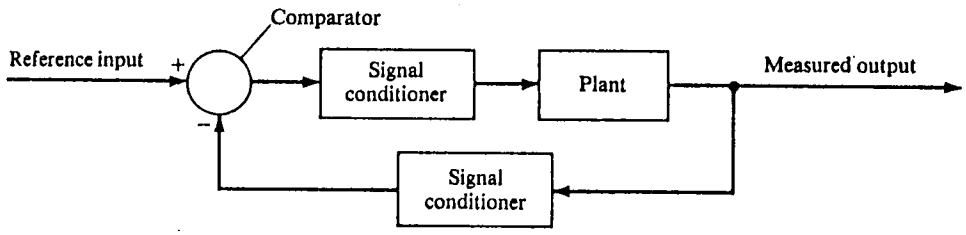
The method above worked well for the physical process given because of the simplicity of the system; the system did not require a complicated mathematical model. However, the following two methods of modeling were considered:

(i) A feedback control system
(ii) An analog model

VI.A.1) A FEEDBACK CONTROL SYSTEM

The feedback control model consists of mathematically connecting the controls sent to a subsystem with the feedback sent back from

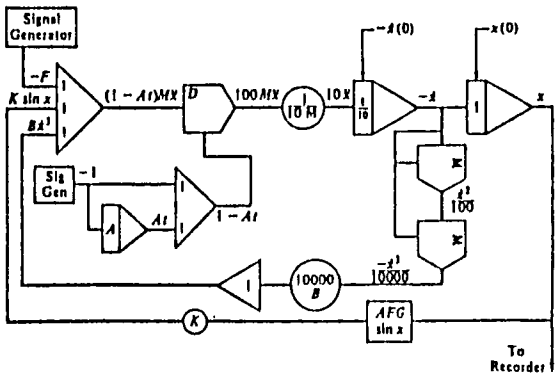
that sub-system in such a way that when all the different subsystems' control signals and feedbacks are considered collectively, the system model can be simplified. The following diagram shows the basic concept of feedback control[1]:



The "plant" in this diagram could be the physical system. The feedback loop with its "signal conditioner" could represent the simulating of measured variables. The "signal conditioner" going into the "plant" could represent conditional program statements which determine which control signals to send to the physical system ("plant"). For a physical system such as a bottling plant, a diagram such as this one could be made for each tank. All of the resulting diagrams could be connected and the resulting block diagram could then be simplified. All blocks would contain a transfer function representing the relation between its input and output. Blocks with differential equations in them could be reduced into simple algeraic relations (using Laplace transforms or Z-transforms). Once all blocks were in algeriac terms, the model could be simplified by the block diagram transformations shown in (Figure VI-1)$_2$.

VI.A.2) AN ANALOG MODEL

An analog model consists of modeling a system as an analog circuit using analog integrators, multipliers, dividers and summers. A differential equation in a system could be modeled as follows:[3]



This figure models the equation:

$$(1 - At)\ M\ \ddot{X} + B\ (X)^3 + k\sin\dot{x} = F$$

The symbols used to make this model are shown in [Figure VI-2]. For a system such as the bottling plant the initial consideration of this type of model did not aid in simplifying the task of creating a simulation program. However, the advantages of hybrid computation was made more apparent by this exercise. If a hybrid computer was ever used to simulate the control of a bottling plant, the simulated measured variables could be calculated by using the analog circuits to yield the true solutions to the time dependent differential equations which describe the physical system. The digital part of the computer could then create simulated control signals by using conditional program statements which test the values created by the analog circuits. This type of computation would eliminate the accuracy problems inherent in numerical methods of integration which approximate with relatively large degrees of error.

VI.B) NUMERICAL METHODS

Numerical methods are methods of solving differential equations which produce approximate solutions at particular points. The accuracy of the numerical method used can have a very significant affect on the ability of a simulation to represent the true behavior of a system. The Euler's method is probably the simplest and least accurate numerical method. This method is based on approximating successive values along a curve by using the slope of the curve at a given point to approximate the value of the next point on the curve. This is explained in detail in [Figure VI-3]$_4$. The Euler's method is, therefore, very accurate for linear relations. But, becomes very inaccurate for functions with curves that curve sharply. Luckily most of the differential equations of the bottling plant simulation problem were not represented by curves which curved sharply. Only the reactor heat balance equation behaved very much non-linearly, this equation has a curve which has already been shown in [Graph #6] in [Sections IV and V]. The rise of the curve (heating) is mostly contributed to the (X^2) part of the equation. The decent of the curve (cooling) is solely contributed to the (-1/10 X) part of the equation. The (-1/10 X) part of the curve represented over 95% of the time involved in heating and cooling. The remaining 5% contributed to the (X^2) part is the most susceptible part of the simulation program because of its

severe nonlinearity. However, the affect of the time delay due to heating had such a small affect on the system's performance that only an extreme error in Euler's method here would have an adverse affect on the system. However, if the cooling time was increased due to an inaccurate approximation of the value of (T_r) the performance of the system would definitely be effected. This will be discussed below.

The inaccuracy of Euler's method can be seen in [Figure VI-4]$_5$. The example used here uses a slope of ($d_Y/d_X = Y^2 + 1$). This is somewhat similar to the heat balance equation (dT_r/dt) (except that we have a (+Tr) instead of a (-1 term). If we let (X = t) and (Y = Tr), it can be seen from the figure that the approximate value of (Y = Tr) begins to diverge from the true solution, for all (h) values, as (X = t) is increased from 0 to 1. This divergence is more exaggerated for larger (h) values. By the time (X= t) is 1 the errors is (0.67% for h = 0.005), (1.33% for h = 0.01), (6.21% for h = 0.05), and (11.52% for h = 0.1). Since the (X^2) term is the major determining factor for error in both this example and the heat balance equation, a general assertion can be made about the effect of this type of error on the bottling plant simulation program; based on interpolation in the example shown above, an approximately (2%) to (6%) error in calculating the reactor temperature (T_r) must be expected for the (h = 0.02 to 0.04) values used. This is however for only one second. The heating

part of the cook cycles typically took 4 seconds. If the error is assumed to grow approximately linearly, the error in predicting T_r could be from (8% to 24%). This would probably not significantly affect the bottling rate due to the increase in the relatively fast heating time, but, if the cooling is not begun until a later time the extra length of cooling would significantly effect the bottling rate. If the heater heats up to 24% above the target heat ($140^{o}C$) to ($173.6^{o}C$), the reactor would have to cool ($73.6^{o}C$) instead of the normal ($40.0^{o}C$). This corresponds to an 84% increase in cooling time. The normal cool time of approximately 250 seconds would now be 460 seconds. Therefore, instead of a (37 second) suspension of bottling between each reactor batch [see Section V-C) Final Analysis] there would be a [(460-250) + 37) = 247 second delay. This would cut the bottling rate in half!

There are many other numerical methods which have much less inherent error than Euler's method. Two of them will be shown here:

(i) The three term Taylor series method
(ii) The fourth order Runge-Kutta method[6][7]

The three term Taylor series method approximates values for (Y) according to:

$$Y_{n+1} = Y_n + h\left(\frac{dY}{dX}\right) + \frac{h^2}{2}\left(\frac{d^2Y}{dX^2}\right)$$

Where (d_Y^2/d_X^2) is obtained by evaluating a given first-order equation with respect to (X) and evaluating at ($X = X_n$). In [Figure VI-5], approximated values of (Y) for ($d_Y/d_X = Y^2 + 1$) are compared with true solutions.

The fourth order Runge-Kutta method uses the form:

$$Y_{n+1} = Y_n + 1/6\ (K_1 + 2\ K_2 + 2\ K_3 + K_4)$$

Where $K_1 = hf\ (X_n, Y_n)$
$K_2 = hf\ (X_n + 1/2\ h, Y_n + 1/2\ K_1)$
$K_3 = hf\ (X_n + 1/2\ h, Y_n + 1/2\ K_2)$
$K_4 = hf\ (X_n + h, Y_n + K_3)$

[Figure VI-6] compares approximated values for $f\ (X_n, Y_n) = (d_Y/d_X = Y^2 + 1)$ with true solutions. A comparison of the numerical methods discussed for ($h = 0.1$) reveals the following at ($X = 1$):

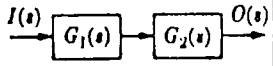
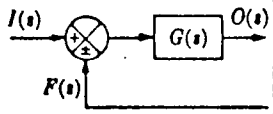
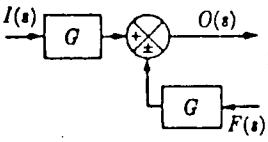
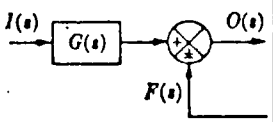
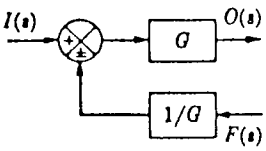
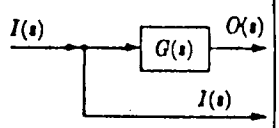
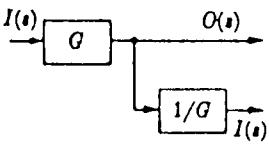
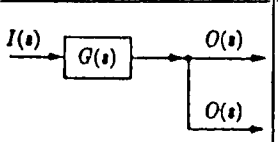
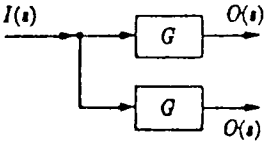
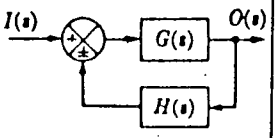
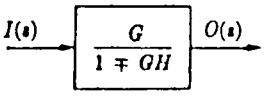
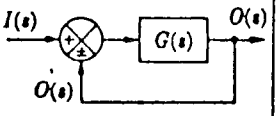
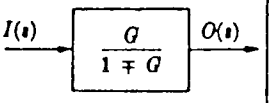
(i) The Euler's method had a (11.52%) error
(ii) The three term Taylor series method had a (1.60%) error
(iii) The fourth order Runge-Kutta method had a (0.00008%) error

Clearly, Euler's method is relatively inaccurate for this type of curve, therefore, there is a significant amount of uncertainty in evaluating the systems performance due to the possible error in accurately predicting values of (T_r). (Even at $h = 0.02$).

VI-C) SIMULATIONS FOR REAL-TIME DECISION SUPPORT

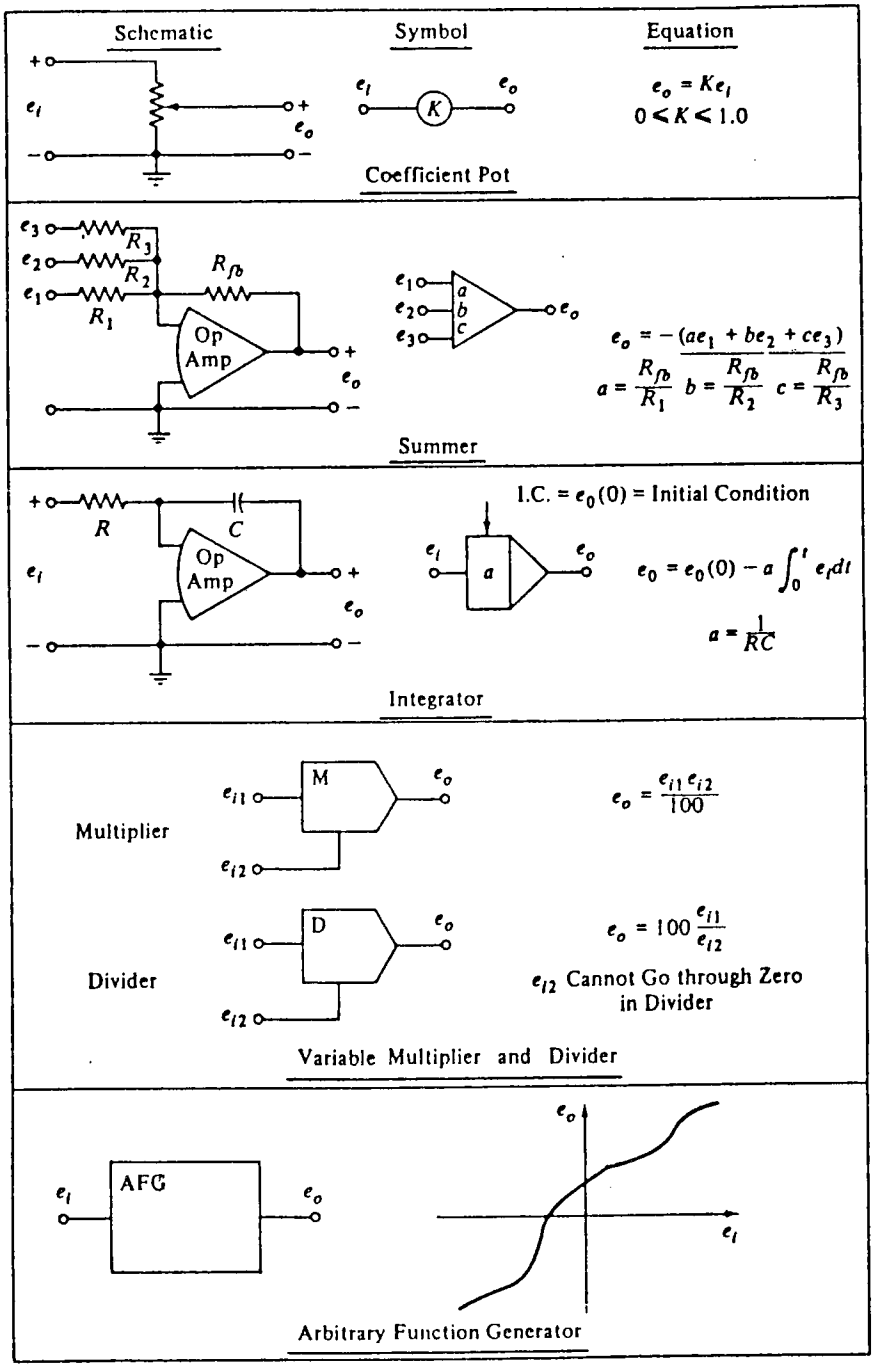
Despite the error inherent in the Euler's method of integration used for this simulation program, a great deal was learned about the variables tested. The observed significance of their relative magnitudes and their interrelation with the performance criteria and other variables was significant despite Euler's method. The design of the normal and disaster control program flow allowed insight into how to design a program to actually control a real-time physical process. The use of this process control simulation program would definitely be a useful tool in supporting the design decisions involved in creating the controls and sizing the equipment for an actual real-time controlled physical process. Even after the physical system is built, the simulation program is a valuable tool since the speed of the simulation run time is much faster than any actual process run time could be. This would allow much quicker decisions on optimal values of variables such as pump rates and heater temperature settings if the physical mechanisms allowed varying speeds and settings.

SOME IMPORTANT BLOCK-DIAGRAM TRANSFORMATIONS

Original diagram	Equivalent diagram	Remarks
		1. Combining two blocks in cascade, $I \cdot G_1 \cdot G_2 = I(G_1 \cdot G_2)$
		2. Moving a summing point behind a block, $(I \pm F)G = (I \cdot G) \pm (F \cdot G)$
		3. Moving a summing point ahead of a block, $(I \cdot G) \pm F = \left(I \pm \frac{1}{G} \cdot F\right) G$
		4. Moving a pickoff point behind a block.
		5. Moving a pickoff point ahead of a block.
		6. Eliminating a feedback loop
		7. Special case for Rule 6 above ($H = 1$).

(Figure VI-1)

ANALOG COMPUTER ELEMENTS



(Figure VI-2)

The value of $x(t)$ at $t = t_0$ is specified by the initial condition (I):

$$x(t_0) = x_0.$$

To estimate the value of $x(t)$ at $t = t_1$, we note that on a small interval the derivative $x'(t)$ will be nearly constant. The derivative at $t = t_0$ is given by the o.d.e. (D):

$$x'(t_0) = f(t_0, x_0).$$

If the derivative were actually constant for $t_0 \le t \le t_1 = t_0 + h$, then the value of $x(t)$ at $t = t_1$ would be given by

$$(E_1) \qquad x_1 = x_0 + hf(t_0, x_0).$$

We adopt this as our estimate for the value of $x(t_1)$ and use it to predict $x(t_2)$. Note that x_1 is the value we would get if we moved from (t_0, x_0) along the tangent line to the curve $x = x(t)$ instead of moving along the curve itself (see Figure 6.1).

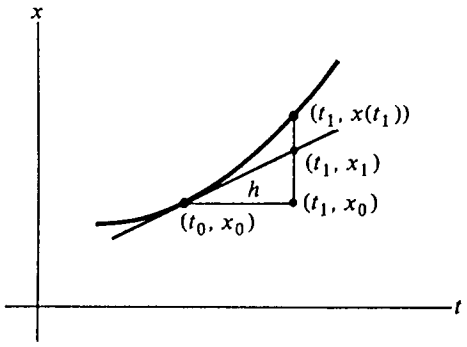


FIGURE 6.1

Similar reasoning over the interval $t_1 \le t \le t_2 = t_1 + h$ leads to an estimate for the value of $x(t)$ at $t = t_2$:

$$(E_2) \qquad x_2 = x_1 + hf(t_1, x_1).$$

Indeed, continuing in this way, we obtain estimates for the values of $x(t)$ at the successive checkpoints $t = t_i$ via the recurrence relation

$$(E_i) \qquad x_i = x_{i-1} + hf(t_{i-1}, x_{i-1}).$$

The final estimate x_n corresponds to $t = t_n = \tau$, and this is our approximation for $x(\tau)$.

Recurrence relations like (E_i) are easily handled on computers or hand calculators. When programming such devices to implement Euler's method for a specific initial value problem, it is useful to first map out strategy, as described in the example below.

Method: EULER'S METHOD					
Problem: $y' = y^2 + 1;\ y(0) = 0$					
x_n	y_n				True solution $Y(x) = \tan x$
	$h = 0.1$	$h = 0.05$	$h = 0.01$	$h = 0.005$	
0.0	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.1000	0.1001	0.1003	0.1003	0.1003
0.2	0.2010	0.2018	0.2025	0.2026	0.2027
0.3	0.3050	0.3070	0.3088	0.3091	0.3093
0.4	0.4143	0.4183	0.4218	0.4223	0.4228
0.5	0.5315	0.5384	0.5446	0.5455	0.5463
0.6	0.6598	0.6711	0.6814	0.6827	0.6841
0.7	0.8033	0.8212	0.8378	0.8400	0.8423
0.8	0.9678	0.9959	1.0223	1.0260	1.0296
0.9	1.1615	1.2055	1.2482	1.2541	1.2602
1.0	1.3964	1.4663	1.5370	1.5470	1.5574

(Figure VI-4)

Method: THREE-TERM TAYLOR SERIES METHOD				
Problem: $y' = y^2 + 1;\ y(0) = 0$				
x_n	y_n			True solution $Y(x) = \tan x$
	$h = 0.1$	$h = 0.05$	$h = 0.01$	
0.0	0.0000000	0.0000000	0.0000000	0.0000000
0.1	0.1000000	0.1002503	0.1003313	0.1003347
0.2	0.2020100	0.2025322	0.2027028	0.2027100
0.3	0.3081933	0.3090436	0.3093243	0.3093363
0.4	0.4210663	0.4223474	0.4227749	0.4227932
0.5	0.5437532	0.5456384	0.5462751	0.5463025
0.6	0.6803652	0.6831445	0.6840955	0.6841368
0.7	0.8366079	0.8407771	0.8422249	0.8422884
0.8	1.0208208	1.0272615	1.0295378	1.0296386
0.9	1.2458742	1.2562453	1.2599903	1.2601582
1.0	1.5328917	1.5505515	1.5571088	1.5574077

(Figure VI-5)

Method: FOURTH-ORDER RUNGE-KUTTA METHOD		
Problem: $y' = y^2 + 1;\ y(0) = 0$		
x_n	$h = 0.1$ y_n	True solution $Y(x) = \tan x$
0.0	0.0000000	0.0000000
0.1	0.1003346	0.1003347
0.2	0.2027099	0.2027100
0.3	0.3093360	0.3093363
0.4	0.4227930	0.4227932
0.5	0.5463023	0.5463025
0.6	0.6841368	0.6841368
0.7	0.8422886	0.8422884
0.8	1.0296391	1.0296386
0.9	1.2601588	1.2601582
1.0	1.5574064	1.5574077

(Figure VI-6)

REFERENCES

1. Earl, Modeling, Simulation, and Control, pp. 2-47.

2. Dalley, E.I.T. Review Notes, p. 13-2.

3. Marxheimer, Engineering Systems Class Notes, p. 107.

4. Bronson, Differential Equations, p. 450.

5. Guterman, Differential Equations, p. 206.

6. Ibid., p. 212.

7. Ibid., p. 228.

BIBLIOGRAPHY

Bronson, Robert Differential Equations. New York: McGraw-Hill, 1973.

Eyman, Earl Modeling, Simulation and Control. New York: West Publishing Company, 1988.

Dalley, Joseph W. E.I.T. Review Notes. Austin: University of Texas, 1980.

Guterman, Martin M. Differential Equations. New York: Saunders College Publishing, 1984.

Marxheimer, R.B. Engineering Systems Class Notes. San Francisco: San Francisco State University, 1986.

APPENDIX 1

CODE

```
22      Ta
180     Te
0.3     HTR
0.05    h
200     PrintIndex
2000    Vrequested
1000    V1
1000    V2
1000    V3
0       Vr
100     Vh
0       Vf
500     P4
500     P5
50      P6
28.125  P7
500     P8
110     Vrcap
1000    Vhcap
3       Vfcap
12      Vbcap
```

```
program IE513;

                    { Process Control Simulation Program }


                      { Submitted to Dr. D. W. Russel }
                               { for IE 513 }

                { by Philip G. Swayne and Joseph T. Wunderlich }

                               { July 25, 1990 }


uses Printer, Crt, Dos, Graph;
type
   Volumes = (V1,V2,V3,V4,Vh,Vr,Vf,Vb);       { discrete array coefficients }
                                              { one for each tank in system }
   Vol_Values = array [Volumes] of real;     { defines a generic real array }
                                  { having the above possible coefficients }
                                        { allowing indexing of all tanks }
   IVol_Values = array [Volumes] of integer;  { generic integer array, same }
                                   { as above, just for output purposses }
   States = (Fill,Heat,Cool,Drain);   { distinct possible states for cooker }
   Message = string[20];                     { text for all system messages }
const
   CoolingConstant = 0.1;                  { values provided by the recipe }
   T_trgtheat = 140.0;                   { heating cut-off point for cooker }
   T_trgtcool = 100.0;                   { cooling cut-oof point }

var
   V, dVdt : Vol_Values;                                 { tank volume arrays }
   V1min, V2min, V3min, V1ref, V2ref, V3ref : real;           { tank limits }
   Vrmax, Vrmin, Vhmax, Vhmin, Vfmax, Vfmin, Vflow, Vbmax : real;
   Vrcap, Vhcap, Vfcap, Vbcap : real;                     { tank capacities }

   dTrdt, Tr, Ta, Te : real;                       { temperatures of reactor }
   HTR : real;                                              { heater rating }

   P1, P2, P3, P4, P5, P6, P7, P8 : real;                     { pump rates }

   h : real;                             { integration constant/sampling rate }

   U1, U2, U3, U4, U5, U6, U7, U8, Uh, Uc : integer;      { normal controls }
   Um, Ur, Us, Ui, Uk1, Uk2, Uk3 : integer;               { master controls }

   NumberOfBottles : integer;                    { ongoing reported results }
   Vdone : real;                               { total volume of product made }
   PercentOverfill, BottlingRate : real;    { system performance parameters }
   Msg1, Msg2, Msg3 : message;                               { system messages }
   CumTime, RealTime : real;                                  { current time }


   Hour1, Hour2, Minute1, Minute2,                    { for real time clock }
      Second1, Second2, Sec1001, Sec1002 : word;

   Tank : Volumes;                { actual coefficients for all tank arrays }

   Vinitial, Vtogo : real;       { volume in tanks at start and left to make }

   Vingred_used, Vfinished_prod, Vrequested : integer;
                                  { integer values used for output purposses }
   V_ : IVol_Values;         { array volume values, for output purposes only }

   Vblost : real;             { amount of overfill for each bottle, or waste }
   Vmaxlostbatch : real;                       { estimate of total batch loss }
                                 { to decide when enough has been cooked }
```

```
   State, PreviousState : States;       { State and PreviousState of cooker }
                                        { as having four possible values: }
                                        { Fill, Heat, Cool, Drain         }

   in_file : text; { stores name of file with all user input initial values }
   out_file : text;     { stores name of file to which all results are sent }

   PrintIndex, PrintCount : integer;           { holds number of how often }
                               { 'less frequent' printouts are to be made }
                              { and count of how long since last printout }

   TextForVideo : message;          { holds text to be sent to Video screen }
   MaxX, MaxY : integer;           { maximum screen dimensions for graphics }
   xBoarder, yBoarder : integer;       { holds width of all display boarder }
   xWidth, yHeight : integer;            { holds dimensions of display bars }
   GraphDriver, GraphMode : integer;    { holds graphics driver information }

   UrDelayed1, UrDelayed2 : integer;   { flag for graphics output, set to 1 }
               { one and two h-cycles, respectively, after Ur is set to 1 }

   StateInd, MsgInd : integer;                        { internal message flag }

   ch : char;               { dummy variable, used in INPUT at end to delay }
                                       { termination of graphics display }

   DisasterFlag : integer;{ flag giving type of disaster which has occurred }

{*************************************************************************}
                        { PROCEDURES and FUNCTIONS }
{*************************************************************************}
                                                    { DISASTER Procedure }

procedure DisasterInControls(Trigger : integer; var DisasterFlag : integer;
                      P1,P2,P3,P4,P5,P6,P7,P8,HTR : real);
var
   DP1, DP2, DP3, DP4, DP5, DP6, DP7, DHTR, DC, DB : real;
begin
DisasterFlag:=0;                                              { disaster flag }
DP1:=1; DP2:=1; DP3:=1; DP4:=1; DP5:=1; DP6:=1; DP7:=1;{ initial conditions }
DHTR:=1; DC:=1; DB:=1;
if Trigger=1 then
   begin
   U1:=U1*1; U2:=U2*1; U3:=U3*1; U4:=U4*1; U5:=U5*1; U6:=U6*1; U7:=U7*1;
   U8:=U8*1; Uh:=Uh*1; Uc:=Uc*1; Ui:=Ui*1; Um:=Um*1; Ur:=Ur*1; Us:=Us*1;
   P1:=P1*DP1; P2:=P2*DP2; P3:=P3*DP3; P4:=P4*DP4; P5:=P5*DP5; P6:=P6*DP6;
   P7:=P7*DP7; HTR:=HTR*DHTR;
   end
end;


{*************************************************************************}
                                                  { GetTimeDiff Procedure }
procedure GetTimeDiff(var Hour1, Minute1, Second1, Sec1001 : word;
                      Hour2, Minute2, Second2, Sec1002 : word);
                                  { This procedure receives two sets of }
                                 { time, the earlier first, and returns }
                               { the difference in the slot of the second }
                               { in the same format.  It does not account }
                               { for the PM to AM time change.           }
begin
   if Sec1001>Sec1002 then   { check if the first is bigger than the second }
      begin
      Sec1002:=Sec1002+100;
      Second2:=Second2-1
      end;
```

```
   Sec1002:=Sec1002-Sec1001; { check if the first is bigger than the second }
   if Second1>Second2 then
      begin
      Second2:=Second2+60;
      Minute2:=Minute2-1
      end;
   Second2:=Second2-Second1; { check if the first is bigger than the second }
   if Minute1>Minute2 then
      begin
      Minute2:=Minute2+60;
      Hour2:=Hour2-1
      end;
   Minute2:=Minute2-Minute1; { check if the first is bigger than the second }
   if Hour1>Hour2 then
      begin
      OutTextXY(MaxX div 2,MaxY div 2,'PM to AM time change error!');
      end;
   Hour2:=Hour2-Hour1
end;


{*****************************************************************************}
                                                    { RealSeconds function }
function RealSeconds(Hr1,Min1,Sec1,Hun1,Hr2,Min2,Sec2,Hun2 : word) : real;
                { This function receives two times, the earlier first, and }
                { returns the difference in seconds as the function.       }
   begin
      RealSeconds:=(3600.0*Hr2+60.0*Min2+Sec2+0.01*Hun2)-
                   (3600.0*Hr1+60.0*Min1+Sec1+0.01*Hun2)
   end;


{*****************************************************************************}
                                  { video screen VOLUME SET UP procedure }
procedure VideoVsetup(x1,y1,xBoarder,yBoarder,xWidth,yHeight : integer;
                      V,Vcap : real);
                  { This procedure prints, in graphics, a bar from X1,Y1 to }
             { X1+xWidth,Y1+yHeight, and also puts a boarder around the bar }
            { with a thickness of xBoarder in the x direction, and yBoarder }
            { in the y direction.  The inside bar is actually two bars. The }
            { lower bar represents the proportion of Vcap that V has, up to }
            { 100%, and the upper bar follows the lower just keeping the    }
            { screen background color where V is not located. }
     { Together the bars represent a tank level (or temperature) indicator. }

var
   x0, x2, x3, x4, y0, y2, y3, y4 : integer;
begin
   x0:=x1-xBoarder;                          { get boarder upper-left corner }
   y0:=y1-yBoarder;
   x3:=x1+xWidth;                         { get lower bar lower-right corner }
   y3:=y1+yHeight;                     { upper bar upper-left corner is x1,y1 }
   x4:=x3+xBoarder;                         { get boarder lower-right corner }
   y4:=y3+yBoarder;
   y2:=round(V/Vcap*(y1-y3)+y3);    { y2 is the boarder between the two bars }
                                    { upper bar lower-right corner is x3,y2 }
                                     { lower bar upper-left corner is x1,y2 }
   SetFillStyle(1,3);                                   { do boarder outline }
   Bar(x0,y0,x4,y4);
   SetFillStyle(7,0);                          { do air space level, upper bar }
   Bar(x1,y1,x3,y2);
   SetFillStyle(1,1);                              { do fluid level, lower bar }
   Bar(x1,y2,x3,y3)
end;
```

```
{*************************************************************************}
                                  { video screen VOLUME CHANGE procedure }
procedure VideoVchange(x1,y1,xWidth,yHeight : integer;
                       V,Vcap : real);
                { This procedure prints, in graphics, two bars between X1,Y1 }
       { and X1+xWidth,Y1+yHeight.  The lower bar represents the proportion }
      { of Vcap that V has, up to 100%, and the upper bar follows the lower }
         { just keeping the screen background color where V is not located. }
     { Together the bars represent a tank level (or temperature) indicator. }
var
   x3, y2, y3 : integer;
begin
   x3:=x1+xWidth;                       { get lower bar lower-right corner }
   y3:=y1+yHeight;                    { upper bar upper-left corner is x1,y1 }
   y2:=round(V/Vcap*(y1-y3)+y3);    { y2 is the boarder between the two bars }
                                     { upper bar lower-right corner is x3,y2 }
                                      { lower bar upper-left corner is x1,y2 }

   SetFillStyle(7,0);                                { do air space, upper bar }
   Bar(x1,y1,x3,y2);
   SetFillStyle(1,1);                               { do fluid level, lower bar }
   Bar(x1,y2,x3,y3);
end;


{*************************************************************************}
                                                    { INTEGRATION Function }
function Integrate(Y,dYdt,h : real ) : real;
                                                          { Euler's Method }
begin
   Integrate := Y + h*dYdt;
end;


{*************************************************************************}
                              { MAIN PROGRAM }
{*************************************************************************}
                                                            { Main Program }
begin
   GetTime(hour1, minute1, second1, sec1001);       { get initial real time }
   assign(out_file,'IE513OUT.DAT');                   { open a results file }
   rewrite(out_file);
   assign(in_file,'IE513IN.DAT');                     { open the Input file }
   reset(in_file);

{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
                                                          { INITIAL VALUES }
    readln(in_file,Ta);           { read in values that are user changeable }
    readln(in_file,Te);           { from an input file }
    readln(in_file,HTR);
    readln(in_file,h);
    readln(in_file,PrintIndex);
    readln(in_file,Vrequested);

    readln(in_file,V[V1]);
    readln(in_file,V[V2]);
    readln(in_file,V[V3]);
    readln(in_file,V[Vr]);
    readln(in_file,V[Vh]);
    readln(in_file,V[Vf]);

    readln(in_file,P4);
    readln(in_file,P5);
    readln(in_file,P6);
    readln(in_file,P7);
    readln(in_file,P8);
```

```
    readln(in_file,Vrcap);
    readln(in_file,Vhcap);
    readln(in_file,Vfcap);
    readln(in_file,Vbcap);

    close(in_file);                              { close the input file }
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
   Tr:=Ta;           { the ingredients are taken as at ampient temperature }

   P1:=P4/4; P2:=P4/2; P3:=P4*3/4;               { P1-P3 are functions of P4 }

   Vinitial:=V[Vh]+V[Vf]+V[Vb];              { check volume already in system }
   Vtogo:=Vrequested;                            { have to make all requested }
   V[Vb]:=0;                                     { start with fresh bottle }
   V[V4]:=30000;               { start with a very high volume, ie limitless }

   V1min:=P1*h/60; V2min:=P2*h/60; V3min:=P3*h/60;
   Vrmax:=Vrcap-10.0-(P1+P2+P3+P4)*h/60; Vrmin:=P5*h/60;
   Vhmax:=Vhcap-1.1*Vrcap; Vhmin:=P6*h/60;
   Vfmax:=Vfcap-P6*h/60; Vflow:=Vbcap*3*(1.0/128.0); Vfmin:=P7*h/60;
   Vbmax:=Vbcap*(1.0/128.0);

   V1ref:=V[V1]; V2ref:=V[V2]; V3ref:=V[V3];

   NumberOfBottles:=0; BottlingRate:=0; Vdone:=0;
   Vblost:=0; PercentOverFill:=0;

   CumTime:=0;                                { start Simulated Run Time at 0 }

   PrintCount:= PrintIndex;       { set count to index to print first cycle }

   Msg1:=' '; Msg2:=' '; Msg3:=' '; MsgInd:=0;
                                              { all controls initially off }
   U1:=0;U2:=0;U3:=0;U4:=0;U5:=0;U6:=0;U7:=0;U8:=0;Uh:=0;Uc:=0;
   Um:=0; Ur:=0; Ui:=0; Us:=0; UrDelayed1:=0; UrDelayed2:=0;
   Uk1:=1; Uk2:=1; Uk3:=1;
                                                   { initial state is Fill }
   State:=Fill; PreviousState:=State;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
                                                   { initial input checks }
   if (V[Vr]>Vrcap) or (V[Vh]>Vhcap) or (V[Vf]>Vfcap)
                                 or (V[Vb]>Vbcap) then Um:=1;
   if PrintIndex<=0 then Um:=1;
   if (h<=0) or (HTR<=0) or (Vrequested<=0) then Um:=1;
   if (Vrcap<=0) or (Vhcap<=0) or (Vfcap<=0) or (Vbcap<=0) then Um:=1;
   if (P1<0) or (P2<0) or (P3<0) or (P4<0) or (P5<0) or (P6<0) or (P7<0)
                                                   or (P8<0) then Um:=1;
   if (V[V1]<0) or (V[V2]<0) or (V[V3]<0) or (V[Vr]<0) or (V[Vh]<0)
                                       or (V[Vf]<0) then Um:=1;
   if (Te<140) or (Ta>100) then Um:=1;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
                                                   { set up VIDEO SCREEN }
                                     { get current computer's parameters }
   GraphDriver:=Detect;
   InitGraph(GraphDriver,GraphMode,'');

   MaxX:=GetMaxX;                      { get maximum number of screen pixels }
   MaxY:=GetMaxY;
   xBoarder:=MaxY div 98;              { set display proportions accordingly }
   yBoarder:=MaxY div 98;           { boarders are about 1% of screen height }
   xWidth:=MaxY div 20; { level indicators have width of about 5% of screen }
   yHeight:=MaxY div 4;          { height and height of 25% of screen height }
```

```
                                        { layout display screen format }
VideoVsetup(xWidth*3,1,xBoarder,yBoarder,xWidth,yHeight,V[V1],1000);  {V1}
VideoVsetup(xWidth*6,1,xBoarder,yBoarder,xWidth,yHeight,V[V2],1000);  {V2}
VideoVsetup(xWidth*9,1,xBoarder,yBoarder,xWidth,yHeight,V[V3],1000);  {V3}
VideoVsetup(xWidth*6,yHeight*2,xBoarder,yBoarder,xWidth,yHeight,      {Vr}
            V[Vr],Vrcap-10);
VideoVsetup(xWidth*15,1,xBoarder,yBoarder,xWidth,yHeight*3,           {Vh}
            V[Vh],Vhcap);
VideoVsetup(xWidth*21,yHeight*2,xBoarder,yBoarder,xWidth,yHeight,     {Vf}
            V[Vf],Vfcap);
VideoVsetup(xWidth*2,yHeight*2,xBoarder,yBoarder,xWidth-10,yHeight,   {Tr}
            Tr,T_Trgtheat);
VideoVsetup(MaxX-xWidth*2,1,xBoarder,yBoarder,xWidth,yHeight*2,    {Vdone}
            Vdone,Vrequested);

SetFillStyle(0,0);
Bar(MaxX*3 div 4-150,1,(MaxX*3 div 4),10);    { label for Simulated Time }
OutTextXY(MaxX*3 div 4-150,1,'Simulated Time:');
Bar(MaxX*3 div 4-150,20,(MaxX*3 div 4),30);         { label for RealTime }
OutTextXY(MaxX*3 div 4-150,20,'Real Time:');

Bar(MaxX*3 div 4-150,50,(MaxX*3 div 4),60);    { label for Bottling Rate }
OutTextXY(MaxX*3 div 4-150,50,'Bottling Rate:');
Bar(MaxX*3 div 4-150,70,(MaxX*3 div 4),80);    { label for Percent Waste }
OutTextXY(MaxX*3 div 4-150,70,'Percent Waste:');
Bar(MaxX*3 div 4-150,90,(MaxX*3 div 4+20),100);  { label NumberOfBottles }
OutTextXY(MaxX*3 div 4-150,90,'Number of Bottles:');
Bar(xWidth*2-10,yHeight*2-20,xWidth*2+40,yHeight*2-10);     { label Temp }
OutTextXY(xWidth*2-10,yHeight*2-20,'TEMP:');
Bar(MaxX-xWidth*2-20,yHeight*2+30,MaxX-xWidth*2+40,yHeight*2+50);
OutTextXY(MaxX-xWidth*2-20,yHeight*2+30,'Volume');        { label Volume }
OutTextXY(MaxX-xWidth*2-20,yHeight*2+40,' Done:');

                              { layout pipes and "good-looks" features }
line(xWidth*3+xWidth div 2,yHeight,xWidth*6+xWidth div 2,yHeight*2);
line(xWidth*6+xWidth div 2,yHeight,xWidth*6+xWidth div 2,yHeight*2);
line(xWidth*9+xWidth div 2,yHeight,xWidth*6+xWidth div 2,yHeight*2);
line(xWidth*11+xWidth div 2,0,xWidth*11+xWidth div 2,yHeight+25);
line(xWidth*11+xWidth div 2,yHeight+25,xWidth*6+xWidth div 2,yHeight*2);
line(xWidth*6+xWidth div 2,yHeight*3,xWidth*6+xWidth div 2,yHeight*3+25);
line(xWidth*6+xWidth div 2,yHeight*3+25,
     xWidth*12+xWidth div 2,yHeight*3+25);
line(xWidth*12+xWidth div 2,yHeight*3+yHeight*2 div 3,
     xWidth*12+xWidth div 2,0);
line(xWidth*12+xWidth div 2,0,xWidth*15+xWidth div 2,0);
line(xWidth*15+xWidth div 2,0,xWidth*15+xWidth div 2,1);
line(xWidth*15+xWidth div 2,yHeight*3,
     xWidth*15+xWidth div 2,yHeight*3+25);
line(xWidth*15+xWidth div 2,yHeight*3+25,
     xWidth*18+xWidth div 2,yHeight*3+25);
line(xWidth*18+xWidth div 2,yHeight*3+25,
     xWidth*18+xWidth div 2,yHeight*2-20);
line(xWidth*18+xWidth div 2,yHeight*2-20,
     xWidth*21+xWidth div 2,yHeight*2-20);
line(xWidth*21+xWidth div 2,yHeight*2-20,
     xWidth*21+xWidth div 2,yHeight*2);
line(xWidth*21+xWidth div 2,yHeight*3,
     xWidth*21+xWidth div 2,yHeight*3+25);
line(xWidth*21+xWidth div 2,yHeight*3+25,
     xWidth*25,yHeight*3+25);

                                          { pump symbols in line }
circle(xWidth*5,yHeight+yHeight div 2,7);                          { P1 }
circle(xWidth*6+xWidth div 2,yHeight+yHeight div 2,7);             { P2 }
circle(xWidth*8,yHeight+yHeight div 2,7);                          { P3 }
circle(xWidth*10,yHeight+yHeight div 2,7);                         { P4 }
```

```
   circle(xWidth*12+xWidth div 2,yHeight+yHeight div 2,7);            { P5 }
   circle(xWidth*12+xWidth div 2,yHeight*3+yHeight div 2,7);          { P6 }
   circle(xWidth*20,yHeight*2-20,7);                                  { P7 }
   circle(xWidth*24,yHeight*3+25,7);                                  { P8 }
   circle(xWidth*5,yHeight*2+yHeight div 2,13);                      { HTR }


{*************************************************************************}
{************************************************** MAIN LOOP *****}
{*************************************************************************}
                          { Main Loop - Once for each cycle h seconds long }
while (Um=0) do
   begin

{-------------------------------------------------------------------------}
                                                  { DIFFERENTIAL EQUATIONS }
   dVdt[V1] := -P1*U1*Uk1*1/60;             { Volume results are in gal/sec }
   dVdt[V2] := -P2*U2*Uk1*1/60;
   dVdt[V3] := -P3*U3*Uk1*1/60;
   dVdt[V4] := -P4*U4*Uk1*1/60;
   dVdt[Vr] := ((P1*U1+P2*U2+P3*U3+P4*U4)*Uk1-(P5*U5+P8*U8)*Uk2)*1/60;
   dVdt[Vh] := (P5*U5*Uk2-P6*U6*Uk3)*1/60;
   dVdt[Vf] := (P6*U6-P7*U7)*Uk3*1/60;
   dVdt[Vb] := P7*U7*Uk3*1/60;
                                               { results are in deg C/sec }
   dTrdt := (HTR*Uh*Uk2*((Te-Tr)*(Te-Tr))-CoolingConstant*
            (Tr-Ta)*(50*(1-Uh)))*1/60;

{-------------------------------------------------------------------------}
                                     { internal for evaluation purposes }
{   if (State=Fill) then StateInd:=1; if (State=Heat) then StateInd:=2;
   if (State=Cool) then StateInd:=3; if (State=Drain) then StateInd:=4;}

{-------------------------------------------------------------------------}
                                                  { SCREEN and FILE OUTPUT }
                        { update the screen if cooking is still in process }
if (UrDelayed2=0) then                                 { Ur DEPENDENT Output }
   begin
   SetFillStyle(0,0);
   Bar(xWidth*2,yHeight*3+30,xWidth*12,yHeight*3+50);
   OutTextXY(xWidth*2,yHeight*3+30,Msg1);                  { update State msg }
   OutTextXY(xWidth*2,yHeight*3+40,Msg2);                 { update Reactor msg }

   case State of
{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
   Fill:                                                              { Fill }
      begin
                                                 { update analog volume }
      VideoVchange(xWidth*3,1,xWidth,yHeight,V[V1],1000);       { analog V1 }
      VideoVchange(xWidth*6,1,xWidth,yHeight,V[V2],1000);       { analog V2 }
      VideoVchange(xWidth*9,1,xWidth,yHeight,V[V3],1000);       { analog V3 }
      VideoVchange(xWidth*6,yHeight*2,xWidth,yHeight,V[Vr],Vrcap-10);  { Vr }

      SetFillStyle(0,0);                          { update digital volume }
      Bar(xWidth*2,yHeight+yBoarder*4-2,xWidth*11,yHeight+yBoarder*4+8);
      Str(V[V1] : 3 : 0,TextForVideo);                  { update digital V1 }
      OutTextXY(xWidth*3,yHeight+yBoarder*4,TextForVideo);
      Str(V[V2] : 3 : 0,TextForVideo);                  { update digital V2 }
      OutTextXY(xWidth*6,yHeight+yBoarder*4,TextForVideo);
      Str(V[V3] : 3 : 0,TextForVideo);                  { update digital V3 }
      OutTextXY(xWidth*9,yHeight+yBoarder*4,TextForVideo);
      Str(V[Vr] : 3 : 1,TextForVideo);                  { update digital Vr }
      Bar(xWidth*6-10,yHeight*3+yBoarder*4-2,
          xWidth*6+40,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*6-10,yHeight*3+yBoarder*4,TextForVideo)
      end;
```

```
{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  Drain:                                              { Drain }
    begin
                                             { update analog volume }
    VideoVchange(xWidth*6,yHeight*2,xWidth,yHeight,V[Vr],Vrcap-10);  { Vr }
    VideoVchange(xWidth*15,1,xWidth,yHeight*3,V[Vh],Vhcap);          { Vh }

    SetFillStyle(0,0);                       { update digital volume }
    Str(V[Vr] : 3 : 1,TextForVideo);             { update digital Vr }
    Bar(xWidth*6-10,yHeight*3+yBoarder*4-2,
        xWidth*6+40,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*6-10,yHeight*3+yBoarder*4,TextForVideo);
    Str(V[Vh] : 4 : 1,TextForVideo);             { update digital Vh }
    Bar(xWidth*15-10,yHeight*3+yBoarder*4-2,
        xWidth*15+50,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*15-10,yHeight*3+yBoarder*4,TextForVideo)
    end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  Heat:                                                { Heat }
    begin
                                                 { update analog Tr }
    VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);

    SetFillStyle(0,0);
    Str(Tr : 3 : 1,TextForVideo);                { update digital Tr }
    Bar(xWidth*2-10,yHeight*3+yBoarder*4-2,
        xWidth*2+40,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
    end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  Cool:                                                { Cool }
    begin
                                                 { update analog Tr }
    VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);

    SetFillStyle(0,0);
    Str(Tr : 3 : 1,TextForVideo);                { update digital Tr }
    Bar(xWidth*2-10,yHeight*3+yBoarder*4-2,
        xWidth*2+40,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
    end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  end;                                             { CASE end }

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
  end;                                 { end Ur DEPENDENT output }

                              { analog update on a continuous basis }
  VideoVchange(xWidth*21,yHeight*2,xWidth,yHeight,V[Vf],Vfcap);     { Vf }

                             { digial updates on a continuous basis }
  SetFillStyle(0,0);
  Bar(xWidth*15,yHeight*3+40,xWidth*25,yHeight*3+50);
  OutTextXY(xWidth*15,yHeight*3+40,Msg3);    { update Lower Controls msg }

  Str(V[Vf] : 1 : 2,TextForVideo);               { update digital Vf }
  Bar(xWidth*21-5,yHeight*3+yBoarder*4-2,
      xWidth*21+35,yHeight*3+yBoarder*4+8);
  OutTextXY(xWidth*21-5,yHeight*3+yBoarder*4,TextForVideo);

  Str(CumTime : 4 : 2,TextForVideo);        { update digital CumTime }
  Bar(MaxX*3 div 4,1,(MaxX*3 div 4)+70,10);
```

```
   OutTextXY(MaxX*3 div 4,1,TextForVideo);

   GetTime(Hour2,Minute2,Second2,Sec1002);        { update digital RealTime }
   RealTime:=RealSeconds(Hour1,Minute1,Second1,Sec1001
                        ,Hour2,Minute2,Second2,Sec1002);
   Str(RealTime : 4 : 2,TextForVideo);
   Bar(MaxX*3 div 4,20,(MaxX*3 div 4)+70,30);
   OutTextXY(MaxX*3 div 4,20,TextForVideo);

   Str(NumberOfBottles : 5 ,TextForVideo);        { update count of bottles }
   Bar(MaxX*3 div 4,90,(MaxX*3 div 4)+60,100);           { NumberOfBottles }
   OutTextXY(MaxX*3 div 4,90,TextForVideo);

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
                     { Output not Ur dependent but not printed continuously }
                                          { PRINTCOUNT dependent output }
                  { write results to screen and file }
                  { print values if changed state or }
                  { every "PrintIndex" times of h    }
   if (PrintCount >= PrintIndex) or (State <> PreviousState) or
      (Vdone >= Vrequested) then            { print if conditions are right }
      begin
{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  SCREEN OUTPUT }
                                  { analog update on a less frequent basis }
      VideoVchange(xWidth*15,1,xWidth,yHeight*3,V[Vh],Vhcap);          { Vh }
      VideoVchange(MaxX-xWidth*2,1,xWidth,yHeight*2,Vdone,Vrequested); { Vr }
      VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);{ Tr }

                                { digital update on a less frequent basis }
      SetFillStyle(0,0);
      if (State=Fill) or (State=Drain) then
         begin
         SetFillStyle(0,0);
         Str(Tr : 3 : 1,TextForVideo);                  { update digital Tr }
         Bar(xWidth*2-10,yHeight*3+yBoarder*4,
             xWidth*2+40,yHeight*3+yBoarder*4+10);
         OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
         end;

      Str(V[Vh] : 4 : 1,TextForVideo);                  { update digital Vh }
      Bar(xWidth*15-10,yHeight*3+yBoarder*4-2,
          xWidth*15+50,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*15-10,yHeight*3+yBoarder*4,TextForVideo);

      Str(Vdone : 4 : 1,TextForVideo);               { update digital Vdone }
      Bar(MaxX-xWidth*2-10,yHeight*2+yBoarder*4,MaxX-xWidth*2+50
          ,yHeight*2+yBoarder*4+10);
      OutTextXY(MaxX-xWidth*2-10,yHeight*2+yBoarder*4,TextForVideo);

      Str(BottlingRate : 4 : 1,TextForVideo);        { update Bottling Rate }
      Bar(MaxX*3 div 4,50,(MaxX*3 div 4)+60,60);
      OutTextXY(MaxX*3 div 4,50,TextForVideo);

      Str(PercentOverfill : 3 : 1,TextForVideo);     { update Percent Waste }
      Bar(MaxX*3 div 4,70,(MaxX*3 div 4)+50,80);
      OutTextXY(MaxX*3 div 4,70,TextForVideo);

{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  FILE OUTPUT }
      Vingred_used:=round((1000-V[V1])*10+Vinitial);
      Vfinished_prod:=round(Vdone+V[Vr]+V[Vh]+V[Vf]+V[Vb]);

      for Tank:= V1 to Vb do
         begin
            V_[Tank]:= round(V[Tank])     { get printable values of volumes }
         end;
      V_[Vf]:=round(V[Vf]*128);   { get value of Fill tank volume in ounces }
```

```
      writeln(out_file,V_[V1],'      ',V_[V2],'      ',V_[V3],'      ',V_[Vr]
         ,'      ',V_[Vh],'      ',V_[Vf],'      ',CumTime,'      ',Vdone);

      PrintCount:=1;              { reset count for time till next printout }
      PreviousState:=State        { reset State to detect change of state   }
      end
   else
{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
                                  { if counter not right then don't print }
                                  { but increment counter }
       PrintCount:=PrintCount+1;
                                     { end of PRINTCOUNT dependent output }

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
                               { check for Ur shut down, to save resources }
   if (UrDelayed1<>0) then UrDelayed2:=1;   { for 2 cycle delay in shutting }
   if (Ur<>0) then UrDelayed1:=1;      { off Reactor Control area displays  }
                                     { once Ur sets, the display will go  }
                                   { on for two cycles then stop changing }
                                                          { OUTPUT end }

{-------------------------------------------------------------------------}
                                                        { INTEGRATIONS }

   for Tank:= V1 to Vb do                        { integration for each tank }
      begin
         V[Tank] := Integrate( V[Tank], dVdt[Tank], h )
      end;
   Tr:= Integrate( Tr, dTrdt, h );    { integration for reactor temperature }

{-------------------------------------------------------------------------}
                                                    { REACTOR CONTROLS }

   if (Um=0) and (Ur=0) then                            { REACTOR CONTROL LOOP }
                                   { operated reactor if Um of Ur not set }
      begin
      case State of                                          { start CASE }
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
         Fill:                                                    { FILL }
            begin
            if (Ui=0) then
               begin
               Vmaxlostbatch:= Vtogo*P7*h*(0.1778);
               if (Vtogo+Vmaxlostbatch > V[Vh]+V[Vf]) then
                  begin
                  if (V[Vh] < Vhmax) then
                     begin
                     if (V[Vr] < Vrmax) then
                        begin
                        if (V1ref>=Vrmax/10) and (V2ref>=Vrmax*2/10) and
                        (V3ref>=Vrmax*3/10) then
                           begin
                           U1:=1; U2:=1; U3:=1; U4:=1;
                           U5:=0; U8:=0; Uh:=0;
                           Msg1:='Filling             '    ;MsgInd:=1
                           end                          { send SYSTEM MESSAGE }
                        else
                           begin
                           Ur:=1;
                           U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; UH:=0;
                           Msg1:='Out of ingredients'   ;MsgInd:=2
                           end                          { send SYSTEM MESSAGE }
                        end
                     else
                        begin
```

```
                  State:= Heat;
                  Uh:=1;
                  U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0;
                  Msg1:='Heating             '       ;MsgInd:=3
                  end                              { send SYSTEM MESSAGE }
               end
            else
               begin
               U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
               Msg1:='Waiting to fill'                ;MsgInd:=4
               end                                 { send SYSTEM MESSAGE }
            end
         else
            begin
            Ur:=1;
            U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
            Msg1:='Done Cooking'                    ;MsgInd:=5
            end                                  { send SYSTEM MESSAGE }
         end
      else
         begin
         U1:=0; U2:=0;U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
         Msg1:='Suspended filling'             ;MsgInd:=6
         end                                     { send SYSTEM MESSAGE }
      end;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
      Heat:                                                      { HEAT }
         begin
         if (Tr < T_trgtheat) then
            begin
            Uh:=1;
            U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0;
            Msg1:='Heating'                          ;MsgInd:=7
            end                                     { send SYSTEM MESSAGE }
         else
            begin
            State:= Cool;
            U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
            Msg1:='Cooling'                          ;MsgInd:=8
            end                                     { send SYSTEM MESSAGE }
         end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
      Cool:                                                      { COOL }
         begin
         if (Tr > T_trgtcool) then
            begin
            U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
            Msg1:='Cooling'                          ;MsgInd:=9
            end                                     { send SYSTEM MESSAGE }
         else
            begin
            State:= Drain;
            U5:=1;
            U1:=0; U2:=0; U3:=0; U4:=0; U8:=0; Uh:=0;
            Msg1:='Draining'                      ;MsgInd:=10
            end                                     { send SYSTEM MESSAGE }
         end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
      Drain:                                                    { DRAIN }
         begin
         if (V[Vr] > Vrmin) then
            begin
            U5:=1;
            U1:=0; U2:=0; U3:=0; U4:=0; U8:=0; Uh:=0;
```

```
            Msg1:='Draining'                              ;MsgInd:=11
            end                                     { send SYSTEM MESSAGE }
         else
            begin
            State:= Fill;
            U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
            Msg1:='Ready to fill';                        ;MsgInd:=12;
            Tr:=Ta;                                 { send SYSTEM MESSAGE }
            V1ref:=V[V1];
            V2ref:=V[V2];
            V3ref:=V[V3]
            end
         end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
      end                                                  { end of CASE }
      end

{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
   else                                        { REACTOR CONTROL LOOP else }
      begin
      U1:=0; U2:=0; U3:=0; U4:=0; U5:=0; U8:=0; Uh:=0;
      if (Um <> 0) then Msg2:='System shut down'      { send SYSTEM MESSAGE }
      else
         begin
         if (Ur <> 0) then Msg2:='Reactor shut down'  { send SYSTEM MESSAGE }
         else Msg2:='Error in controls'               { send SYSTEM MESSAGE }
         end
      end;                                       { REACTOR CONTROL LOOP end }

{-------------------------------------------------------------------------}
                                                          { LOWER CONTROL }
                                                          { P6 and below  }

      if (V[Vh] > Vhmin) and (Uk3=1) and (V[Vf] < Vfmax) then U6:=1
      else U6:=0;       { turn U6 on if master control not set ( Us or Um ) }
                              { and if Hold tank not empty and if Fill tank }
                                          { less than its maximum value }

      if (Uk3=1) then            { if master controls Us or Um not set then }
                           { ok to bottle and conveyor - LOWER CONTROL LOOP }
         begin
         if (V[Vb] < Vbmax) and (V[Vf] > Vfmin) then
                                    { run P7 and don't conveyor if bottle }
                                       { not full and Fill tank not empty }
            begin
            U7:=1;
            Uc:=0;                                { send SYSTEM MESSAGE }
            Msg3:='Filling bottle'                        ;MsgInd:=13
            end
         else
            begin
            if (V[Vb] >= Vbmax) then
                           { conveyor and don't run P7 if bottle is full }
               begin
               Uc:=1;
               U7:=0;                             { send SYSTEM MESSAGE }
               Msg3:='Inserting new bottle';              MsgInd:=14;

                              { evaluate parameters that change only when }
                                 { more bottles of product are completed }

               Vblost:=Vblost+(V[Vb]*128-Vbcap);   { add amount of overfill }
                                       { for this bottle to total so far }
                                             { Vblost is cumulative loss }
```

```
          NumberOfBottles:=NumberOfBottles + 1;      { add one more to }
                                                    { count of bottles done }

          Vdone:=NumberOfBottles*(12.0/128.0); { get total volume done }
                                       { based on total number of bottles }
                                       { produced, not on total amount of }
                                  { materials used, therefore if waste is }
                               { not zero then more volume of ingredients }
                                           { than end product is required }

          Vtogo:=Vrequested-Vdone;   { adjust volume needed to be made }
                                             { based on completed product }

          PercentOverfill:=Vblost/Vbcap/NumberOfBottles*100.0;
                                  { calculate total percent waste/overfill }
                  { as [(cum loss)/(# bottles)]/(bottle capacity)*100% }
             { where (cum loss)/(# bottles) is average loss per bottle }

          V[Vb]:=0         { set bottle volume back to 0 for new bottle }
          end
       else        { don't conveyor or run P7 since Fill tank is empty }
          begin
          U7:=0; Uc:=0;                          { send SYSTEM MESSAGE }
          Msg3:='Low level- Fill tank';               MsgInd:=15;
          end
       end
    end
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
   else                                        { LOWER CONTROL LOOP else }
      begin
                                       { don't conveyor or run P7 since }
                                       { either Us or Um is set          }
      U7:=0; Uc:=0;                                { send SYSTEM MESSAGE }
      Msg3:='Suspended bottling'                       ;MsgInd:=16
      end;                                      { LOWER CONTROL LOOP end }

{-----------------------------------------------------------------------}
                                                    { DISASTER CONTROLS }

  if ((DisasterFlag>=1) and (DisasterFlag<=5)) or (DisasterFlag=8) then
    begin
    Ui:=1;
    if V[Vr]>Vrmin then U8:=1       { if disaster in pumps of upper area }
    else                            { and reactor not empty then dump it }
      begin
      U8:=0;                                  { else shut down upper area }
      Ur:=1
      end;
    Msg3:='Disaster in top section'                { send SYSTEM MESSAGE }
    end;
  if DisasterFlag=6 then
    begin
    if State=Fill then  { if P6 is broken then finish batch and shut down }
    else Ui:=1;
    Msg3:='P6 is broken'                           { send SYSTEM MESSAGE }
    end;
  if DisasterFlag=7 then
    begin
    if State=Fill then  { if P7 is broken then finish batch and shut down }
    else Ui:=1;
    Msg3:='P7 is broken'                           { send SYSTEM MESSAGE }
    end;
  if DisasterFlag=9 then
    begin
    Uc:=0;            { if conveyor is broken shut of P7 and the conveyor }
    U7:=0;
```

```
      if State=Fill then  { with conveyor broken finish batch and shut down }
      else Ui:=1;
      Msg3:='Conveyor is broken'                          { send SYSTEM MESSAGE }
      end;
    if DisasterFlag=10 then
      begin
      Uc:=0;                { if out of bottles shut of P7 and the conveyor }
      U7:=0;
      if State=Fill then   { when out of bottles finish batch and shut down }
      else Ui:=1;
      Msg3:='Out of bottles'                              { send SYSTEM MESSAGE }
      end;                                               { DISASTER CONTROLS end }

{-------------------------------------------------------------------------}
                                                          { MASTER CONTROLS }

                                  { master controls are adjusted after all }
                                  { other controls and in response to some }

   if (Vdone >= Vrequested) then Us:=1;          { if enough volume has been }
                                { produced then shut of LOWER CONTROL area }

   if (Ur<>0) and (V[Vh]<=Vhmin) and (V[Vf]<=Vfmin) then Us:=1;    { if all }
                                             { empty then shut off bottler }

   if (Ui<>0) and (V[Vr] <= Vrmin) then Ur:=1;  { if ingredients tanks have }
                                     { have been shut off for some reason, }
                                     { and the reactor is empty, then shut }
                                     { off the entire REACTOR CONTROL area }
   if (Ur<>0) and (Us<>0) then Um:=1;     { if the REACTOR CONTROL area has }
                          { been shut off as well as the LOWER CONTROL area, }
                          { then shut down the entire system                 }

                                          { assign values to CONTROL FLAGS }
                        { used to reduce the number of AND's and OR's needed }
   if (Um=1) or (Ur=1) or (Ui=1)
   then Uk1 := 0
   else Uk1 := 1;                          { Uk1, Uk2 and Uk3 have opposite   }
   if (Um=1) or (Ur=1)                     { logic from other U's in the      }
   then Uk2 := 0                           { system.  Here, if Uk1=0 it means }
   else Uk2 := 1;                          { KILL the system, 1 means let it  }
   if (Um=1) or (Us=1)                     { run.                             }
   then Uk3 := 0
   else Uk3 := 1;                                        { MASTER CONTROLS end }

{-------------------------------------------------------------------------}
                                  { pass controls through disaster routine }
                                               { to allow fault evaluations }

   { DisasterInDetectors(Ta,Te,Tr,V[V1],V[V2],V[V3],V[Vr],V[Vh],V[Vf]);     }
                                                                { not used }

   DisasterInControls(1,DisasterFlag,P1,P2,P3,P4,P5,P6,P7,P8,HTR);

{-------------------------------------------------------------------------}
                                                { MISCELLANEOUS PARAMETERS }
                                    { these parameters change continuously }
                                    { regardless of what controls are done }

   CumTime:=CumTime+h;                { update the current Simulated Run Time }

   BottlingRate:=NumberOfBottles/CumTime*60.0;                { bottling rate }
                                                             { in bottles/min }

end;                    { end of main loop }
```

```
{*************************************************************************}
{************************************************** MAIN LOOP end *****}
{*************************************************************************}


{****************************************************** FINAL OUTPUT *****}
                                                 { SCREEN and FILE OUTPUT }
                          { update the screen if cooking is still in process }
if (UrDelayed2=0) then                                 { Ur DEPENDENT Output }
   begin
   SetFillStyle(0,0);
   Bar(xWidth*2,yHeight*3+30,xWidth*12,yHeight*3+50);
   OutTextXY(xWidth*2,yHeight*3+30,Msg1);                  { update State msg }
   OutTextXY(xWidth*2,yHeight*3+40,Msg2);                 { update Reactor msg }

   case State of
{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
   Fill:                                                                 { Fill }
      begin
                                                  { update analog volume }
      VideoVchange(xWidth*3,1,xWidth,yHeight,V[V1],1000);       { analog V1 }
      VideoVchange(xWidth*6,1,xWidth,yHeight,V[V2],1000);       { analog V2 }
      VideoVchange(xWidth*9,1,xWidth,yHeight,V[V3],1000);       { analog V3 }
      VideoVchange(xWidth*6,yHeight*2,xWidth,yHeight,V[Vr],Vrcap-10);  { Vr }

      SetFillStyle(0,0);                              { update digital volume }
      Bar(xWidth*2,yHeight+yBoarder*4-2,xWidth*11,yHeight+yBoarder*4+8);
      Str(V[V1] : 3 : 0,TextForVideo);                    { update digital V1 }
      OutTextXY(xWidth*3,yHeight+yBoarder*4,TextForVideo);
      Str(V[V2] : 3 : 0,TextForVideo);                    { update digital V2 }
      OutTextXY(xWidth*6,yHeight+yBoarder*4,TextForVideo);
      Str(V[V3] : 3 : 0,TextForVideo);                    { update digital V3 }
      OutTextXY(xWidth*9,yHeight+yBoarder*4,TextForVideo);
      Str(V[Vr] : 3 : 1,TextForVideo);                    { update digital Vr }
      Bar(xWidth*6-10,yHeight*3+yBoarder*4-2,
          xWidth*6+40,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*6-10,yHeight*3+yBoarder*4,TextForVideo)
      end;

{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
   Drain:                                                               { Drain }
      begin
                                                  { update analog volume }
      VideoVchange(xWidth*6,yHeight*2,xWidth,yHeight,V[Vr],Vrcap-10);  { Vr }
      VideoVchange(xWidth*15,1,xWidth,yHeight*3,V[Vh],Vhcap);           { Vh }

      SetFillStyle(0,0);                              { update digital volume }
      Str(V[Vr] : 3 : 1,TextForVideo);                    { update digital Vr }
      Bar(xWidth*6-10,yHeight*3+yBoarder*4-2,
          xWidth*6+40,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*6-10,yHeight*3+yBoarder*4,TextForVideo);
      Str(V[Vh] : 4 : 1,TextForVideo);                    { update digital Vh }
      Bar(xWidth*15-10,yHeight*3+yBoarder*4-2,
          xWidth*15+50,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*15-10,yHeight*3+yBoarder*4,TextForVideo)
      end;

{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
   Heat:                                                                 { Heat }
      begin
                                                   { update analog Tr }
      VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);

      SetFillStyle(0,0);
      Str(Tr : 3 : 1,TextForVideo);                       { update digital Tr }
      Bar(xWidth*2-10,yHeight*3+yBoarder*4-2,
```

```
        xWidth*2+40,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
    end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  Cool:                                              { Cool }
    begin
                                             { update analog Tr }
    VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);

    SetFillStyle(0,0);
    Str(Tr : 3 : 1,TextForVideo);                { update digital Tr }
    Bar(xWidth*2-10,yHeight*3+yBoarder*4-2,
        xWidth*2+40,yHeight*3+yBoarder*4+8);
    OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
    end;

{- - - - - - - - - - - - - - - - - - - - - - - - - - }
  end                                               { CASE end }

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
end;                                     { end Ur DEPENDENT output }

                                 { analog update on a continuous basis }
  VideoVchange(xWidth*21,yHeight*2,xWidth,yHeight,V[Vf],Vfcap);      { Vf }

                                 { digial updates on a continuous basis }
  SetFillStyle(0,0);
  Bar(xWidth*15,yHeight*3+40,xWidth*25,yHeight*3+50);
  OutTextXY(xWidth*15,yHeight*3+40,Msg3);     { update Lower Controls msg }

  Str(V[Vf] : 1 : 2,TextForVideo);               { update digital Vf }
  Bar(xWidth*21-5,yHeight*3+yBoarder*4-2,
      xWidth*21+35,yHeight*3+yBoarder*4+8);
  OutTextXY(xWidth*21-5,yHeight*3+yBoarder*4,TextForVideo);

  Str(CumTime : 4 : 2,TextForVideo);           { update digital CumTime }
  Bar(MaxX*3 div 4,1,(MaxX*3 div 4)+70,10);
  OutTextXY(MaxX*3 div 4,1,TextForVideo);

  GetTime(Hour2,Minute2,Second2,Sec1002);       { update digital RealTime }
  RealTime:=RealSeconds(Hour1,Minute1,Second1,Sec1001
                       ,Hour2,Minute2,Second2,Sec1002);
  Str(RealTime : 4 : 2,TextForVideo);
  Bar(MaxX*3 div 4,20,(MaxX*3 div 4)+70,30);
  OutTextXY(MaxX*3 div 4,20,TextForVideo);

  Str(NumberOfBottles : 5 ,TextForVideo);       { update count of bottles }
  Bar(MaxX*3 div 4,90,(MaxX*3 div 4)+60,100);          { NumberOfBottles }
  OutTextXY(MaxX*3 div 4,90,TextForVideo);

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
                    { Output not Ur dependent but not printed continuously }
                                          { PRINTCOUNT dependent output }
                  { write results to screen and file }
                  { print values if changed state or }
                  { every "PrintIndex" times of h    }
  if (PrintCount >= PrintIndex) or (State <> PreviousState) or
     (Vdone >= Vrequested) then               { print if conditions are right }
     begin
{- - - - - - - - - - - - - - - - - - - - - - - -  SCREEN OUTPUT }
                              { analog update on a less frequent basis }
     VideoVchange(xWidth*15,1,xWidth,yHeight*3,V[Vh],Vhcap);            { Vh }
     VideoVchange(MaxX-xWidth*2,1,xWidth,yHeight*2,Vdone,Vrequested); { Vr }
     VideoVchange(xWidth*2,yHeight*2,xWidth-10,yHeight,Tr,T_Trgtheat);{ Tr }
```

```
                                { digital update on a less frequent basis }
      SetFillStyle(0,0);
      if (State=Fill) or (State=Drain) then
         begin
         SetFillStyle(0,0);
         Str(Tr : 3 : 1,TextForVideo);                    { update digital Tr }
         Bar(xWidth*2-10,yHeight*3+yBoarder*4,
             xWidth*2+40,yHeight*3+yBoarder*4+10);
         OutTextXY(xWidth*2-10,yHeight*3+yBoarder*4,TextForVideo);
         end;

      Str(V[Vh] : 4 : 1,TextForVideo);                    { update digital Vh }
      Bar(xWidth*15-10,yHeight*3+yBoarder*4-2,
          xWidth*15+50,yHeight*3+yBoarder*4+8);
      OutTextXY(xWidth*15-10,yHeight*3+yBoarder*4,TextForVideo);

      Str(Vdone : 4 : 1,TextForVideo);                 { update digital Vdone }
      Bar(MaxX-xWidth*2-10,yHeight*2+yBoarder*4,MaxX-xWidth*2+50
          ,yHeight*2+yBoarder*4+10);
      OutTextXY(MaxX-xWidth*2-10,yHeight*2+yBoarder*4,TextForVideo);

      Str(BottlingRate : 4 : 1,TextForVideo);          { update Bottling Rate }
      Bar(MaxX*3 div 4,50,(MaxX*3 div 4)+60,60);
      OutTextXY(MaxX*3 div 4,50,TextForVideo);

      Str(PercentOverfill : 3 : 1,TextForVideo);       { update Percent Waste }
      Bar(MaxX*3 div 4,70,(MaxX*3 div 4)+50,80);
      OutTextXY(MaxX*3 div 4,70,TextForVideo);

{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  FILE OUTPUT }
      Vingred_used:=round((1000-V[V1])*10+Vinitial);
      Vfinished_prod:=round(Vdone+V[Vr]+V[Vh]+V[Vf]+V[Vb]);

      for Tank:= V1 to Vb do
         begin
            V_[Tank]:= round(V[Tank])      { get printable values of volumes }
         end;
      V_[Vf]:=round(V[Vf]*128);    { get value of Fill tank volume in ounces }

      writeln(out_file,V_[V1],'      ',V_[V2],'      ',V_[V3],'      ',V_[Vr]
         ,'      ',V_[Vh],'      ',V_[Vf],'      ',CumTime,'      ',Vdone);

      PrintCount:=1;               { reset count for time till next printout }
      PreviousState:=State         { reset State to detect change of state   }
      end
   else
{-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  }
                                  { if counter not right then don't print }
                                  { but increment counter }
       PrintCount:=PrintCount+1;
                                     { end of PRINTCOUNT dependent output }

{-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- }
                                                              { OUTPUT end }

close(out_file);                                          { close result file }

GetTimeDiff(Hour1,Minute1,Second1,Sec1001,Hour2,Minute2,Second2,Sec1002);
writeln('Actual run time was ',Hour2,':',Minute2,':',Second2,':',Sec1002);

{************************************************** END FINAL OUTPUT *****}
ch:=ReadKey                          { INPUT statement used to retain final }
                                     { video screen till a key is touched   }
end.
```

APPENDIX 2

DATA POINTS FOR GRAPHS

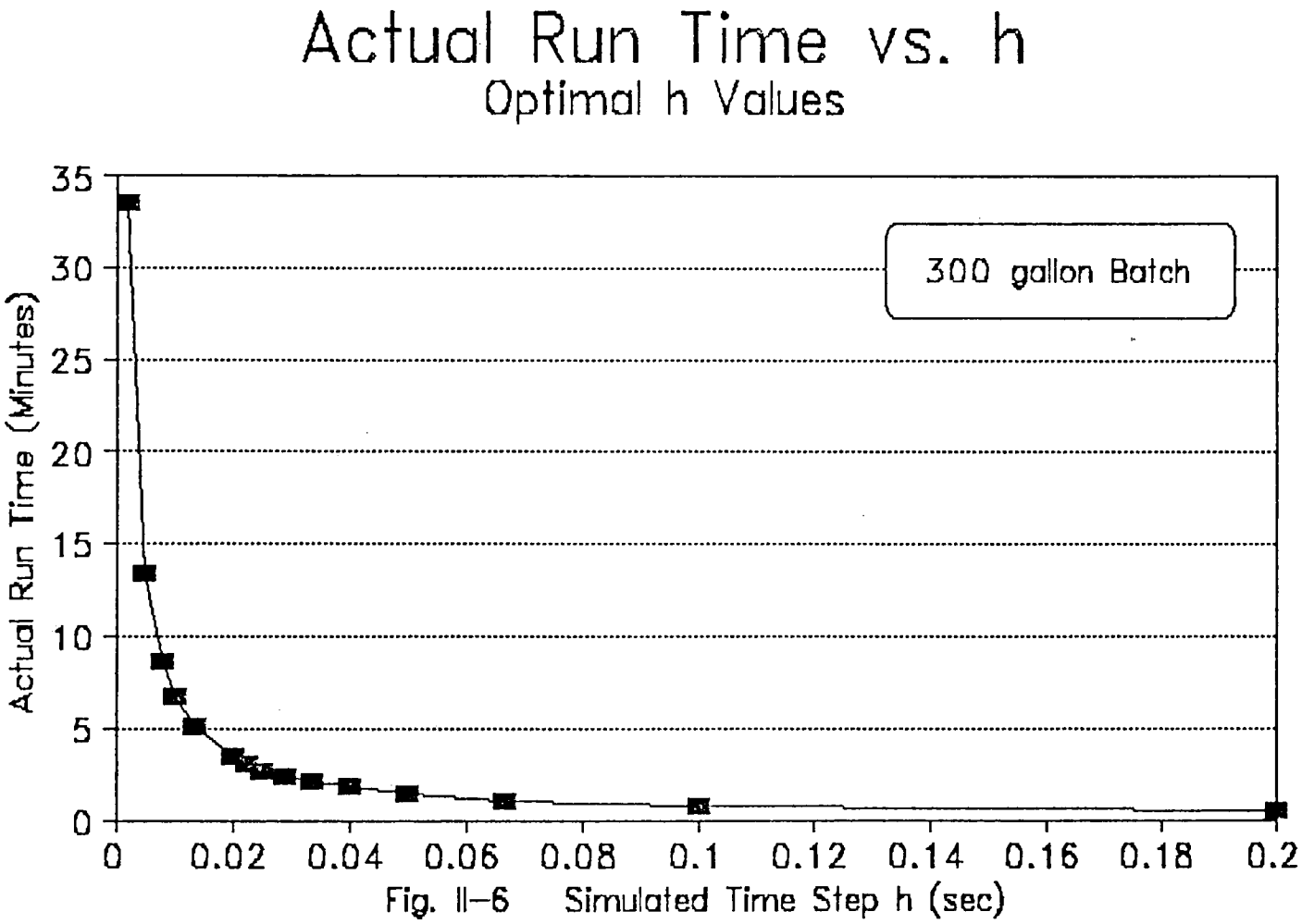

Actual Run Time vs. h
Optimal h Values
300 gallon Batch
Actual Run Time (Minutes)
35
30
25
20
15
10
5
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Simulated Time Step h (sec)

Fig. II–6

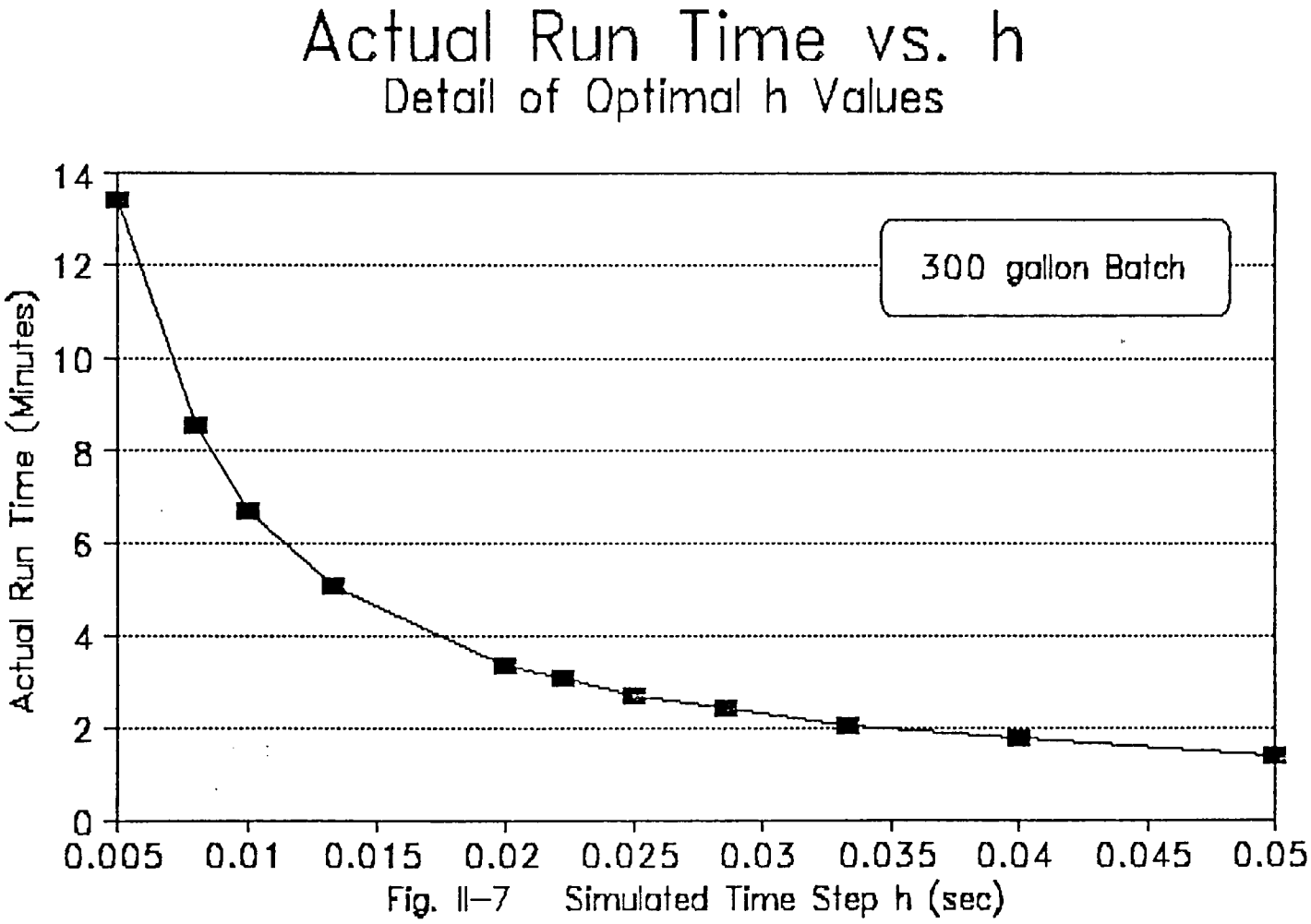

Actual Run Time vs. h
Detail of Optimal h Values
300 gallon Batch
Actual Run Time (Minutes)
14
12
10
8
6
4
2
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
Simulated Time Step h (sec)

Fig. II–7

Actual Run Time vs. h			Cycles	Ounces
h	Run Time seconds	Run Time minutes	per Bottle	per Cycle
0.002	2008.62	33.477	100	0.12
0.005	804.11	13.40183	40	0.3
0.008	513.23	8.553833	25	0.48
0.01	403.6	6.726667	20	0.6
0.01334	303.79	5.063167	15	0.8
0.02	202.7	3.378333	10	1.2
0.022223	183.89	3.064833	9	1.333333
0.025	162.25	2.704167	8	1.5
0.02858	143.58	2.393	7	1.714286
0.033334	123.47	2.057833	6	2
0.04	103.7	1.728333	5	2.4
0.05	81.29	1.354833	4	3
0.0667	65.64	1.094	3	4
0.1	46.69	0.778167	2	6
0.2	30.48	0.508	1	12

Percent Waste & Bottling Rate vs. h