# **CHAPTER 10**

# **Fault Detection**

# Introduction

Fault detection is the process of detecting failures, also known as faults, in a dynamical system. It is an important area for systems that are supposed to operate without human supervision. There are many ways of detecting failures. The simplest is using boolean logic to check against fixed thresholds. For example, you might check an automobile's speed against a speed limit. Other methods include fuzzy logic, parameter estimation, expert systems, statistical analysis, and parity space methods. In this section, we will implement one type of fault detection system, a detection filter. This is based on linear filtering. The detection filter is a state estimator tuned to detect specific failures. We will design a detection filter system for an air turbine. We will also show how to build a graphical user interface (GUI) as a front end to the fault detection simulation.

# 10.1 Modeling an Air Turbine

### Problem

We need to make a numerical model of an air turbine to demonstrate detection filters.

## Solution

Write the equations of motion for an air turbine. We will use a linear model of the air turbine to simplify the detection filter design. This will allow us to model the system with a linear state space model.

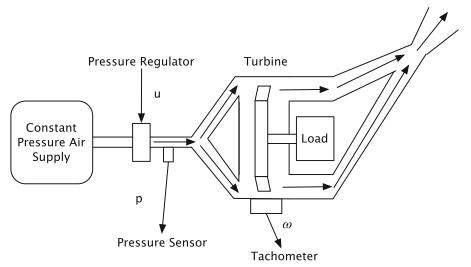


Figure 10.1: Air turbine. The arrows show the airflow. The air flows through the turbine blade tips causing it to turn.

#### How It Works

Figure 10.1 shows an air turbine.<sup>1</sup> It has a constant pressure air supply. We can control the valve from the air supply, the pressure regulator, to control the speed of the turbine. The air flows past the turbine blades causing it to turn. The control needs to adjust the air pressure to handle variations in the load. We measure the air pressure p downstream from the valve, and we also measure the rotational speed of the turbine  $\omega$  with a tachometer.

The dynamical model for the air turbine is

$$\begin{bmatrix} \dot{p} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau_p} & 0 \\ \frac{K_t}{\tau_t} & -\frac{1}{\tau_t} \end{bmatrix} \begin{bmatrix} p \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{K_p}{\tau_p} \\ 0 \end{bmatrix} u$$
(10.1)

This is a state space system:

$$\dot{x} = ax + bu \tag{10.2}$$

where

$$a = \begin{bmatrix} -\frac{1}{\tau_p} & 0\\ \frac{K_t}{\tau_t} & -\frac{1}{\tau_t} \end{bmatrix}$$
(10.3)

$$b = \begin{bmatrix} \frac{K_p}{\tau_p} \\ 0 \end{bmatrix}$$
(10.4)

<sup>&</sup>lt;sup>1</sup>PhD thesis of Jere Schenck Meserole, "Detection Filters for Fault-Tolerant Control of Turbofan Engines," Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 1981.

The state vector is

$$\left[\begin{array}{c}p\\\omega\end{array}\right] \tag{10.5}$$

The pressure downstream from the regulator is equal to  $K_{pu}$  when the system is in equilibrium.  $\tau_p$  is the regulator time constant, and  $\tau_t$  is the turbine time constant. The turbine speed is  $K_{tp}$  when the system is in equilibrium. The tachometer measures  $\omega$ , and the pressure sensor measures p. The load is folded into the time constant for the turbine.

The code for the right-hand side of the dynamical equations is shown in the following. Only one line of code is needed. The rest returns the default data structure. The simplicity of the model is due to its being a state space model. The number of states could be large, yet the code would not change.

RHSAirTurbine.m

```
function xDot = RHSAirTurbine(~, x, d)
27
28
29
  % Default data structure
 if ( nargin < 1 )
30
     kP = 1;
31
     kТ
32
        = 2;
     tauP = 10;
33
     tauT = 40;
34
35
     С
          = eye(2);
36
     b
         = [kP/tauP;0];
         = [-1/tauP 0; kT/tauT -1/tauT];
37
     а
38
     xDot = struct('a',a,'b',b,'c',c,'u',0);
39
40
     if( nargout == 0)
       disp('RHSAirTurbine struct:');
41
     end
42
43
     return
   end
44
```

The response to a step input for u is shown in Figure 10.2. The pressure settles faster than the turbine speed. This is due to the turbine time constant and the lag in the pressure change. The residuals are very small because there are no failures.

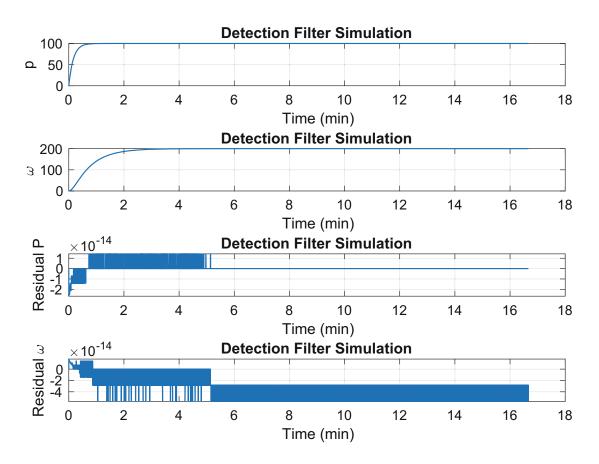


Figure 10.2: Air turbine response to a step pressure regulator input. The residuals are zero as expected.

# 10.2 Building a Detection Filter

#### Problem

We want to build a system to detect failures in our air turbine using the linear model developed in the previous recipe.

#### Solution

We will build a detection filter that detects pressure regulator failures and tachometer failures. Our plant model (continuous a, b, and c state space matrices) will be an input to the filter building function.

#### How It Works

The detection filter is an estimator with a specific gain matrix that multiplies the residuals. The residuals are the difference between the estimated outputs and the outputs:

$$\begin{bmatrix} \dot{\hat{p}} \\ \dot{\hat{\omega}} \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau_p} & 0 \\ \frac{K_t}{\tau_t} & -\frac{1}{\tau_t} \end{bmatrix} \begin{bmatrix} \hat{p} \\ \hat{\omega} \end{bmatrix} + \begin{bmatrix} \frac{K_p}{\tau_p} \\ 0 \end{bmatrix} u + \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} p - \hat{p} \\ \omega - \hat{\omega} \end{bmatrix}$$
(10.6)

where  $\hat{p}$  is the estimated pressure and  $\hat{\omega}$  is the estimated angular rate of the turbine. The *D* matrix is the matrix of detection filter gains. These feedback the residuals, the difference between the measured and estimated states, into the detection filter. The residual vector is

$$r = \begin{bmatrix} p - \hat{p} \\ \omega - \hat{\omega} \end{bmatrix}$$
(10.7)

The residuals are the difference between the measured values and the estimated values. The D matrix needs to be selected so that this vector tells us the nature of the failure. The gains should be selected so that

- 1. The filter is stable.
- 2. If the pressure regulator fails, the first residual  $p \hat{p}$  is nonzero, but the second remains zero.
- 3. If the turbine fails, the second residual  $\omega \hat{\omega}$  is nonzero, but the first remains zero.

A gain matrix is

$$D = a + \begin{bmatrix} \frac{1}{\tau_1} & 0\\ 0 & \frac{1}{\tau_2} \end{bmatrix}$$
(10.8)

The time constant  $\tau_1$  is the pressure residual time constant. The time constant  $\tau_2$  is the tachometer residual time constant. In effect, we cancel out the dynamics of the plant and replace them with decoupled detection filter dynamics. These time constants should be shorter than the time constants in the dynamical model so that we detect failures quickly. However, they need to be at least twice as long as the sampling period to prevent numerical instabilities.

We will write a function with three actions, an initialize case, an update case, and a reset case. varargin is used to allow the three cases to have different input lists. The function signature is

#### DetectionFilter.m

funct	ion d = D	DetectionFilter(	action,	varargin )
-------	-----------	------------------	---------	------------

The header and syntax for DetectionFilter are shown as follows. We used LaTeX equations to describe the function.

```
1 %% DETECTIONFILTER Builds and updates a linear detection filter.
  2 %% Forms
       % d = DetectionFilter( 'initialize', d, tau, dT )
  3
      % d = DetectionFilter( 'update', u, y, d )
  4
      % d = DetectionFilter( 'reset', d )
  5
       8
  6
 7 %% Description
  8 % The detection filter gain matrix d is designed during the initialize
  9 % action. The continuous matrices are then discretized using the
                   internal
      % function CToDZOH. The esimated state and residual vectors are
10
                   initialized
       % to the size dictated by a. During the update action, the residuals
11
                  and
     % new estimated state are calculated and stored in the data structure d
12
       %
13
14 % The residuals calculation is
15 😵
       16
       Ŷ
17
       % The estimated state calculated with the detection filter gains is
18
       2
19
20
        \frac{1}{x} \frac{1}{x} = \frac{1}{x} + \frac{1}{
      %
21
22 %% Inputs
                                                     (1,:) 'initialize' or 'update'
23
       % action
24 😵 d
                                                       (.) Data structure
25 😽
                                                                           .a (:,:) State space continuous a matrix
26 😽
                                                                            .b (:,1) State space continuous b matrix
                                                                            .c (:,:) State space continuous c matrix
       %
27
28 😵 tau
                                                     (:,1) Vector of time constants
29 % dT
                                                      (1,1) Time step
       % u
                                                       (:,1) Actuation input
30
      %
               У
                                                      (:,1) Measurement vector
31
32 😽
33 %% Outputs
                                                       (.)
34
       %
                   d
                                                                        Updated data structure
35 😵
                                                                            .a (:,:) State space discrete a matrix
36 😵
                                                                            .b (:,1) State space discrete b matrix
37 😽
                                                                             .c (:,:) State space discrete c matrix
38 😽
                                                                            .d (:,:) Detection filter gain matrix
39 😽
                                                                            .x (:,1) Estimated states
40 😽
                                                                            .r (:,1) Residual vector
```

The filter is built and initialized in the following code in DetectionFilter. The continuous state space model of the plant, in this case, our linear air turbine model, is an input. The selected time constants  $\tau$  are also an input, and they are added to the plant model as in Equation 10.8. The function discretizes the plant a and b matrices and the computed detection filter gain matrix d.

```
48
   function d = DetectionFilter( action, varargin )
49
50
   switch lower (action)
51
     case 'initialize'
52
       d
           = varargin\{1\};
53
       tau = varargin{2};
54
       dT = varargin{3};
55
56
       % Design the detection filter
57
58
       d.d = d.a + diag(1./tau);
59
       % Discretize both
60
                  = CToDZOH(d.d, d.b, dT);
61
       d.d
        [d.a, d.b] = CTODZOH(d.a, d.b, dT);
62
63
       % Initialize the state
64
       d.x = zeros(m, 1);
66
       d.r = zeros(m, 1);
67
```

The update for the detection filter is in the same function, as the next action in the switch statement. Note the equations implemented as described in the header.

```
case 'update'
69
70
         u
              = varargin{1};
              = varargin{2};
71
         У
         d
              = varargin{3};
72
73
         r
              = y - d.c \star d.x;
         d.x = d.a \star d.x + + d.b \star u + d.d \star r;
74
75
         d.r = r;
```

Finally, we create a reset action to allow us to reset the residual and state values for the filter in between simulations. After this action, we end the switch statement.

```
77  case 'reset'
78  d = varargin{1};
79  m = size(d.a,1);
80  d.x = zeros(m,1);
81  d.r = zeros(m,1);
82  end
```

# 10.3 Simulating the Fault Detection System

#### Problem

We want to simulate a failure in the plant and demonstrate the performance of the failure detection.

#### Solution

We will build a MATLAB script that designs the detection filter using the function from the previous recipe and then simulates it with a user selectable pressure regulator or tachometer failure. The failure can be total or partial.

#### **How It Works**

The script designs a detection filter using DetectionFilter from the previous recipe and implements it in a loop. Runge-Kutta integration propagates the continuous domain right-hand side of the air turbine, RHSAirTurbine. The detection filter is discrete time.

The script has two scale factors uF and tachF that multiply the regulator input and the tachometer output to simulate failures. Setting a scale factor to zero is a total failure, and setting it to one indicates that the device is working perfectly. If we fail one, we expect the associated residual to be nonzero and the other to stay at zero. Failures can be any number between zero and one. Partial failures are not necessarily related to a specific mechanical failure but are useful for testing the system.

#### DetectionFilterSim.m

```
%% Simulation of a detection filter
1
  % Simulates detecting failures of an air turbine.
2
3
  % An air turbine has a constant pressure air source that sends air
  % through a duct that drives the turbine blades. The turbine is
4
  % attached to a load.
5
  %
6
  % The air turbine model is linear. Failures are modeled by multiplying
7
  % the regulator input and tachometer output by a constant. A constant
8
  % of 0 is a total failure and 1 is perfect operation.
9
14
  %% User inputs
15
16
  % Failures. Set to any number betweem 0 and 1 is 0 is total failure. 1
17
      is working perfectly.
  % uF scales the actuation u. tachF scales the rate measurement.
18
19
  uF
       = 0;
  tachF = 1;
20
21
22 % Time constants for failure detection
23 tau1 = 0.3; % sec
```

24 tau2 = 0.3; % sec

```
25
26 % End time
  tEnd = 1000; % sec
27
28
29 % State space system
30 d = RHSAirTurbine;
31
32 %% Initialization
dT = 0.02;  sec
34 n = ceil(tEnd/dT);
35
36 % Initial state
37 X = [0;0];
38
39 %% Detection Filter design
40 dF = DetectionFilter('initialize',d,[tau1;tau2],dT);
41
42 %% Run the simulation
43
44 % Control. This is the regulator input.
45 u = 100;
46
47 % Plotting array
48 xP = zeros(4, n);
49 t = (0:n-1) * dT;
50
51 for k = 1:n
   % Measurement vector including measurement failure
52
   y = [x(1);tachF*x(2)]; % Sensor failure
53
    xP(:,k) = [x;dF.r];
54
55
    % Update the detection filter
56
    dF = DetectionFilter('update',u,y,dF);
57
58
    % Integrate one step
59
    d.u = uF*u; % Actuator failure
60
    x = RungeKutta(@RHSAirTurbine, t(k), x, dT, d);
61
62
  end
63
64 %% Plot the states and residuals
65 [t,tL] = TimeLabel(t);
       = {'p' '\omega' 'Residual P' 'Residual \omega' };
66 YL
67 tTL
         = 'Detection Filter Simulation';
68 PlotSet( t, xP,'x label',tL,'y label',yL,'plot title',tTL,'figure title
      ',tTL)
```

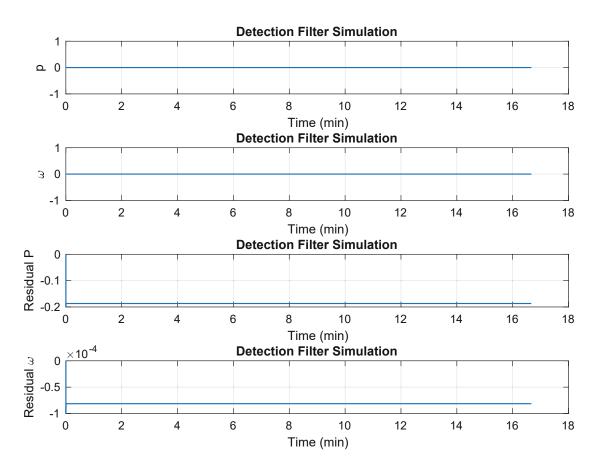


Figure 10.3: Air turbine response to a failed regulator.

In Figure 10.3, the regulator fails and its residual is nonzero. In Figure 10.4, the tachometer fails and its residual is nonzero. The residuals show what has failed clearly. Simple boolean logic (i.e., if end statements) are all that is needed.

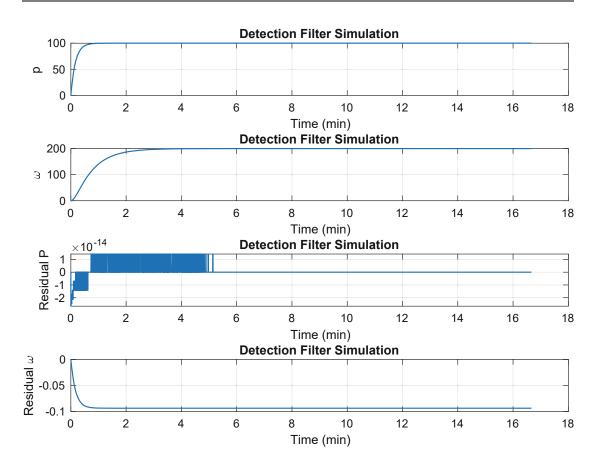


Figure 10.4: Air turbine response to a failed tachometer.

# 10.4 Building a GUI for the Detection Filter Simulation

#### Problem

We want a GUI to provide a graphical interface to the fault detection simulation that will allow us to evaluate the filter's performance.

#### Solution

We will use the MATLAB App Designer to build a GUI that will allow us to

- 1. Set the residual time constants
- 2. Set the end time for the simulation

- 3. Set the pressure regulator input
- 4. Introduce a pressure regulator or tachometer fault at any time
- 5. Display the states and residuals in a plot

#### How It Works

The MATLAB App Designer is invoked by typing appdesigner at the command line. There are several options for GUI templates, or a blank GUI; we will start from the GUI with uicontrols. First, let's make a list of the controls we will need from our desired features list earlier:

- Edit boxes for the simulation duration, residual time constants  $\tau_1$  and  $\tau_2$ , pressure regulator setting u
- Edit boxes for the pressure regulator and tachometer fault parameters, with buttons for sending the newly commanded values to the simulation
- Text box for displaying the calculated detection filter gains
- Run button for starting a simulation
- Two plot axes

In order to change the fault parameters while the simulation is running, we will need the loop to be checking a variable that can be externally set by the GUI. We can do this using global variables.

There are several templates that we can use. We will start with the basic blank template. Type appdesigner in the command window. Figure 10.5 shows the interface.

Double-click the blank app template.

Add the app DFGUI. It will appear in your folder as DFGUI.mlapp. Add the following to the blank template:

#### 1. Parameter input boxes

- (a) Duration
- (b) Input
- (c) Tau 1
- (d) Tau 2
- (e) Gains (2-by-2 matrix)

#### CHAPTER 10 ■ FAULT DETECTION

ATLAB APP D				
And	Want to embed your ov	vn HTML content? Use	the HTML UI componer	nt. Learn More
Dpen	Getting Started   GUIDE Migration Strate	gies   Displaying Graphics in App Designer	Release Notes	
cent Apps	- New			
Mortgage				
	Blank App	2-Panel App with Auto-Reflow	3-Panel App with Auto-Reflow	
	- Examples: General			
	Interactive Tutorial	Respond to Numerical Input	Respond to User Selections	Embed HTML Content
	Lay Out Controls in a Grid			
	Examples: Programming Tasks			
	$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$			
	Link Data to a Tree	Analyze an Image	Configure a Timer	Display Specialized Axes
	Create a Table	Query Website Data	Pass Data Between Apps	

Figure 10.5: The interface to appdesigner.

- 2. Failure input boxes
  - (a) Tachometer
  - (b) Input
  - (c) Send button for tachometer
  - (d) Send button for input

	p Designer - app1.mlapp
NER CANVAS	
W Open Save App Share Run PLE SHARE RUN of mapp x	
OMPONENT LIBRARY	Design View Code View C
arch P 🔳 🐯	
MMON	
Axes Button Check Box	
30 to to the test of t	
iate (an Enge In Field HTML Image Text)	
A total abel List Box Rado Button Group	
1 2 0 3 Press idder Spinner State Button	
Text Area Toggle Button Group	
0 - 0 Tree	
ITAINERS	
d Layout Panel Tab Group	
RE TOOLS	

Figure 10.6: Snapshot of the blank app.

- 3. Calculate button
- 4. Reset button
- 5. State plot
- 6. Residual plot

You add items by dragging and dropping them on the window from the items on the lefthand side. We use numeric for the input text boxes. Figure 10.7 shows the completed interface. There are four push buttons.

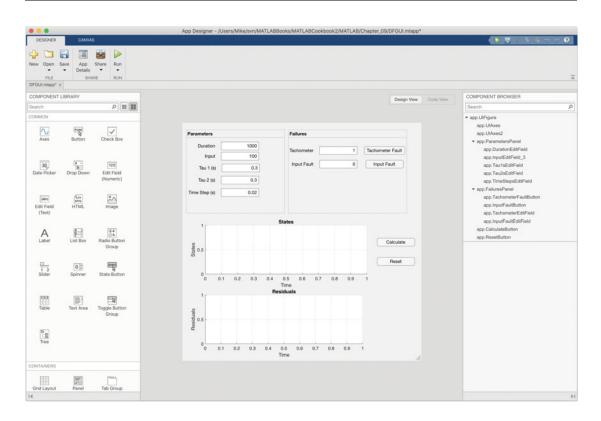


Figure 10.7: Snapshot of the app after the interface is done.

You can add information about the app. Figure 10.8 shows the window for app information. The app appears in the app menu as shown in Figure 10.9.

We select the callback for calculate. The App Designer highlights where the code should go. We copy relevant code from the simulation script. We get the inputs from the text boxes.

DESIGNER CANVAS	App Designer - /Users/Mike/svn/MATLABBooks/MATLABCookbook2/MATLAB/Chapter	_oaroroor.mapp	
PLE SHARE RUN COLINIEQO' IN			
OMPONENT LIBRARY		Design Wew Code View	COMPONENT BROWSER
arch 2 (= 12		Design York Color York	Search
MAON	App Details	×	
Axes Button Check Box	Paramete Name Duratic DFGUI 1.0		app.UIAxes app.UIAxes2 • app.ParametersPanel app.DurationsEdtField
ac Picker Drop Down Edit Field (Numeric)	Author Te Summary Detection Filter GUID App screen shot is captured		app.tnputEditField app.Tau1sEditField app.Tau2sEditField app.GainDEditField • app.FaluresFanel
and Arg 200 Str Field HTML Image (Text)	Gae		app.TachorneterButton app.InputButton app.TachometerEdtField app.InputEdtField_2
A Label List Box Radio Button Group	Provides a GUI for the detection filter script.		app CalculateButton app ResettButton
Ti 2 Sider Spinner State Button	App details display in certain situations, such as when you share your app or v your app in some system file browsers.	ew.	
Table Text Area Toggle Button Group	> 0.5 OK Can	201	
Tree	0 0 02 04 0.6 0.8 1		
(TAINERS			
rid Layout Panel Tab Group			

Figure 10.8: App Details let you add information about the app for users.

Figure 10.10 shows the code. You access parameters from the text boxes using app.xxx.Value. For all plot-related functions, you need to add the axes handle using app.UAxes.or app.UAxe2s.



Figure 10.9: The app appears in the app menu. You get to this window by hitting the design app button.

Figure 10.11 shows a debugger breakpoint. You have full access to the debugger in App Designer. You will also see MATLAB warnings on the right.

Figure 10.12 shows the app after a run.

And the set of the se	nt % % % Internation to the second se	<pre>     Enable app coding alers Show Tips and Very Resources And Ver</pre>	Starch app.Ulr/gure app.Ulr/gure app.Ulr/gure app.Ulr/acs2 app.Il/acs2 app.TarametersPanel app.Invariational app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel app.TarametersPanel	He Q
	EDT VEW	Run Despriver Le: d, [taul;tau2],dT);	COMPONENT BROWSER Search	
	<pre>Button pushed function: CalculateB inction CalculateButtonPushed(app. % get the data from the app duration = app.InputEdifField_3.V duration = app.InputEdifField_3.V atchF = app.Insuffield[Field_3.U app.TaulsEdifField_4.Val app.TaulsEdifField_Val fault = app.TaulsEdifField_Val % Set up the model % Set up the model % Plotting array n = ceil(duration/dT); x = [0:01.)*dT; x = [0:01.)*dT; x = [0:01.)*dT; x = [0:01.00.000.000.000.000.000.000.000.000.</pre>	Deagn Vere de: de: s; d, [tau1;tau2],dT);	COMPONENT BROWSER Search app UNAxes app UNAXES	
	Button pushed function: CalculateB metrion CalcutateButtonPushed(app. * get the data from the app u duration = app.InputEditField_3.V tachF = app.InputFaultEditField. tachF = app.InputFaultEditField. TachmeterEditField. TachmeterEditField. * Set up the model * Set up the model df = BHSAirTurbine; df = DetectionFilter('initi * Plotting array * Plotting array	.d,[tau1;tau2],dT);	<ul> <li>Search</li> <li>app UlAves</li> <li>app UlAves2</li> <li>app UlAves2</li> <li>app LutorEdiffield.3</li> <li>app.TurusStepsEdiffield</li> <li>app.TurusStepsEdiffield</li> <li>app.TautStepsEdiffield</li> <li>app.TachometerFaulBurapp. Input"</li> </ul>	
	Button pushed function: CalculateB metion CalculateButtonPushed(app. * get the data from the app u duration = app.InputEditField_3.V derf = app.InputEditField_3.V tach = app.InputEditField. T = app.InputEditField. T = app.TaulsEditField.Val tau2 = app.TaulsEditField.Val tau2 = app.TaulsEditField.Val tau2 = app.TaulsEditField.Val f = BHSAirTurbine; d = BHSAirTurbine; d = BHSAirTurbine; t = ceild(duration/dT); x = ceis(duration/dT); x = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d = (0	rtton vent) vent) alue; .Value; Value; Value; e; e; e; hilze',d,[tau1;tau2],dT);	<ul> <li>app. UlFigure</li> <li>app. UlAves</li> <li>app. UlAves2</li> <li>app. UlAves2</li> <li>app. UnationEctifield</li> <li>app. InterdineEdiField</li> <li>app. TranslepsEdifield</li> <li>app. TranslepsEdifield</li> <li>app. TranslepsEdifield</li> <li>app. TranslepsEdifield</li> <li>app. TranslepsEdifield</li> <li>app. TranslepsEdifield</li> </ul>	Atton
	<pre>% get the data from the app duration = app.InputEditField.3v duration = app.InputEditField.3v tachf = app.InputEditField. dT = app.InputFaultEditField. dT = app.InputEditField.val tau2 = app.TauSEditField.val as = app.TauSEditField.val d = RHSAirTurbine; d = RHSAirTurbine; d = RHSAirTurbine; d = RHSAirTurbine; d = ceild(duration/dT); x = ceis(duration/dT); x = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT; d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d = (0:-1)*dT;d</pre>	vent) Nue; alue; L'Value; Value; Value; e; e; hlze',d,[taul;tau2],dT);	app. UlAves app. UlAves2 app. UlAves2 • app. ParametersPanel app. IndurationEditField app. Trait.EditField app. Trait.editField app. Trait.messpeEditField app. Trait.meretEditField app. Trait.meretEditField app. InputFaultEditField	a Atton
	ation ation 11 22 et up t et up t	llue; alue; "Value; Value; Value; e; e; hize',d,[tau1;tau2],dT);	app. UIAves2 • app ParametersPanel app. DinutionEditField. app. InturtEditField. app. Trau2sEditField app. Trau2sEditField app. TranometerFaultBuu app. TrachometerEditField app. TachometerEditField app. TachometerEditField app. InputFaultEditField	a Atton
	ation :hF 11 22 et up t et up t	alue; .Value; .Value; Value; value; e; e; ilize',d,[tau1;tau2],dT);	<ul> <li>app ParametersPanel app. DurationEdiField.3 app. IntrationEdiField.3 app. TautSEdiField app. TimeStopsEdiField</li> <li>app. TautSetopsEdiField</li> <li>app. TarbometerFaultBuu app. IncorrecterEdiField app. TarbometerEdiField</li> </ul>	d tho
	:hF 12 iet up t iotting	.Value; Value; Value; e; e; hlize',d,[tau1;tau2],dT);	api	d tton
	1 12 set up t lotting	ryaue; Value; e; ilize',d,[tau1;tau2],dT);	ide	d fiton
	1 set up t lotting	e; e; llize',d,[tau1;tau2],dT);	ide	tton
	iet up t iet up t	e; ilize',d,[tau1;tau2],dT);	ide	tton d
	et up t lotting	llize',d,[tau1;tau2],dT);	ide	utton kd
	lotting	<pre>ilize',d,[tau1;tau2],dT);</pre>	app.TachometerFaultBut app.InputFaultButton app.TachometerEditFiek app.tnputFaultEditFiekd	utton
	lotting	<pre>ilize',d,[tau1;tau2],dT);</pre>	app.InputFaultButton app.TachometerEditField app.InputFaultEditField	P
	lotting		app.TachometerEditField app.InputFaultEditField	p
			app.InputFaultEditField	
			app.CalculateButton app.ResetButton	
			Inspector Calibacks	
	601 k = 1.0		Search	D ::: 0
	% Measurement vector including measurement failure	asurement failure		
mon         mon	<pre>y == [x(1);tachF*x(2)]; % Sensor failure</pre>	nsor failure	• III F	[
	<pre>xP(:,k) = [x;dF.r];</pre>		Title Parameters	beters
	% Undate the detection filter		TitlePosition	
	<pre>dF = DetectionFilter('update',u,y,dF);</pre>	',u,y,dF);	<ul> <li>COLOR AND STYLING</li> </ul>	
la contra c	% Interrate one ster		ForegroundColor 0.00,0.	• 00'0'00'0'00'0
(perese)	d.u = uF*u; % Actuator failure	e	BackgroundColor 0.94,0.	0.94,0.94,0.94
	H	RungeKutta( @RHSAirTurbine, t(k), x, dT, d );	BorderType line	•
	end		+ FONT	
The second se	<pre>[t,tL] = TimeLabel(t);</pre>		FontName	ica 🔹
3 0	% Plot the states and residuals			
1	axes(app.UIAxes)		FontWeight	]
	<pre>plot(app.UIAxes,t, xP(1:2,:) ) vlahel(app.UIAves +!).</pre>			
	<pre>legend(app.UIAxes,'p','\omega')</pre>		Printer manual	
	10 - 111 - J			
	axes(app.ULAxes2) n]nt(ann IITAxes2 + xp(3.4 .) )		POSITION	
	xlabel(app.UIAxes2,tL);		<ul> <li>CALLBACK EXECUTION CONTROL</li> </ul>	ſ
	<pre>legend(app.UIAxes2,'r_p','r_{\omega}')</pre>	(.{	PARENT / CHILD	
	T		<ul> <li>IDFNTIFIFRS</li> </ul>	

Figure 10.10: The light area is where the code goes. The code is from the simulation script.

#### CHAPTER 10 ■ FAULT DETECTION

•		App Designer - /Users/Mike/svn/MATLABBooks/MATLABCookbook2/MATLAB/Chapter_09/DFGUI.mlapp	
DESIGNER			
Save         Collect         Function         Property         App Input	Go To •	Comment % %     Comment % %     Indent % %     Evable app coding alerts     Show Tips     Continue Step 4 Step Out     Out Debugging	
FILE INSERT	NAVIGATE	E EDIT VIEW RESOURCES DEBUG	14
<ul> <li>CODE BROWSER</li> </ul>			COMPONENT BROWSER
Callbacks   Functions   Properties			Search
Search P	1 1	app.inputSwitch.Position = [215 65 45 20];	◆ app.UlFigure
Add a caliback function to make your app		% Create CalculateButton	app.UIAxes
respond to user minimizations such as outlots clicks,	1	<pre>app.CalculateButton = uibutton(app.UIFigure, 'push');</pre>	app.UIAxes2
	1 1	<pre>app.CalculateButton.Position = [514 303 100 22]; app.CalculateButton.Text = 'Calculate';</pre>	app.DurationsEditField
			app.InputEditField
		% Create ResetButton	app.Tau1sEditField
	L I	appresetention - substant approximately provided approximately approxima	app.1 auzsEditField app.GainDEditField
	t.	<pre>app.ResetButton.Text = 'Reset';</pre>	<ul> <li>app.FailuresPanel</li> </ul>
		% Show the figure after all components are created	app.TachometerEditField
	ı.	app.UIFigure.Visible = 'on';	app.InputEditField_2
	1	end	app. Lacrometer Switch app.InputSwitch
		A but seeded and defined and the	app.CalculateButton
	-0	% App creation and detetion methods (Access = public)	app.ResetButton
APP LAYOUT			Inspector   Calibacks
Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passing Passin		% Construct app function app = DFGUI	
And the second s	ţ	<pre>% Create UIFigure and components createComponents(app)</pre>	
lee Pr	•	<pre>% Register the app with App Designer registerApp(app, app.UlFigure)</pre>	
- 10 - 10 - 10 - 10	•	if nargout == 0	
a si	1.1	clear app end	
	i.	end	
		% Code that executes before app deletion function delete(app)	
	I.	<pre>% Delete UIFigure when app is deleted delete(app.UIFigure)</pre>	
11	1	end	
	1	C10	

Figure 10.11: You can use the debugger in the App Designer.

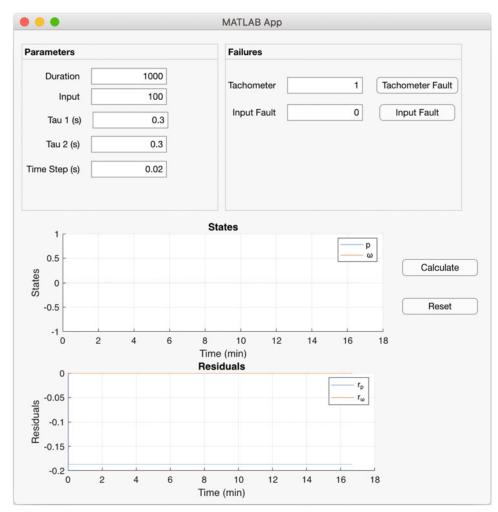


Figure 10.12: The app after a run.

# Summary

This chapter has demonstrated how to design a detection filter for detecting faults in a dynamical system. The system is demonstrated with an air turbine that can experience a pressure regulator failure and a tachometer failure. In addition, we used App Designer to design a GUI to automate filter simulations. The GUI demonstrates real-time plotting and injecting failures into an ongoing simulation loop. Table 10.1 lists the code developed in the chapter.

File	Description
RHSAirTurbine	Air turbine dynamical model in continuous state space form
DetectionFilter	Builds and updates a linear detection filter
DetectionFilterSim	Simulation of a detection filter
DetectionFilterGUI	Run the detection filter simulation from a GUI
DFGUI.m1App	App Designer app
DFGUI.mlappinstall	DFGUI app installer

Table 10.1: Chapter Code Listing