Delft University of Technology

Fault Diagnosis and Fault Tolerant Control
SC42130

Final Report
Group 14

Authors:

Jorge Bonekamp 4474554
Gerardo Moyers 4820800
Casper Spronk 4369475

January 23, 2019

Contents

1 Homework 1

1.1 Components and services

Obviously the main service Waymo offers is an autonomous car, or what they call themselves a ‘fully self-driving’ vehicle. The Waymo car can move without any input from the passenger(s). In order to be able to do this the car has to be aware of its surroundings. This is done using a combination of components, namely: LIDAR sensor, Camera sensor, Radar, GPS and audio. The ‘Self-driving software’ of Waymo is said to perceive, predict and subsequently plan the safe movement (by controlling throttle and steering wheel) of the car in the right direction in real-time. Waymo is claiming to guarantee safe travel by extensive testing of five distinct types of safety that are relevant when on the road. Additionally Waymo claims to have solved the ‘hand-off’ problem (of the steering wheel) that occurs in less advanced self-driving cars, when the car does not know what to do in some situation. Also redundant safety-critical systems are present in the car, and furthermore a process is in place that can deal with cyber security threats.

1.2 Approaches to fault tolerance

One of the primary ways that Waymo is approaching a fault tolerant design is by extensive testing and simulating of the control inputs based on the data coming in from the sensors for certain scenarios. In this way they create ’self-learning’ software to the car. New updates to the software will filter out bugs or misinterpretation of data that could occur in the software. These new software releases are critically validated by Waymo to ensure that the cars are able to drive around safely.

Another way that this car could fail is by a failure of the critical driving features, i.e. throttle, steering, braking etc. But as mentioned above for these critical systems redundant systems are built in. These system kick in when one of the main systems fails and in this way some form of fault tolerant control is implemented.

Another way that the vehicle can fail is by one of its safety critical features failing. The way that Waymo has tried to implement fault tolerance to this potential fault is again by adding redundancy for these systems. By doing this the systems will be online even if one of the safety critical features has failed and bring the car to a safe stop.

1.3 Convincing fault tolerant vehicles

The way in which Waymo ensures that the self-driving vehicle is fault tolerant is by explaining the software and hardware redundancies, and how the vehicle react in case a fault or a failure happened.

- Hardware redundancy

  - Waymo has a backup computer in case a fault or a failure is detected in the main computer. In case of a failure in the first computer, the secondary one is completely capable to control all the systems.

- Software redundancy
  - Combination of functional analysis, simulation tools and on-road driving testing to understand any possible situation and react efficiently.
  - Waymo's software does not just detect the objects, it predicts theirs behavior and react against it before it happened.
  - Have an accurate differentiation between objects, pedestrians, cyclists, motorcyclists and cars. With these the car can efficiently react in the case of a sudden change of the external objects.
  - Detect changes in the environment that affect the driving and determine if it has to stop for security.

Taking into the equation the possibility of *hazard* scenarios Waymo used various hazard assesment method such as preliminary hazard analysis, fault tree, and Design Failure Modes and Effect Analyses (DFMEA). An other approach is by doing extensive testing and simulation of the vehicles. Waymo uses simulations tests, on road tests such as testing on public road, crash avoidance capabilities, and hardware reliability and durability tests.

We conclude Waymo is convincing in saying their vehicle is fault tolerant towards the services the car provides. The Waymo car can only ensure acceptable low risk in extensively tested and validated scenarios. Still there will always be some risk that their car is not fully fault tolerant, since there may always be some scenario for which the software or redundancy systems have not accounted for.

2 Homework 2

2.1 Fault Tree Analysis

A fault tree analysis is a graphical way to show if failure of a particular component can lead to failure of the whole system. The main components to be included in the FTA are the following: five groups of sensor types, the controller, and power supply. The sensors are divided in lasers, camera sets, radar, audio sensors, and GPS. For all these sensors Waymo has implemented redundant sensors, which are also included in the fault tree analysis graph. Except for the analytical redundancy of the GPS-sensor using inertial sensors, these inertial sensors are not included in the FTA. Furthermore, Waymo indicates in its paper the existence of backup for the power supply, the main controller, steering and braking. All these redundancies are relevant for the analysis of when a component failure leads to system failure, and are therefore included in the FTA. The interconnection between the defined components was found in the Waymo report and shown in the FTA using logic gates. It is important to note that we are not considering the car's steering wheel and the brakes as these two components do not have an influence on the self driving nature of the car. If we were analyzing the whole vehicle and not just the self driving system they would have been considered in the analysis. So the root in the FTA-graph, 'system failure', resembles a failing input to the mechanical subsystems of the car-system as a whole.

For the fault tree analysis the main components are defined with the following abbreviations:

- Sensor components
  - LiDIAR short range (L_s1)
  - LiDIAR short range backup (L_s2)
  - LiDIAR medium range(L_m1)
  - LiDIAR medium range backup (L_m2)
  - LiDIAR long range (L_l1)
  - LiDIAR long range backup (L_l2)
  - Audio detection (AD1)
  - Audio detection backup (AD2)
  - Global Positioning System (GPS1)
  - Global Positioning System backup (GPS2)
  - Radar (R1)
  - Radar backup (R2)
  - Camera Set (CS1)
  - Camera Set backup (CS2)
- Power Supply (PS1)
- Power Supply backup (PS2)
- Electronic Control Unit (ECU1)
- Electronic Control Unit backup (ECU2)

Leading to the following graph.

Using this FTA graph we can determine the disjoint normal form and determine the minimal cut sets:

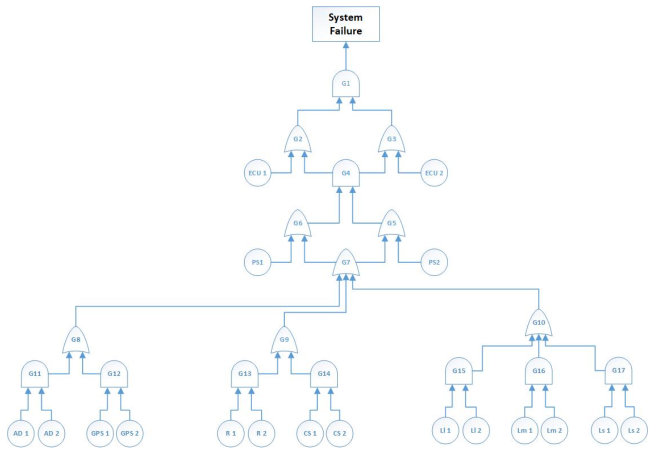


Figure 1: FTA

- Disjoint normal form.
  - G1=(ECU1 $\wedge ECU2)\vee((PS1\wedge PS2)\vee(((L_l1\wedge L_l2)\vee(L_m1\wedge L_m2)\vee(L_s1\wedge L_s2))\vee((R1\wedge R2)\vee(CS1\wedge CS2))\vee((AD1\wedge AD2)\vee(GPS1\wedge GPS2))))$
- Minimal Cut Sets (MCS)
  1. $\{PS1, PS2\}$
  2. $\{ECU1, ECU2\}$
  3. $\{AD1, AD2\}$
  4. $\{GPS1, GPS2\}$
  5. $\{R1, R2\}$
  6. $\{CS1, CS2\}$
  7. $\{L_l1, L_l2\}$
  8. $\{L_m1, L_m2\}$
  9. $\{L_s1, L_s2\}$

As we can observe from the graph and the minimal cut sets the system is robust and for a failure to happen a lot of components and its backups have to fail at the same time, due to all the redundancies built into the system.

2.2 Failure Mode and Effect Analysis

For the Failure mode and effect analysis (FMEA) the following set of components were used to give a (simplified) visual representation of the Waymo failure modes and their behaviour. The first level of components includes only the power supply for the systems, from the graph it can be seen that a failure in power supply affects all subsystems. Similar to this kind of essential subsystem you might name ‘Communication’ as a first level component, but this is not included in the FMEA graphical representation. The second level of analysis contains all sensor components. Note that for both the power supply and the sensors the backup or redundant systems are not shown. So when a failure in one subsystem occurs it is assumed that both the main and the backup or redundant subsystem have that same failure. The last level of components are ‘actuating’ components like steering and braking, since these may have similar failure modes, for simplicity, they are taken as one subsystem. Using these defined components we come to figure 2 representation for the FMEA with the following boolean matrices.

$$M_{audio} = M_{GPS} = M_{Radar} = M_{Camera} = M_{Lidar} = \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix}$$

$$M_{ECU} = \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix} \qquad M_{Actuators} \begin{vmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$

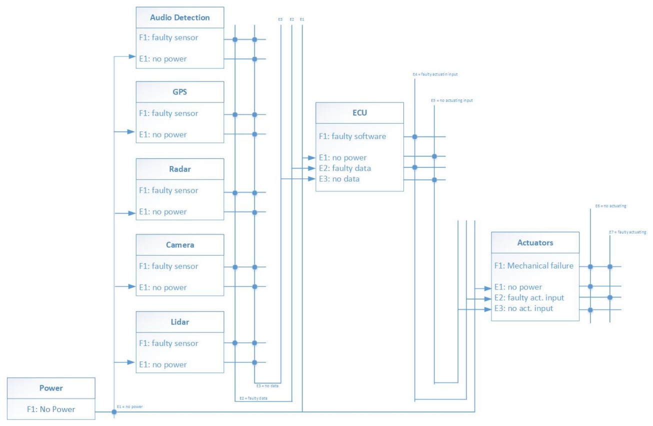


Figure 2: FMEA

From this FMEA graph we can see what failures may occur and how they propagate. Looking at the level of subsystems with all the sensors, we notice that they may have the failure modes named ’faulty sensor’ this would cause the fault effects of no data transmission or faulty data transmission towards the ECU. Also when the sensors have no power as input effect, this failure mode will also propagate as the ’no data transmission’ effect. In the next level the Electrical Control Unit (ECU) may be inherently fault (e.g. calculate wrong faulty actuating inputs based on true data). Or the ECU may propagate effects of failing sensors or no power, into the effects of ’faulty actuating input’ or ’no actuating input’. These effect in turn propagate through the actuator subsystem level, which causes ’faulty actuating’ or ’no actuating’. These effect may also occur when the actuators have a mechanical failure.

3 Homework 3

3.1 Mean-change detection using deterministic limit check

To simulate the signal to observed we defined a signal with Gaussian white noise with arbitrary constant variance and an arbitrary mean that changes at time instant k0 = 1001. Since the mean change is large in comparison to the variance, we may write a simple algorithm that checks whether the signal 'z' is within some boundary given by deterministic limit. The output of the algorithm is 0, [FALSE] or 1, [TRUE], true meaning that a change is detected. This algorithm will work perfectly with a mean change relatively large,leading into a no false positives or negatives detected. To get a more realistic behavior we will decrease the mean change relatively to the variance of the noise.

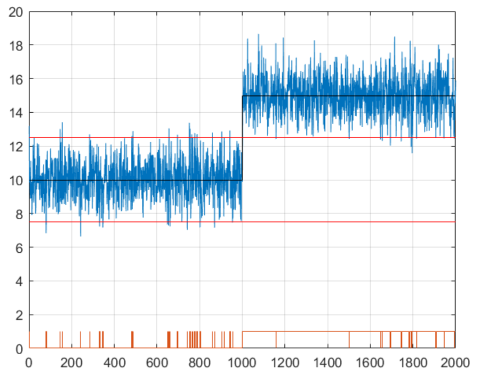


Figure 3: Signal and Small Limits

Some false negatives and false positives start to pop up, as it can be seen in figure 3, and the False Alarm Rate (FAR) and Missed Detection Rate (MDR) will only become worse if you want to detect even smaller mean changes.In order to improve this algorithm you may add the constraint that an W amount of consecutive instants the deterministic limit has to be crossed before the algorithm produces a [TRUE] output. This will decrease the chance of a false output due to noise, but delay the detection of the mean change by W time instants. A similar thing you can do is to average the noisy signal over a time window W, this will increase the robustness of your algorithm to noise peaks, but again delays the detection rate by W. In figure 4 it can be seen the behavior of the applied algorithm.

Using the averaged signal (in purple, with a moving averaging window of 30 time instants) you can detect small mean changes. However, as can be seen from the figure, for the first time instants the signal can as the signal can only by

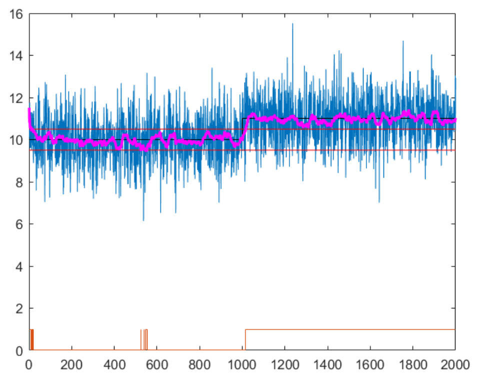


Figure 4: Averaged Signal

averaged over a truncated part of the set window. Also due to the averaging (or low-pass filtering) effect the algorithm is slower to detect the true mean change. This may be somewhat counteracted by using a weighted average, that lays more weight on the more recent time instants of your window. In conclusion, the deterministic limit check can be made less susceptible to noise by averaging the signal first, however this will cause delay on your detection dependent on the averaging window.

3.2 Mean-change detection using probabilistic limit check

By using a probabilistic test the variance can be used to determine if a signal is likely to contain a fault (mean-change) or not. The main advantage is that this test uses the signal data to determine if there is a fault in the signal. This means that limits do not need to be determined. Instead, the probability that this signal is within some interval of the estimated mean needs to be calculated. This is done with the following equations:

$$\hat{\mu}(k) = \hat{\mu}(k-1) + \frac{1}{k}[z(k) - \hat{\mu}(k-1)] \tag{1}$$

$$\hat{\sigma}^2(k) = \frac{k-2}{k-1}\hat{\sigma}^2(k-1) + \frac{1}{k}[z(k) - \hat{\mu}(k-1)]^2 \tag{2}$$

Using the estimated mean and variance we can calculate with increasing accuracy a Gaussian distribution for the measured signal. If a value of the signal is outside the region of acceptance (with area 1 - α), then the signal will be detected as containing a fault (mean-change). The value of α has to be low enough to prevent the system constantly giving off a false alarm.

In the figure above you can see an example of the probabilistic method in use. The output of the algorithm is again plotted as 0 [FALSE] and 1 [TRUE]. One thing that has to be noted is that the signals mean and variance should be

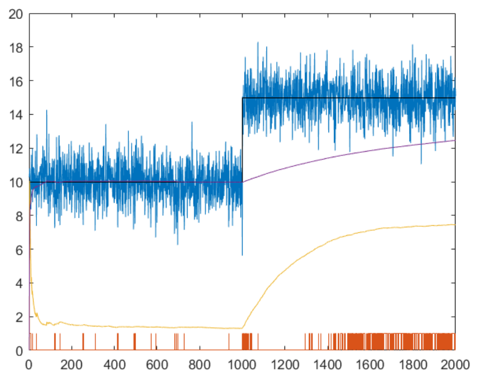


Figure 5: Probabilistic Method

robust enough to not change to much if the signal produces a fault, as a faulty signal will change the mean and variance. Again at the first instants you can see some false detection's in the plotted output, this is due to the fact that the estimate of mean and variance is rough, caused by the small amount of signal values that can be used as data. In the instants 100 - 1000 you there are some false alarms due to peak noise, and at the mean changing instant k0 = 1001 you can see the true detection. For the time instants after the mean change, using the algorithm doesn't make sense, since the estimate for the mean and variance are way off with respect to the real mean, plotted in black. In Matlab we have confirmed that our design α matches the value for the FAR, by applying the algorithm as above many times to the same signal but with regenerated noise sequence.

3.3 Synthetic multi-sinusoidal signal

The first thing to implement is the multi-sinusoidal signal for this we compute it in Matlab by adding 2 sinusoidals with different frequency and amplitude. The fundamental frequency is 1 Hz with an amplitude of 0.25 and the harmonics have a frequency of 20 Hz with an amplitude of 0.0625. The resulting signal can be seen in figure 6, which shows 2 periods of the multisinusoidal signal.

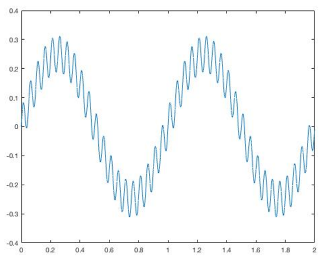


Figure 6: Multisinusoidal.

To simulate a fault in the system, we create an other sinusoidal, that has or a different frequency, or a different amplitude or a different phase in a spurious harmonic. In this simulation it is assumed that only one type of fault can happened at the same time for visualization purposes. The 3 different faults can be seen in figure 7, the top figure contains a fault in frequency, the second a fault in amplitude and the third a fault in phase.

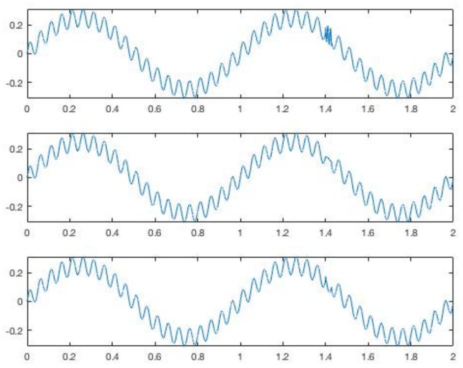
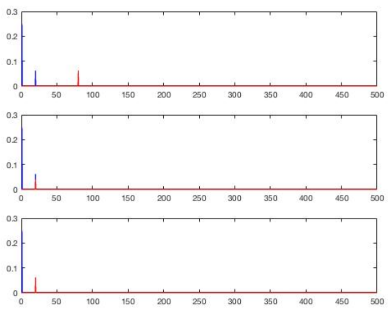
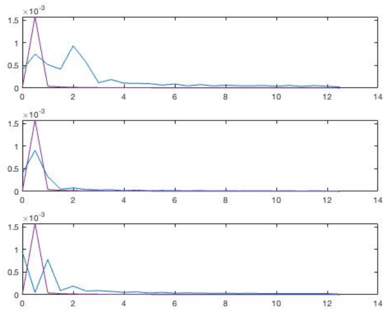


Figure 7: Faulty Sinusoidal harmonic.

Because detecting faults of sinusoidal signals in time domain is very complicated, we use the Fourier transform to achieve this detection. The algorithm computes the Fast Fourier Transform (FFT) in each harmonic of the whole signal, and then compares it with a non-currupted one. It can be determined if there is a fault in any harmonic. This is done with a loop that takes the period of the harmonics and evaluates the FFT obtained through the whole signal. This can be seen in figures 12a and 8b the Fourier transform of the whole system and the fast Fourier transform of the harmonics respectively. Because of the changes in values of the FFT are extremely small, we have to multiply them by a gain of 100000 and use the command *ceil* in matlab and then compare the values. This step was made due to the sensitivity of the FFT and some times it deliver a false fault. To make the algorithm more robust we implement a boundary of $\pm 1.5\%$, because in a real scenario the actual signal might be not perfect, like most of the real components, it will have a range of operation. This will help the system to do not get false positives. The algorithm delivers a fault matrix with values of 0 [TRUE] and 1 [FALSE] indicating the detection of a fault. An extra implementation was made, by searching the position of the fault. With this we can determine which harmonic has the problem.



(a) Fourier Transformation.



(b) Fast Fourier Transformation.

Figure 8: Fault detection multi-sinusoidal.

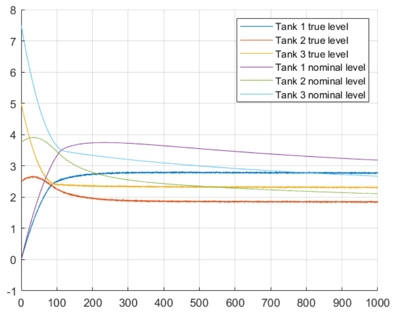
4 Homework 4

4.1 Task 1: Fault detection in three tank model

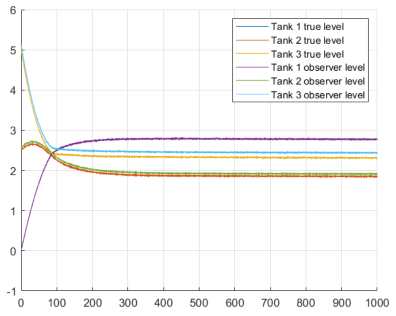
We used the provided three tank Simulink model. Then we added a random number generator block to simulate the measurement uncertainty on the levels. The values for the level are around 3 m, so we scaled the blocks to generate zero mean numbers with variance 0.05 m. To ensure we always get a new independent noise signal, the blocks generate numbers dependent on a seed variable that changes every time the Matlab code is run.
Then we created a discrete time copy of the provided model, by changing the tank plants to include a discrete time integrator block. This discrete model now assumes some uncertainty on the model parameters, these variables are assigned in the Matlab code.
If we then simulate both the true continuous time model and the discrete time model for the same constant pump-flow input we get the following result.

The next figures show that the uncertainty of the model parameters results in an slight offset in the final equilibrium state. The uncertainty in the output value is not present for the discrete time model, because you calculate those values directly instead of measuring, which is subject to noise. Next we implemented a fault detection observer, that creates an estimate of the state from the nominal dynamics and an output error feedback term with a feedback gain lambda equals 1. The observer estimated the following state trajectories for the tank levels.

From figure 9b it can be seen that the observer converges to the real model, except for some noise term since the observer nominal dynamics are based on noisy output measurements. Also since the initial conditions for the observer are different, there is a relatively large error in the first time instants, which reduces quickly due to the feedback term in the observer.



(a) System vs Model



(b) System vs Observer

Figure 9: Delayed observer dynamics

Next we created a threshold generator in Matlab. The threshold uses the fact that the following equation holds:

$$|r_i(k+1)| \leq \lambda_i |r_i(k)| + \bar{\delta}_i \tag{3}$$

We then used this to detect three different simulated faults. Our detection algorithm produces a zero vector with a 1 at time instant k, if a fault is detected a time instant k. For illustration we now plot the detection of a leakage in tank 1. Looking at the next figure, clearly the absolute value of the residual of tank 1 start to grow at the fault time instant k=400, after it accumulates enough (over the detection time) it crosses the absolute value of the threshold generator value and the detection algorithm produces a 1 from there.

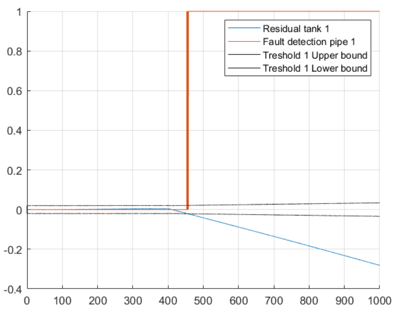


Figure 10: Threshold fault detection

5 Homework 5

5.1 Creating two Local Fault Diagnosers

Firstly, we need to divide the three tank systems into two smartly chosen subsystems. A logical choice is to consider tank 1 and tank 3 as one subsystem and tank 2 as a subsystem. The local states for subsystem 1 will then be the heights of tank 1 and tank 2, $x1 = [x1, x3]$, and for subsystem 2: x2 = [x2]. The interconnection variables are the heights of tank 2 and tank 3. How the observer model of the three tank system was divided into two subsystems can be found in the Simulink model and in the figure below.

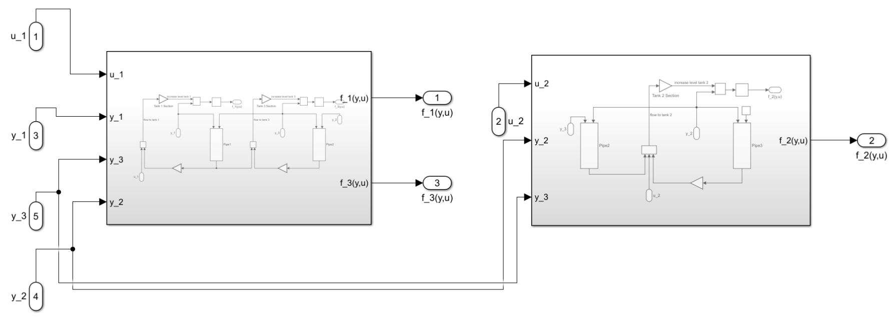
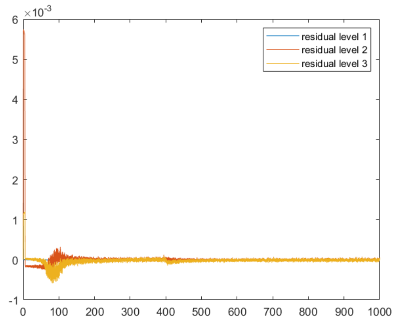
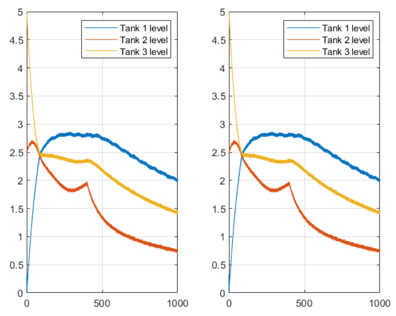


Figure 11: Subsystems

To show that without communication delays the performances are the same, we implemented the original observer system in parallel to the interconnected subsystem observer system. By subtracting both outputs and showing that the residual is zero, we can verify data is the same for both observers, e.g. the performances are the same and residuals are zero.
Then, we implemented another LFD observer system, but now we added communication delay to the transmission of the interconnection variables between both subsystems. In Simulink this has been modelled with an additional parallel LFD observer. Now delay blocks have been added in between the interconnection variables. The results are shown in the figures on the next page. It can now be shown that the residual between the model without communication delay oscillates around zero, with amplitude dependent on the duration of the delay. If we assume a time delay of 500 time instants, 5 s, and a leakage at t = 400s, we get the following observer results. In the figure below the observer with and without delay are plotted to the left and right, respectively. From these plots it can be seen that there is almost no difference due to the time delay. This is to be expected because the connection between the subsystems is a pipe can only let through a limited amount of water per time unit. This amount is small in comparison to the volume of the tanks. So we conclude the LFD observer is robust against time delays.



(a) Residuals



(b) Observer levels

Figure 12: Delayed observer dynamics

6 Homework 6

6.1 Introduction

In this assignment we are asked to design a controller for the three tank non-linear system, possibly containing faults. First we linearize the system and design an output feedback controller, then we design a fault tolerant controller using model matching and virtual sensor approach.

6.2

To linearise the system, we first find an equilibrium point, using conservation of mass and the fact that in an equilibrium all tank levels stay constant. We can pick arbitrary pump flow and pipe flow for tank 2 with the condition $U_2 < \phi_{2,0}$. From there we calculate the other levels and flow rates. We make sure we have the following relation for the tank levels: X1 > X3 > X2. If we then take partial derivatives we get the following linearised dynamical equations for the plant:

$$\dot{X}_{1\delta} = -\frac{c_1 A_1^p \sqrt{2g}}{A_1 2\sqrt{X_{1e} - X_{3e}}} X_{1\delta} + \frac{U_{1\delta}}{A_1} \tag{4}$$

$$\dot{X}_{2\delta} = \left(-\frac{c_2 A_2^p \sqrt{2g}}{A_2 2\sqrt{X_{3e} - X_{2e}}} - \frac{c_3 A_3^p \sqrt{2g}}{A_2 2\sqrt{X_2}}\right) X_{2\delta} + \frac{U_{2\delta}}{A_2} \tag{5}$$

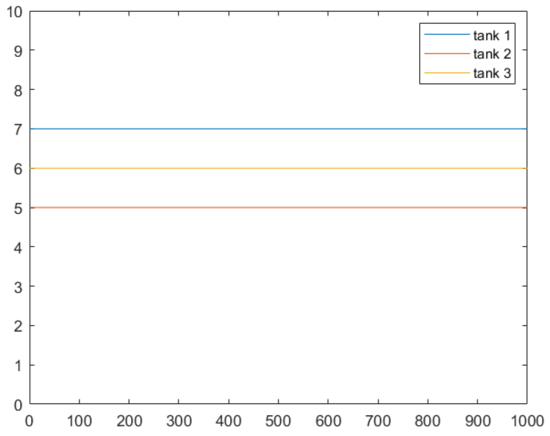
$$\dot{X}_{3\delta} = \left(-\frac{c_1 A_1^p \sqrt{2g}}{A_3 2\sqrt{X_{1e} - X_{3e}}} - \frac{c_2 A_2^p \sqrt{2g}}{A_3 2\sqrt{X_{3e} - X_{2e}}}\right) X_{3\delta} \tag{6}$$

Now we design a linear output feedback controller; u = -KC, so that the system [A-BKC] is asymptotically stable.

Next, we have simulated a sensor fault and an actuator fault with an accommodated controller using the model matching approach. We simulate a faulty sensor by changing C = [1 1 1] to C_f = [1 0.5 1]. Using the model matching technique and pseudo inverse method; $u(t) = -KPy(t)$ where,

$$P = CC_f^+ = CC_f'(C_f C_f')^{-1} \tag{7}$$

In the figure below the nonlinear tanks trajectory results are shown for the described faulty plant.



As expected the new controller can also keep the levels constant at the same chosen equilibrium values by correcting for the faulty sensor.

Now, we model an actuator fault by scaling the B matrix to B_f = [B(1) 0.5B(2) B(3)], note that B_f has the same column span as B, which ensures a solution exists if we implement $u(t) = -KNy(t)$ with,

$$N = B_f^+ B = (B_f' B_f)^{-1} B_f' B \tag{8}$$

Next we use the virtual sensor method to generate an observer that lets a nominal controller control a faulty plant. To do this a couple of new matrices have to be calculated including the Luenberger observer. This observer is then used to observe the faulty plant and correct for faults found between the expected output of the linear plant and the actual output of the nonlinear plant. When implementing this the observer will be able to correct for any potential fault without having to recalculate the entire model, instead the model will be able to adapt to a new fault.

The virtual actuator works with a similar principle as the virtual sensor, that is that the model will be able to adapt to the detected fault and not have to be recalculated if a change in the fault is detected. This virtual actuator is able to control the system even if one of the two actuators fails. The model should work by implementing the block diagram on slide 7 of lecture 4 in simulink.

Due to a matrix error in the simulink model that we were unable to resolve we were unable to get the models working. The model for the linear model matching should be correct except for the matrix multiplication error. The nonlinear systems would then be simulated by using the Δy of the nonlinear plant instead of the Δy of the linearized plant.

The matrix error also occurred in the virtual sensor approach. We are less certain that we did this in the correct manner, however due to the matrix error we were unable to test our system and check if it indeed was correct or not. The virtual sensor approach uses both the linear and nonlinear plant in determining what control action it needs to take and therefore does not need to be run twice for the linear and nonlinear approach.

Appendices

AppendixHomework 1

AppendixHomework 2

AppendixHomework 3

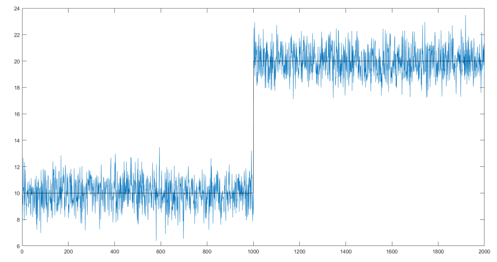


Figure 13: Random Signal

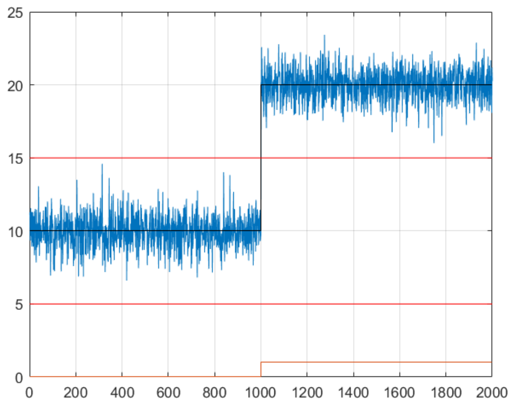


Figure 14: Signal with limits