Fault detection example on resilience analysis for warehouse with autonomous robots

Attila Lelkó and Balázs Németh*

1 Introduction

This example is based on the Matlab example *Control and Simulate Multiple Warehouse Robots*[1]. The original example shows how to control and simulate multiple robots working in a warehouse facility or distribution center. The robots drive around the facility picking up packages and delivering them to stations for storing or processing.

A central scheduler sends commands to robots to pick up packages from the loading station and deliver them to a specific unloading station. The robot controller plans the trajectory based on the locations of the loading and unloading stations, and generates velocity commands for the robot. These commands are fed to the plant, which contains a differential-drive robot model for executing the velocity commands and returning ground-truth poses of the robot. The poses are fed back to the scheduler and controller for tracking the robot status. This workflow is done for a group of 5 robots, which are all scheduled, tracked, and modeled simultaneously.

Given this rather complex control system, the goal of the resilience analysis is to detect potential malfunctions in the dynamics of the robots based on measurements of the system states and inputs. The analysis does not require knowledge on the dynamic model of the robots.

2 Fundamentals of the resilience analysis

The goal of this documentation is to provide a brief handout for the code uploaded to the related GitHub repository.[2] Therefore, only the basics can be found here, for the details the authors refer to the related literature. In this example, the dynamics of a warehouse robot can be approximated based on the Kolmogorov-Arnold representation theorem in Linear Parameter-Varying (LPV) form as[3]

$$x_{k+1} = Ax_k + B(x_k)u_k, \tag{1}$$

where A is the identity matrix, $x = [x,\ y,\ \psi]^{\mathrm{T}}$ the pose of the robot, and the inputs are $u = [v,\ \omega]^{\mathrm{T}}$. The change in the state between consecutive time steps can be written as:

$$\Delta x_k = B(x_k)u_k, \tag{2}$$

*A. Lelkó and B. Németh are with Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN).

[1]https://www.mathworks.com/help/robotics/ug/control-and-simulate-multiple-warehouse-robots.html

[2]https://github.com/sztaki-hu/WarehouseExample

[3]Németh, B, and Gáspár P., "Scheduling-Informed LPV Approach for Control Design Purposes," submitted to *6th IFAC Worshop on Linear Parameter Varying Systems* (2025)

where $B(x_k)$ maps the inputs of a robot into the change of states. The values of $B(x_k)$ can be approximated by taking n measurements of $\Delta X_k = [\Delta x_k,\ \Delta x_{k+1}, \dots \Delta x_{k+n},\ ]$ and $U_k = [u_k,\ u_{k+1}, \dots u_{k+n},\ ]$ and calculating the least-square approximation of:

$$\hat{B}(x_k) = \Delta X_k U_k^{\mathrm{T}} (U_k U_k^{\mathrm{T}})^{-1} \tag{3}$$

The algorithm calculates $\hat{B}(x_k)$ every time step. The resilience analysis is based on detecting atypical values of $\hat{B}(x_k)$ that suggest a malfunction in the warehouse robot.

3 Illustration of healthy operation

During normal operation, the algorithm results in the values in Figure 1 for matrix $\hat{B}(x_k)$:

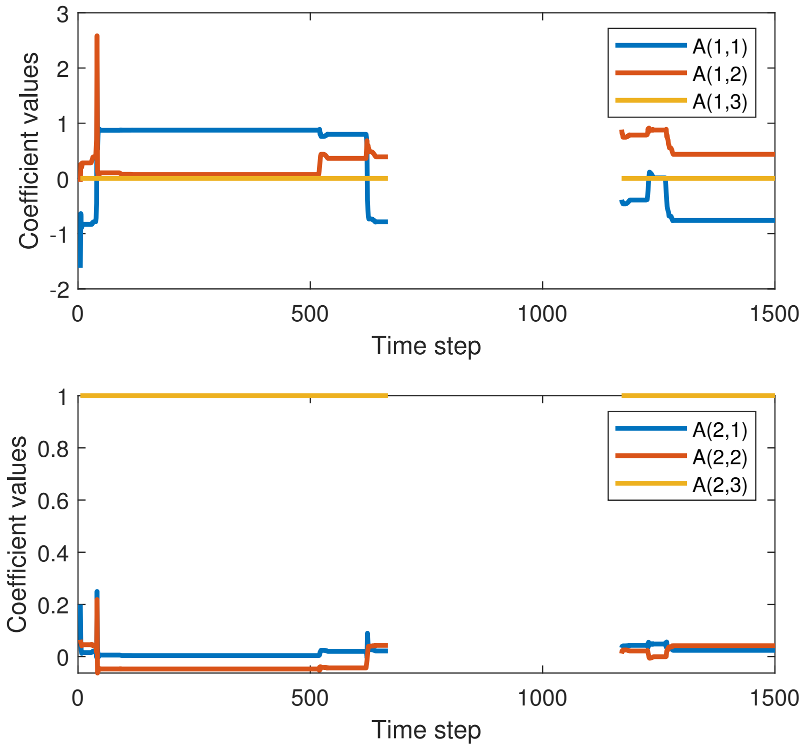

Figure 1: $\hat{B}(x_k)$ during normal operation

Time steps without corresponding values indicate situations where the identification cannot be carried out. These situations occur typically when the robot is not moving ($v_k = 0$ and $\omega_k = 0$). Based on the presented values the normal operation can be characterized (without any knowledge on the motion model) by the typical values of the elements in $\hat{B}(x_k)$.

- $A(1,1)$ and $A(1,2)$ varies between [-1;1] with larger spikes when the robot turns
- $A(1,3)$ is 0 during the whole operation and $A(2,1)$ and $A(2,2)$ is around 0.
- $A(2,3)$ is 1 during the whole operation.

Since A(1,3) and A(2,3) are constants during normal operation, sudden changes in these values can be the best indication of a fault in the warehouse robot.

As a reference, the trajectory of the robot during normal operation can be seen in Figure 2.

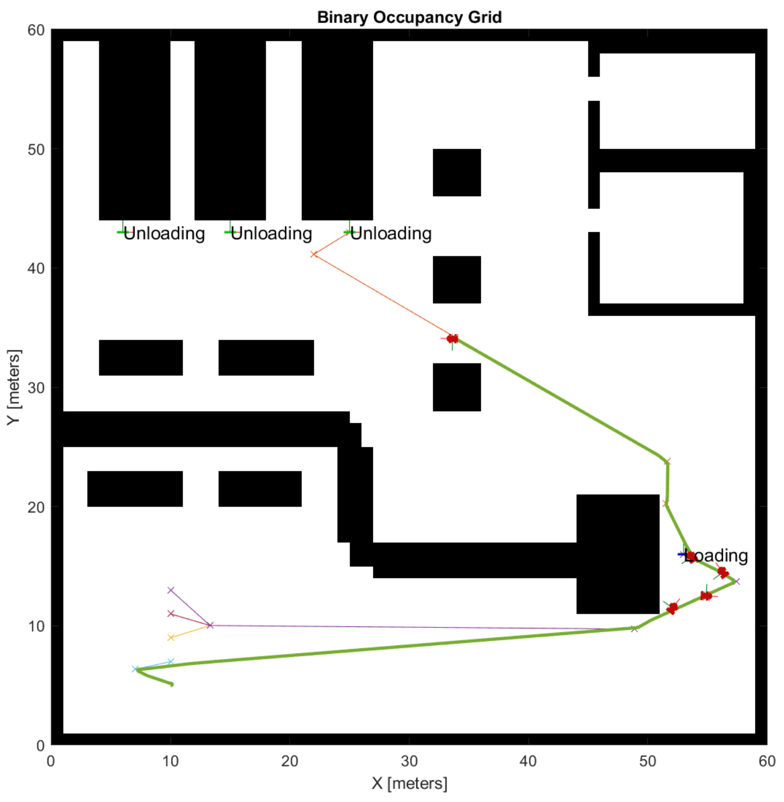

Figure 2: Effect of a small fault at $T = 300$s on the robot trajectory (green)

4 Detection of a small-scaled fault

The next example shows a scenario where a small fault occurs during operation. The bearing in the left wheel of a robot fails, increasing the friction of that wheel. This results in reducing the wheel speed to 90% of the value set by the controller. The values of $\hat{B}(x_k)$ are shown in Figure 3.

This small fault is not enough for the close loop to lose stability, the controller can navigate the robot along its path, as seen in Figure 4. However, the presented algorithm can signal a fault even before it can be observed in the robot's trajectory. The only difference from Figure 2 is that the robot was not able to travel as further along the path in the given time due to the bearing malfunction.

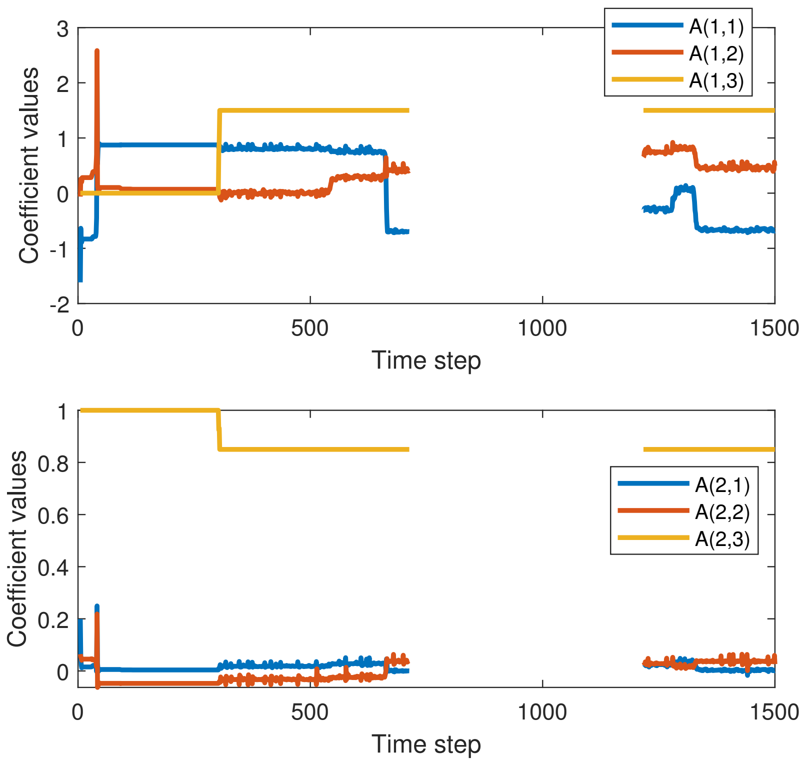

Figure 3: Effect of a small fault at $T = 300$ time step on $\hat{B}(x_k)$

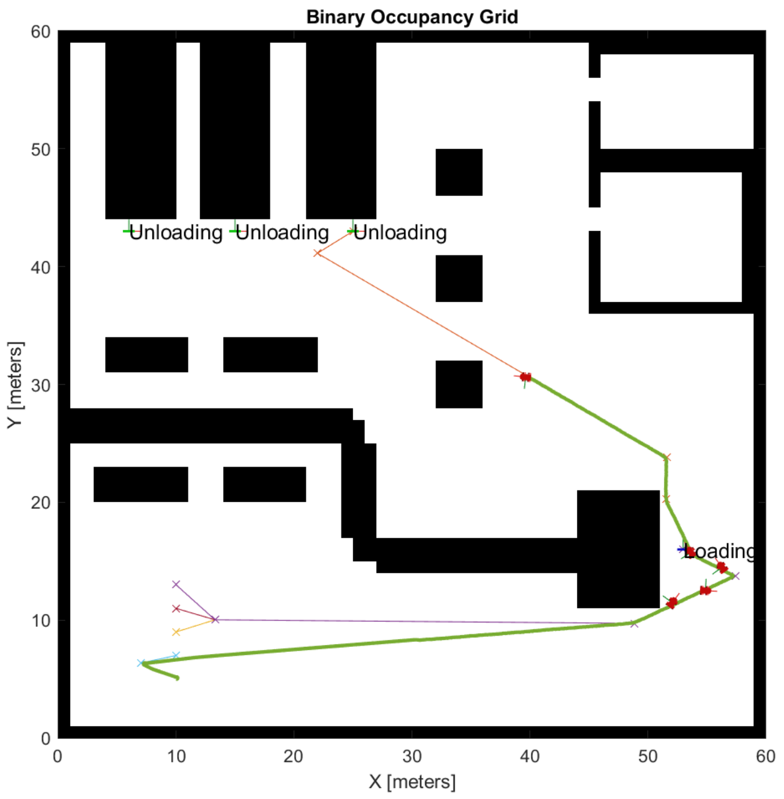

Figure 4: Effect of a small fault at $T = 300$ time step on the robot trajectory (green)

5 Detection of a large-scaled fault

Another example shows a scenario with a significantly larger fault when the controller is not able to stabilize the robot's motion. In this case, the speed of both wheels is overridden and the robot starts going backward. This kind of fault can be easily recognized by just looking at the trajectories in Figure 6, however, this algorithm can detect it by examining the values of $\hat{B}(x_k)$ in Figure 5.

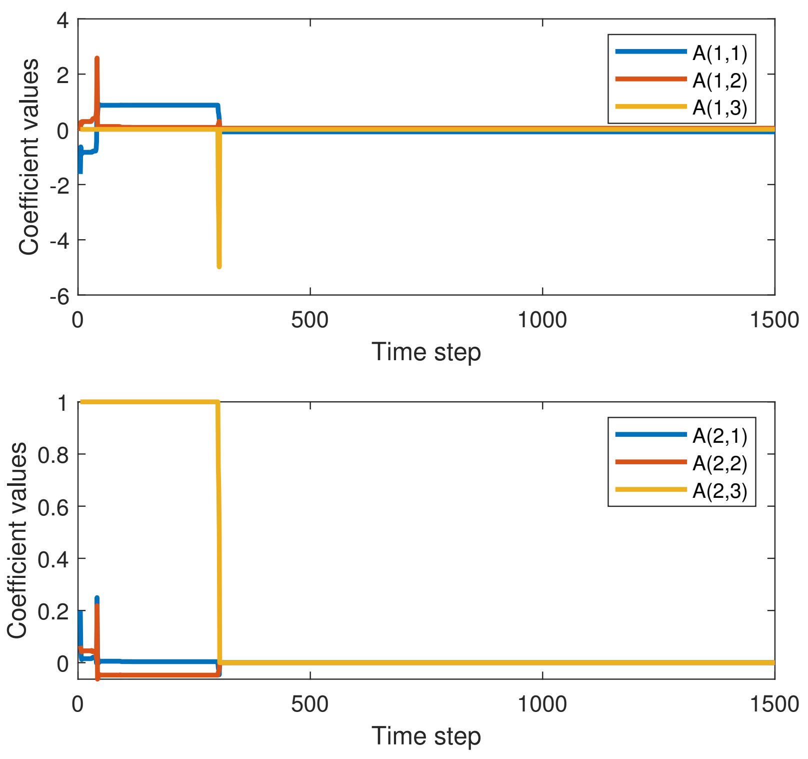

Figure 5: Effect of a large fault at $T = 300$ time step on $\hat{B}(x_k)$

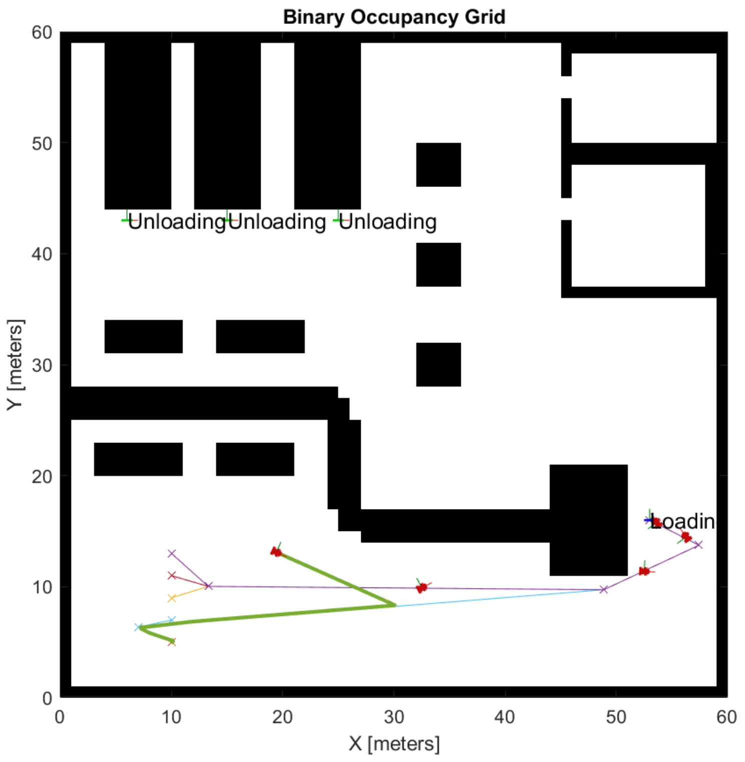

Figure 6: Effect of a large fault at $T = 300$ time step on the robot trajectory (green)