Proof of concept for detecting faults in manufacturing robots using neural networks

Ali C. Kheirabadi, Ph.D.[1]

[1]Vancouver, BC, Canada

January 24, 2021

1 Introduction

This brief document provides a simple demonstration of how artificial neural networks may be used for fault detection in robotic manipulators used in manufacturing environments. The task is achieved via three steps:

1. *Simulating a robotic manipulator to generate data* – Ideally, a physical robot from a manufacturing plant should be used for data collection. However, in order to minimize time and costs during an algorithm's prototyping stage, a simulation is perfectly appropriate and desirable.
2. *Tuning a neural network to learn fault-free operation* – With a sufficient amount of data collected under normal conditions, a simple feed-forward neural network may be constructed to predict fault-free operation. Specifically, the neural network maps the causal relationships between the loads that drive the robotic manipulator and the movements of the mechanism. Since these relationships are established under normal operation, then, if measured data fails to match neural network predictions, a fault is likely to have occurred.
3. *Simulating and detecting faulty joints* – In the final stage, the friction loads in the robotic joints are increased to simulate a fault (*e.g.* broken ball-bearings). Measured data is then compared to neural network predictions, and simple logical statements are used to identify exactly which joints are faulty.

2 Simulating a robotic manipulator to generate data

2.1 System description

A visual description of the robotic manipulator system is shown in Fig. 1. At the center exists the robotic manipulator which consists of a base and two links. On either side of the manipulator exist two workstations. The objective of the robot is to carry workpieces from any point on Workstation 1 or Workstation 2 to any other point on either workstation.

The top diagram in Fig. 1 shows a side-view of the system. Here, the angles θ_1 and θ_2 corresponding to Link 1 and Link 2 are visible. Additionally, the motor torques T_{θ_1} and T_{θ_2} necessary for driving each link are also shown. The bottom diagram in Fig. 1 shows a top-view of the system. Here, the base's yaw angle γ and its driving torque T_γ are visible. The yaw mechanism of the base is necessary for rotating the robot to face different workstations.

Although not shown in the diagram, all joints and the base contain friction factors which resist rotation. The friction factor that resists yaw motion of the base is represented by the variable b_γ. The joint friction factors that resist rotations of Link 1 and Link 2 are denoted by the terms b_{θ_1} and b_{θ_2}.

2.2 Mathematical model

A detailed mathematical model of the robotic manipulator is presented in Appendix A. In the current section, a practical description is provided for non-technical readers. The dynamic model of the manipulator may be broken down into the following two processes which are also highlighted in Fig. 2:

1. *Kinetics* - The motors located at the robot's base and at the joints of Link 1 and Link 2 generate torques that drive the angular motions of the links. This relationship between torques and angular movements is described through kinetics.

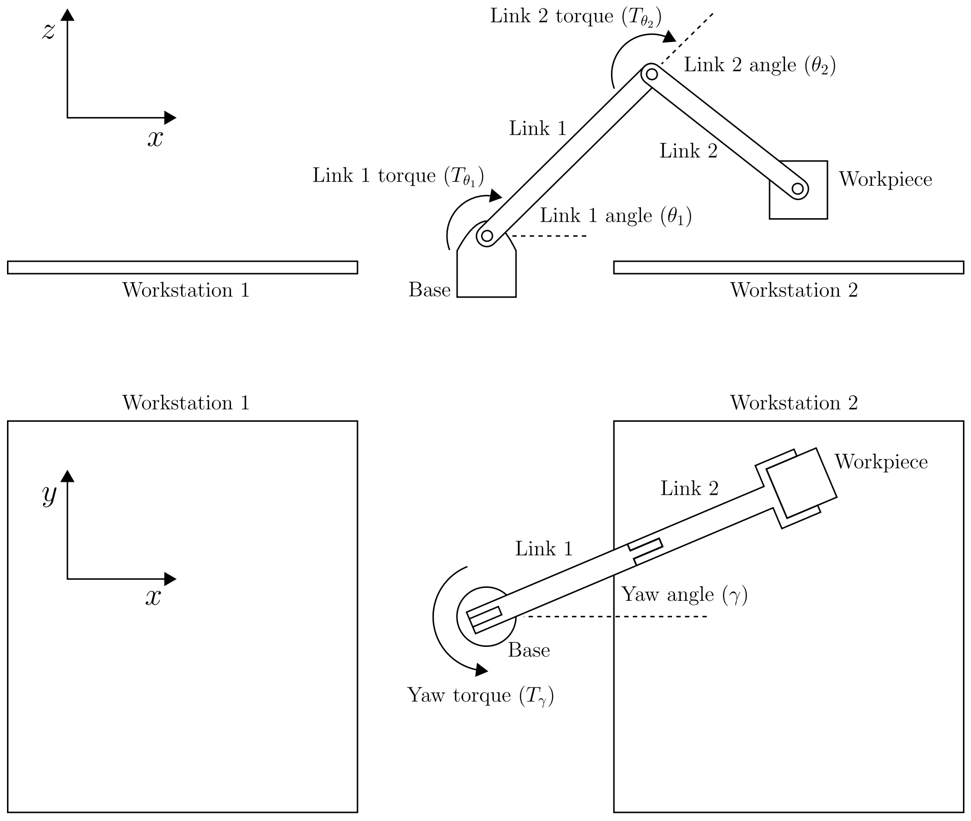

Figure 1: Side-view and top-view schematics of the robotic manipulator system operating between two workstations.

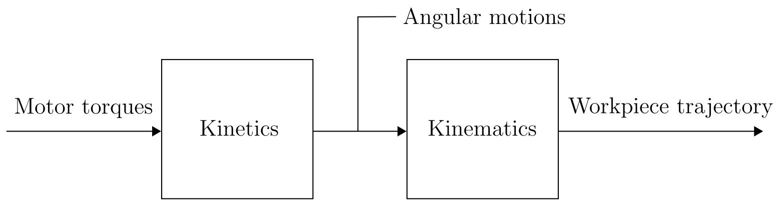

Figure 2: Block diagram of the dynamic model used to simulate a robotic manipulator.

2. *Kinematics* - Once the angular motions of the links have been established, the trajectory of the workpiece is computed using the robot's geometric properties. This relationship between angular movements and linear trajectories is described using kinematics.

In the physical world, the processes described in Fig. 2 occur from left to right. That is, the motors generate torques that rotate the links, and the rotations of the links dictate the trajectory of the workpiece. In the current project, the desired trajectory of the workpiece is prescribed, and the required torques and angular movements of the links are back-calculated. The processes in Fig. 2 are therefore implemented from right to left. This approach eliminates the need to design control systems that yield the desired workpiece trajectory; hence reducing prototyping time and costs.

Ultimately, the output of the mathematical model is a system of equations that describe the torque generated by each motor as a function of the angular motions of the robot as follows:

$$T_\gamma = f_\gamma(\gamma, \theta_1, \theta_2, \dot{\gamma}, \dot{\theta}_1, \dot{\theta}_2, \ddot{\gamma}, \ddot{\theta}_1, \ddot{\theta}_2), \quad (1)$$

$$T_{\theta_1} = f_{\theta_1}(\gamma, \theta_1, \theta_2, \dot{\gamma}, \dot{\theta}_1, \dot{\theta}_2, \ddot{\gamma}, \ddot{\theta}_1, \ddot{\theta}_2), \quad (2)$$

$$T_{\theta_2} = f_{\theta_2}(\gamma, \theta_1, \theta_2, \dot{\gamma}, \dot{\theta}_1, \dot{\theta}_2, \ddot{\gamma}, \ddot{\theta}_1, \ddot{\theta}_2), \quad (3)$$

and the angular motions are computed based on the desired workpiece trajectory. The dot operator above each variable represents a time-derivative. For example, if γ is the robot's yaw angle, then $\dot{\gamma}$ is the yaw angular velocity,

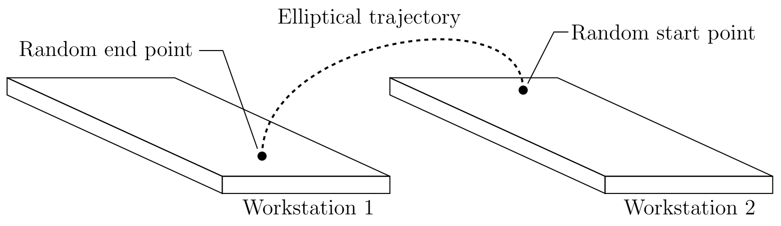

Figure 3: Schematic demonstrating the process of randomly generating a workpiece trajectory for a single run.

and $\ddot{\gamma}$ is the yaw angular acceleration. What these equations basically state is that, for every possible state of motion of the robot's links, there exists a unique combination of motor torques that are required to produce that state. The purpose of a neural network is then to *learn* this relationship between angular motions and torques by mapping out the above equations.

2.3 Data generation

The process of data generation for neural network tuning is shown in Fig. 3. The purpose of the robot is to move a workpiece from one location on either workstation to any other location on either workstation. This process of moving a single workpiece from one point to another is called a *run*. In order to plan a single run, two points on either workstation are randomly selected, and an elliptical trajectory is generated between them. This ellipse represents the desired trajectory of the workpiece for the current run, and the motor torques required to deliver this trajectory are computed using the mathematical model from Appendix A.

The above process is then repeated for N runs to generate sufficient data for neural network tuning. A sample set of data collected over $N = 10$ runs is plotted in Fig. 4. The top figure show the robot's yaw and link angles over the runs. The next two figures show the velocities and accelerations of these angles. The final plots shows the corresponding yaw and link motor torques. This information is displayed to help the reader understand what measurements would be necessary from a manufacturing plant for fault detection, and how these time-series data might appear when visualized.

3 Tuning a neural network to *learn* fault-free operation

3.1 Neural network description

The specific neural network used for the current proof of concept is shown in Fig. 5. The input layer consists of nine neurons that represent the angular motions of the robotic manipulator; that is, its angles, angular velocities, and angular accelerations. There is a single hidden layer consisting of 20 neurons which feeds into an output layer with three neurons that represent the robot's motor torques. The neurons in the hidden layer contain a nonlinear saturation function while those in the output layer do not. This scheme enables the neural networks to be used for nonlinear regression (*i.e.* the outputs can possess any numerical value and are not required to range from zero to one).

3.2 Quantifying fault-free operation

The neural network shown in Fig. 5 is tuned using data collected from $N = 500$ runs. Figure 6 compares the neural network's torque predictions to measured values over $N = 5$ runs with no faults. Focusing on a qualitative assessment for now, it is evident from Fig. 6 that the neural network successfully predicts all three torque values throughout the entire simulation.

For a quantitative evaluation, root-mean-square-error (RMSE) values between neural network predictions and measured torques are plotted in Fig. 7 for fault-free operation. In this figure, each plotted RMSE value corresponds to a single run. That is, once a run is completed, the RMSE for that specific run is computed and plotted. A total of $N = 400$ runs are performed. Across the majority of runs, RMSE values for all three torques fall below $0.1\,\mathrm{kN \cdot m}$. In a minority of runs, RMSE values exceed $0.2\,\mathrm{kN \cdot m}$. This results simply indicates that the neural network is not perfect, and that there will always be occasions in which a fault-detection algorithm leads to false diagnoses. The next section simulates faults in order to quantify the rates of false alarms and successful diagnoses.

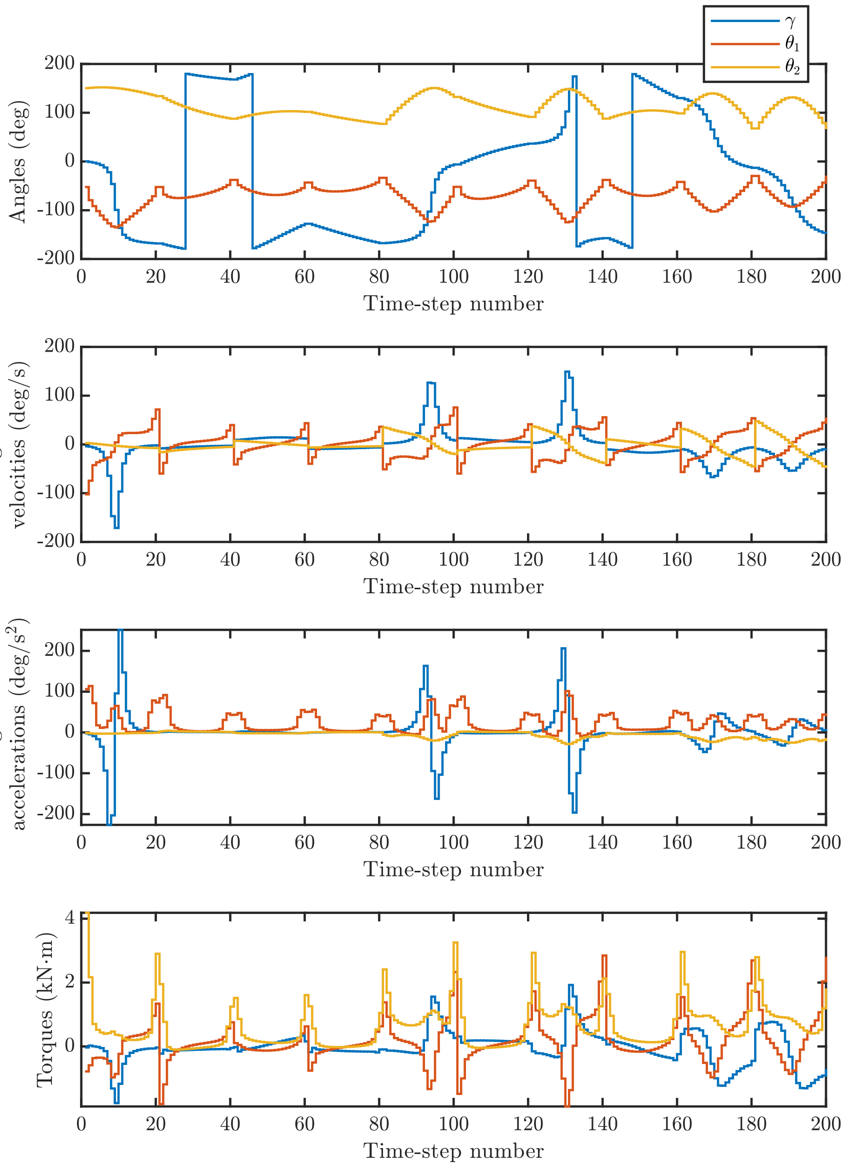

Figure 4: Angular motions and motor torque data collected over $N = 10$ runs under fault-free operation.

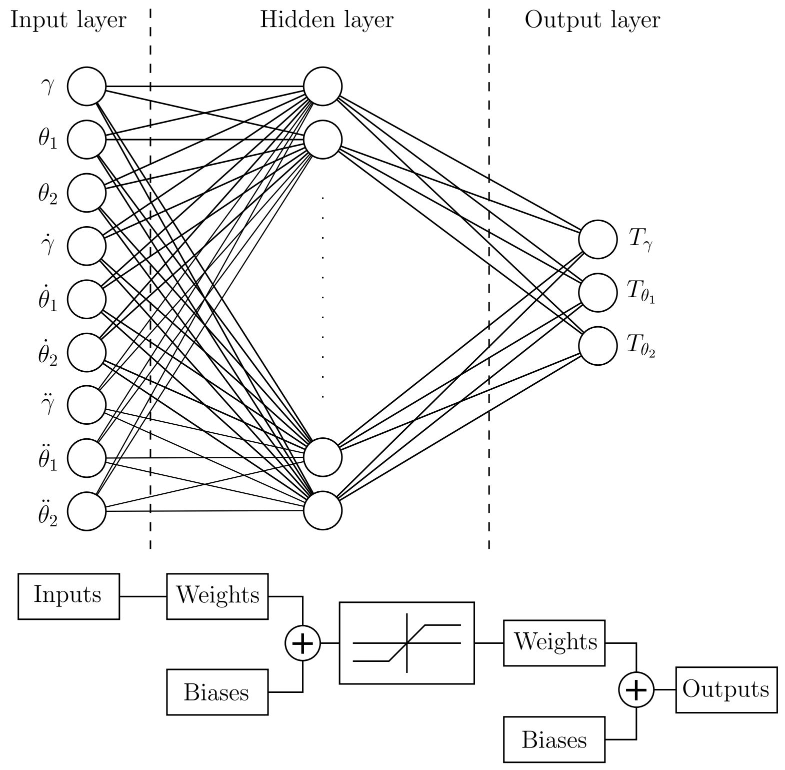

Figure 5: Schematic demonstrating the inputs, outputs, and structure of the neural network used to model the dynamics of a robotic manipulator.

4 Simulating and detecting faulty joints

4.1 Effect on neural network performance

In this proof of concept, a fault occurs due to joint failure. This failure may be caused by, for instance, ball-bearing damage within the joint. To simulate such an occurrence, the friction factor of the joint that encounters a fault is doubled. First, it is interesting to visualize the impact of failure of all three joints on neural network performance.

Figure 8 plots a comparison between neural network predictions and measured torque values across $N = 5$ runs. During these runs, all joints are operating with faults; that is, their friction factors have doubled compared to the simulations that were used to tune the neural network. From a qualitative standpoint, it is evident that the neural network's ability to predict motor torque values has diminished. The reader may revisit Fig. 6 to see how well the neural network performed with no faults.

Furthermore, visualizing RMSE values across $N = 400$ runs with faults in all three joints is also informative; this result is shown in Fig. 9. When all three joint experience faults, RMSE values for all torque predictions rise from $0.1\,\text{kN}\cdot\text{m}$ to $0.5\,\text{kN}\cdot\text{m}$ in the majority of runs. RMSE values exceed $0.6\,\text{kN}\cdot\text{m}$ in a minority of runs. The fact that changes in joint friction result in such a substantial rise in RMSE values indicates that the RMSE is a meaningful candidate metric for fault detection.

4.2 Performance of a basic fault-detection algorithm

The simple fault-detection method assessed in the current proof of concept is outlined in Algorithm 1. This algorithm first measures the angular motions and motor torques of the robot throughout each run, then computes the motor torques throughout the run as predicted by the neural network using the measured angular motions.

The subsequent logic on which this algorithm is based states that, if the neural network predictions (*i.e.* the expected motor torques) deviate excessively from measured torque values, then a fault in the mechanical system must be present. To implement this logic, RMSE values between predicted and measured torques throughout the run are first determined. Then, a logical operation is performed to check whether these RMSE values exceed some predefined tolerance α. If the RMSE value corresponding to a particular joint exceeds α during a particular run, then the joint must have operated with a mechanical fault throughout the run.

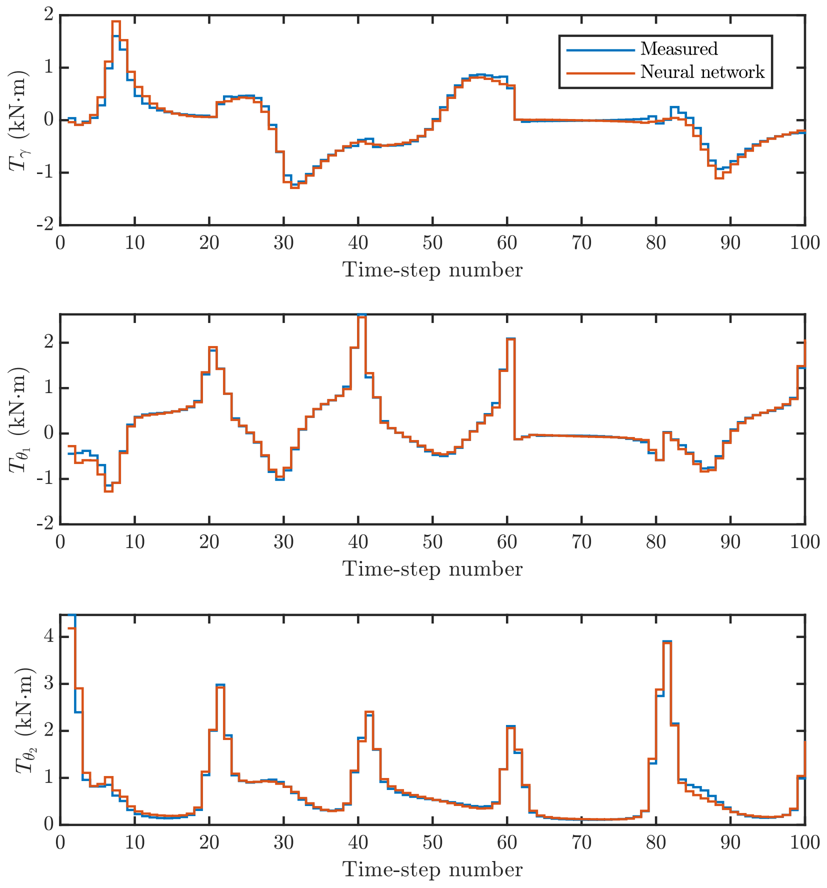

Figure 6: Comparison between measured torque data and values predicted by the neural network over $N = 5$ runs under fault-free operation.

Algorithm 1: Tolerance-based fault detection method implemented at the end of each run.

```
Measure angles $\gamma$, $\theta_1$, $\theta_2$ throughout the run;
Measure angular velocities $\dot{\gamma}$, $\dot{\theta}_1$, $\dot{\theta}_2$ throughout the run;
Measure angular accelerations $\ddot{\gamma}$, $\ddot{\theta}_1$, $\ddot{\theta}_2$ throughout the run;
Measure motor torques $T_\gamma$, $T_{\theta_1}$, $T_{\theta_2}$ throughout the run;
Predict motor torques $\hat{T}_\gamma$, $\hat{T}_{\theta_1}$, $\hat{T}_{\theta_2}$ throughout the run using neural networks;
Compute $E_\gamma = \mathrm{RMSE}(\hat{T}_\gamma - T_\gamma)$, $E_{\theta_1} = \mathrm{RMSE}(\hat{T}_{\theta_1} - T_{\theta_1})$, $E_{\theta_2} = \mathrm{RMSE}(\hat{T}_{\theta_2} - T_{\theta_2})$;
if $E_\gamma > \alpha$ then
    Report fault in the base yaw joint;
end
if $E_{\theta_1} > \alpha$ then
    Report fault in the joint of Link 1;
end
if $E_{\theta_2} > \alpha$ then
    Report fault in the joint of Link 2;
end
```

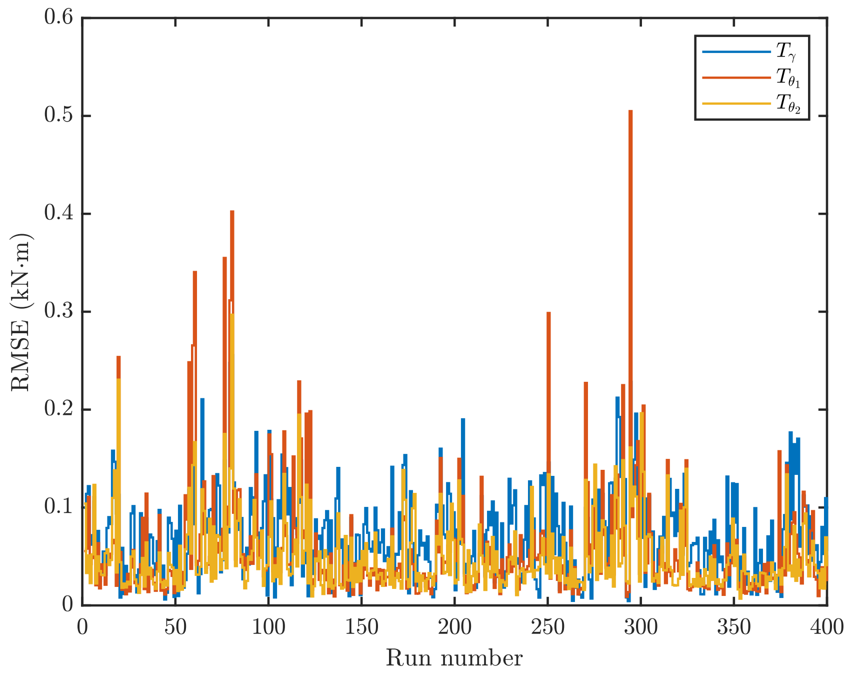

Figure 7: Evolution of RMSE values between measured torque data and neural network predictions over $N = 400$ runs under fault-free operation.

To assess the performance of this algorithm, $N = 400$ runs are generated with a 5 % probability of fault occurrence in any joint. This process is repeated for values of the RMSE tolerance α ranging from zero to one. Figure 10 shows the success rate of the fault detection algorithm as a function of α through two metrics. In the top figure, the percentage of faults that are diagnosed correctly is displayed. In the bottom figure, the percentage of fault-free runs that are misdiagnosed as faulty is shown. The second figure may also be interpreted as the rate of false alarms.

Using Fig. 10, one may determine the optimal value of α for the current application. For instance, at $\alpha = 0.1$, the top figure shows that 80 to 100 % of faults are diagnosed correctly. However, the bottom figure shows that this success rate comes at a cost of about a 20 % rate of false alarm reports. Depending on the cost of missing a fault versus that of a false alarm, the ideal value of α may be established for a given process.

5 Conclusion

This proof of concept study indicates that even a simple tolerance-based fault detection algorithm combined with a neural network predictor can successfully predict fault occurrences in a manufacturing robot. The algorithm is not perfect however, and false alarms and fault misdiagnoses are to be expected. The performance of the algorithm may be enhanced through two avenues of research:

1. *Fine-tuning the neural network* - Through a combination of collecting more data and testing various neural network structures, root-mean-square-error (RMSE) values may be reduced. These improvements would create a finer line between fault-free and faulty operations.

2. *Implementing more advanced fault-detection schemes* - The tolerance-based fault detection algorithm investigated in the current study is the simplest available option. Other techniques that rely on information other than RMSE values should be pulled from the literature and compared.

A Detailed mathematical model

This appendix section describes the process of computing mathematical expressions for motor torques based on a given or desired workpiece trajectory. An energy-based approach is followed; that is, Lagrange's equations are used

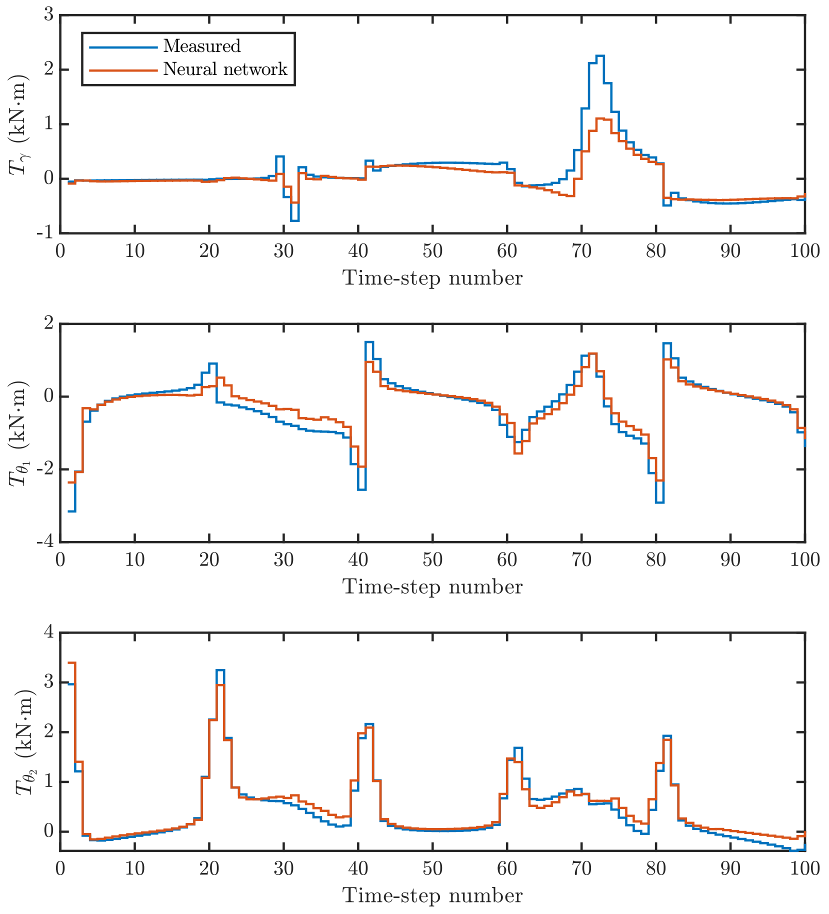

Figure 8: Comparison between measured torque data and values predicted by the neural network over $N = 5$ runs under faulty operation.

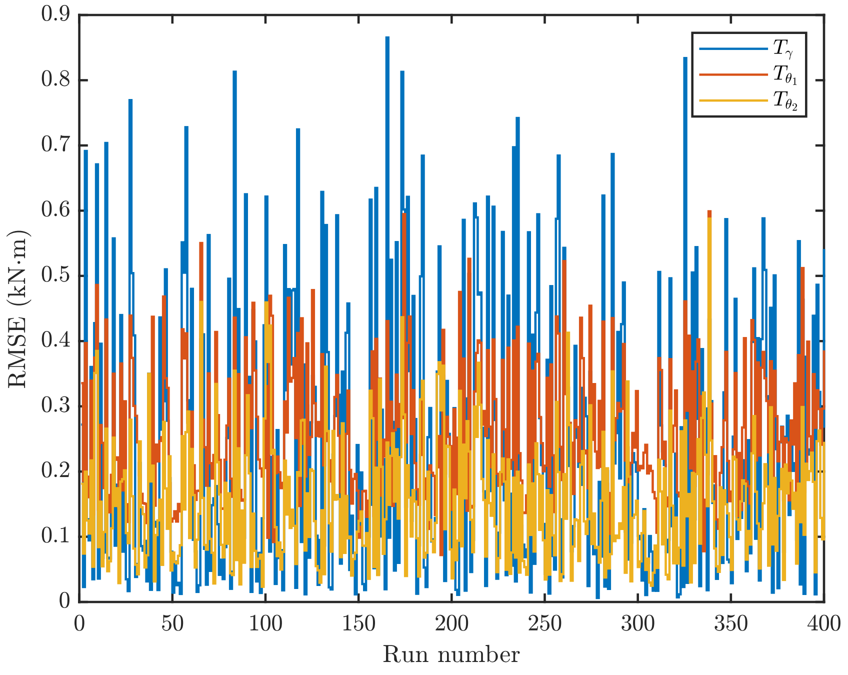

Figure 9: Evolution of RMSE values between measured torque data and neural network predictions over $N = 400$ runs under faulty operation.

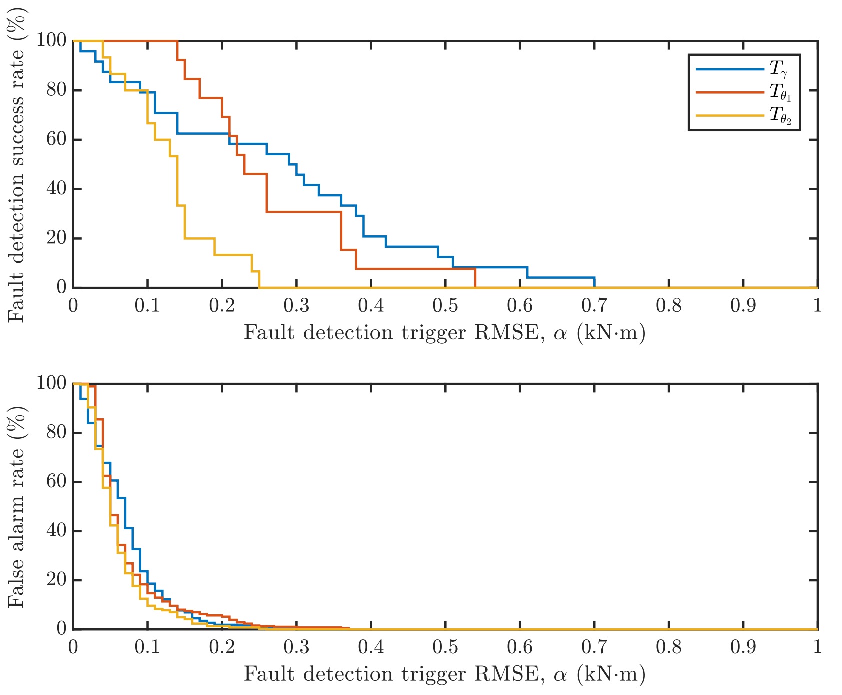

Figure 10: Performance of the fault detection algorithm over $N = 400$ runs with a fault probability of 5 % for any joint. The two figures display the a) rate of successful fault diagnoses, and b) the rate of false alarms (i.e. fault diagnoses when no fault has actually occurred).

to compute the expressions that describe the robot's motion based on it's energy. The starting point is therefore Lagrange's equations as follows:

$$\begin{aligned} T_\gamma &= b_\gamma\dot{\gamma} + \frac{d}{dt}\left(\frac{dE_\mathrm{k}}{d\dot{\gamma}}\right) - \frac{dE_\mathrm{k}}{d\gamma} + \frac{dE_\mathrm{p}}{d\gamma}, && (4)\\ T_{\theta_1} &= b_{\theta_1}\dot{\theta}_1 + \frac{d}{dt}\left(\frac{dE_\mathrm{k}}{d\dot{\theta}_1}\right) - \frac{dE_\mathrm{k}}{d\theta_1} + \frac{dE_\mathrm{p}}{d\theta_1}, && (5)\\ T_{\theta_2} &= b_{\theta_2}\dot{\theta}_2 + \frac{d}{dt}\left(\frac{dE_\mathrm{k}}{d\dot{\theta}_2}\right) - \frac{dE_\mathrm{k}}{d\theta_2} + \frac{dE_\mathrm{p}}{d\theta_2}, && (6) \end{aligned}$$

where E_k and E_p are the kinetic and potential energies of the system. Since, for simplicity, the robot's links are assumed to be massless, only the mass of the workpiece contributes to the energy terms as follows:

$$\begin{aligned} E_\mathrm{k} &= \frac{1}{2}m\left(\mathbf{v}^\mathrm{T}\mathbf{v}\right), && (7)\\ E_\mathrm{p} &= mg\left(\mathbf{r}\cdot\hat{k}\right), && (8) \end{aligned}$$

where m is the mass of the workpiece, g is gravitational acceleration, and $\mathbf{r}$ and $\mathbf{v}$ are the position and velocity vectors of the workpiece at any given time. The velocity vector $\mathbf{v}$ is simply the time-derivative of the position vectors as follows:

$$\mathbf{v} = \dot{\mathbf{r}}, \quad (9)$$

while the position vector $\mathbf{r}$ is found though kinematics using the robot's geometric properties as follows:

$$\mathbf{r} = \mathbf{R}_\gamma\mathbf{R}_{\theta_1}\left(\mathbf{r}_1 + \mathbf{R}_{\theta_2}\mathbf{r}_2\right). \quad (10)$$

The vectors $\mathbf{r}_1$ and $\mathbf{r}_2$, which are defined as follows:

$$\begin{aligned} \mathbf{r}_1 &= \begin{bmatrix} L_1 & 0 & 0 \end{bmatrix}^\mathrm{T}, && (11)\\ \mathbf{r}_2 &= \begin{bmatrix} L_2 & 0 & 0 \end{bmatrix}^\mathrm{T}, && (12) \end{aligned}$$

overlap with the robot's two links of lengths L_1 and L_2 when they are in a zero-displacement position. That way, any displacement of the links may be computed by applying transformations using the following rotation matrices:

$$\begin{aligned} \mathbf{R}_\gamma &= \begin{bmatrix} \cos\gamma & -\sin\gamma & 0\\ \sin\gamma & \cos\gamma & 0\\ 0 & 0 & 1 \end{bmatrix}, && (13)\\ \mathbf{R}_{\theta_1} &= \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1\\ 0 & 1 & 0\\ -\sin\theta_1 & 0 & \cos\theta_1 \end{bmatrix}, && (14)\\ \mathbf{R}_{\theta_2} &= \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2\\ 0 & 1 & 0\\ -\sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix}. && (15) \end{aligned}$$

The above system of equations contain all aspects of the physics involved in modelling a robotic manipulator. They may be solved using a symbolic processor (*i.e.* MATLAB, Mathematica, etc.) to yield the final analytical expressions for T_γ, T_{θ_1}, and T_{θ_2}.