« THE 2ND INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE «SCIENCE AND INFORMATION TECHNOLOGIES IN THE MODERN WORLD»

SECTION NAME – AUTOMATION AND ROBOTICS

ADAPTIVE MODELING OF ROBOT DYNAMICS USING MACHINE LEARNING AND NUMERICAL ODE SOLVING

Tereshchenko Oleksandr
Graduate student
Faculty of Information Security and Electronic Communications
National University "Zaporizhzhia Polytechnic", Ukraine
hanter231103@gmail.com
Korotunova Olena
Ph.D., Associate Professor
Department of Mathematics
National University "Zaporizhzhia Polytechnic", Ukraine
ovkorotunova@gmail.com
Zaytseva Tetyana
Ph.D., Associate Professor
ztan2004@ukr.net
Computer Technologies Department
Oles Honchar Dnipro National University, Ukraine
Shyshkanova Ganna
Ph.D., Associate Professor
Department of Mathematics
National University "Zaporizhzhia Polytechnic", Ukraine
shganna@ukr.net

Traditionally, dynamic models of robots are described using systems of differential equations derived from physical laws: Newton's second law or Lagrange's equations. However, such models often assume the presence of precise values of physical parameters, such as mass, moments of inertia, friction coefficients and elasticity. In reality, these parameters: change over time (wear, heating, deformation); depend on the external environment (humidity, temperature); may not be known exactly at the time of development.

Even though mathematical approaches remain favored despite their complexity, Machine learning-based methods have emerged as a powerful tool for addressing prediction tasks [1, 2].

The approach proposed in this paper is to combine the classical numerical solution of ODE (Ordinary Differential Equations) with machine learning methods such as neural networks, regression, and Bayesian models. The goal is to adapt the mathematical model of the robot to real operating conditions, improve the accuracy of predictions, and improve control stability.

When controlling a two-link manipulator working with variable loads, the standard dynamic model will not be able to accurately predict the behavior when the load changes [3]. If you add a trainable neural network that adjusts the mass matrix depending on the current state, the system will be able to: automatically adapt control, predict behavior under an unknown load, and avoid dangerous resonance modes and oscillations.

In classical modeling for a system with generalized coordinates $q = [q_1, q_2, ..., q_n]^T$, the robot dynamics is described by the equation

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = \tau, \tag{1}$$

where $L = T - V$ is a smooth function called a Lagrangian; T is the kinetic energy and V is the potential energy of the system; τ is the vector of generalized forces.

After deriving the equations, we obtain the standard form in matrix form:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau, \tag{2}$$

where $M(q)$ is the mass matrix; $C(q,\dot{q})$ is the matrix Corriolis function; $G(q)$ is the vector of gravitational forces.

The classical approach has problems [4] associated with uncertainty of parameters, such as the mass of links can change when gripping objects; friction depends on wear, contamination or lubrication; temperature affects the elasticity and viscosity of components; the environment can introduce disturbances (e.g. sliding on a wet surface).

A hybrid approach is proposed where the model is split into two parts:

$$\ddot{q} = f_{phys}(q,\dot{q},\tau) + f_{ML}(q,\dot{q},\tau), \tag{3}$$

where $f_{phys}(q,\dot{q},\tau)$ is the physical model based on Lagrangian; $f_{ML}(q,\dot{q},\tau)$ is the ML-compensator, the model trained on data to compensate for errors in the physical model.

The error model is trained as follows:

$$\varepsilon(t) = \ddot{q}_{real}(t) - f_{phys}(q(t),\dot{q}(t),\tau(t)). \tag{4}$$

The goal is to train an ML model $f_{ML}(q,\dot{q},\tau) \approx \varepsilon(t)$. Architecture of neural networks: input - $q,\dot{q},\tau$, output - ε. Training is performed using data from a real robot or simulation + noise.

Here we show Python code with PyTorch (Fig. 1) as an example of the implementation of the ML-compensator based on a neural network that learns from data to compensate for errors in a physical model.

```
def generate_data(n_samples=1000):
    np.random.seed(0)
    q = np.random.uniform(-1, 1, n_samples)
    dq = np.random.uniform(-2, 2, n_samples)
    tau = np.random.uniform(-5, 5, n_samples)
    b_true = 0.5
    ddq_true = tau - b_true * dq
    ddq_phys = tau  # f_phys assumes b = 0
    error = ddq_true - ddq_phys  # это будет "цель" для ML
    X = np.vstack((q, dq, tau)).T
    y = error.reshape(-1, 1)
    return train_test_split(X, y, test_size=0.2)
# Neural network for f_ml
class MLCompensator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(3, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )
    def forward(self, x):
        return self.net(x)
# Machine Learning in process
def train_model(model, X_train, y_train, X_val, y_val, epochs=1000):
    optimizer = optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.MSELoss()
    train_losses, val_losses = [], []
    for epoch in range(epochs):
        model.train()
        inputs = torch.tensor(X_train, dtype=torch.float32)
        targets = torch.tensor(y_train, dtype=torch.float32)
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        model.eval()
        with torch.no_grad():
            val_inputs = torch.tensor(X_val, dtype=torch.float32)
            val_targets = torch.tensor(y_val, dtype=torch.float32)
            val_outputs = model(val_inputs)
            val_loss = criterion(val_outputs, val_targets)
        train_losses.append(loss.item())
        val_losses.append(val_loss.item())
        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Train Loss={loss.item():.5f}, Val
Loss={val_loss.item():.5f}")
    return train_losses, val_losses
# RUN
X_train, X_val, y_train, y_val = generate_data()
model = MLCompensator()
train_losses, val_losses = train_model(model, X_train, y_train, X_val, y_val)
# Screen
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Val Loss")
plt.xlabel("Epoch")
plt.ylabel("MSE Loss")
plt.legend()
plt.title("Training of ML Compensator (f_ml)")
plt.show()
# to use
def f_ml(q, dq, tau):
    x = torch.tensor([[q, dq, tau]], dtype=torch.float32)
    with torch.no_grad():
        return model(x).item()
# the acceleration error prediction
print("f_ml(q=0.1, dq=1.0, tau=2.0) =", f_ml(0.1, 1.0, 2.0))
```

Figure 1. Code of implementation of the ML-compensator.

The example is simplified for a one-dimensional system (e.g., a linear actuator), where the input data are: position q, velocity $\dot{q}$, control action τ, and the output is the acceleration error ε that needs to be compensated. This code generates training data with a known model error (e.g., friction was forgotten); trains the neural network to predict the acceleration error based on the current state and the control signal; implements the function f_ml(q, dq, tau), which can be used as part of a hybrid model $\ddot{q} = f_{phys}(q, \dot{q}, \tau) + f_{ML}(q, \dot{q}, \tau)$. The code can be adapted to a more complex model.

The advantages of the proposed approach include adaptability to changing conditions; improved trajectory prediction accuracy, and a more robust control system under real parameter deviations. Thus, the hybrid approach, which combines physically interpretable differential equations and machine learning techniques, allows us to build more robust, adaptive, and accurate models for robotics. This approach opens the way to more intelligent and autonomous control systems that can operate effectively under uncertainty.

References

1. W. Deng, F. Ardiani, Kh. T.P. Nguyen, M. Benoussaad, K. Medjaher. (2024). Physics informed machine learning model for inverse dynamics in robotic manipulators. Applied Soft Computing, Vol. 163, pp. 111877

2. J. S. Toquica, P. S. Oliveira, W. S.R. Souza, J. M. S.T. Motta, D. L. Borges. (2021). An analytical and a Deep Learning model for solving the inverse kinematic problem of an industrial parallel robot. Computers & Industrial Engineering, Vol. 151, pp. 106682.

3. S. E. Ivanov, T. Zudilova, T. Voitiuk, L. N. Ivanova. (2020). Mathematical Modeling of the Dynamics of 3-DOF Robot-Manipulator with Software Control. Procedia Computer Science, Vol. 178, pp. 311-319.

4. X. Wang, X. Liu, L. Chen, H. Hu. (2021). Deep-learning damped least squares method for inverse kinematics of redundant robots. Measurement, Vol. 171, pp. 108821