Robotics Final Report

Team 1

December 2025

1 Milestone 1

For Milestone 1, we started with the literature review. We chose a "Screwdriver-Fastening Robotic Manipulator" based on the research paper "Robotic Fastening with a Manual Screwdriver" by Ling Tang. Our choice was based on the simplicity of the robotic manipulator as well as its alignment with our course and project requirements. We finalized this milestone by choosing a suitable robotic model: KUKA iiwa-14.

2 Milestone 2

Milestone 2 mainly revolved around the forward and inverse position kinematics of our robotic manipulator. To fulfill this milestone, we started by deriving the Modified Denavit-Hartenburg (MDH) parameters of the robot, then we extracted the homogeneous transformation matrices of each link in order to obtain the final matrix that gives the full position and orientation of the robot's end effector in space. We then followed by solving forward kinematics by hand, each step at a time. We finalized this milestone by generating a simple GUI to kickstart the project's demonstration.

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 1: Homogeneous Transformation Matrix for Link i.

$$^0T_7 = A_1A_2A_3A_4A_5A_6A_7$$

Figure 2: Overall Forward Kinematics from Base to End-Effector.

3 Milestone 3

Upon extracting the position kinematics, we moved a few steps further for milestone 3. We started by extracting the forward velocity kinematics through the Jacobian matrix, linear velocity of the end-effector, the angular velocity, and the joint velocities. We then elevated our work and started evaluating the inverse velocity kinematics. Afterwards, we implemented trajectory tracking methods through Joint-Space Tracking, and Task-Space Tracking.

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = J(\mathbf{q})\,\dot{\mathbf{q}}$$

Figure 3: Forward Velocity Kinematics relating joint velocities to end-effector velocity.

$$\mathbf{v} = J_p(\mathbf{q})\,\dot{\mathbf{q}}, \qquad \boldsymbol{\omega} = J_r(\mathbf{q})\,\dot{\mathbf{q}}$$

Figure 4: Linear and angular velocity components of forward velocity kinematics.

$$\dot{\mathbf{q}} = J^{+}(\mathbf{q})\,\dot{\mathbf{x}}_{\text{des}}$$

Figure 5: Inverse Velocity Kinematics using the Moore–Penrose Pseudo-Inverse.

$$J^{+} = J^{\top}(JJ^{\top})^{-1}$$

Figure 6: Pseudo-inverse of the Jacobian matrix.

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_d + K_p(\mathbf{q}_d - \mathbf{q})$$

Figure 7: Joint-Space Trajectory Tracking Control Law.

$$\dot{\mathbf{q}} = J^{+}(\mathbf{q})\,[\dot{\mathbf{x}}_d + K_p(\mathbf{x}_d - \mathbf{x})]$$

Figure 8: Task-Space Tracking Control Law.

$$J^{+} = J^{\top}(JJ^{\top} + \lambda^2 I)^{-1}$$

Figure 9: Damped Least Squares (DLS) inverse for Jacobian singularity avoidance.

$$\ddot{\mathbf{x}} = J(\mathbf{q})\,\ddot{\mathbf{q}} + \dot{J}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$$

Figure 10: Acceleration-level relationship between joint and end-effector motion.

The work of this milestone could be summarized through the modes we implemented:

- fk_vel — Compute end-effector velocity from $(q, \dot{q})$
- ik_vel — Compute required $\dot{q}$ for desired end-effector motion

4 Milestone 4

For Milestone 4, we worked on trajectory planning. The first planning method was task-space planning which computes a Cartesian path for the robot's end-effector and uses an iterative Jacobian-transpose inverse kinematics algorithm. The joint-space trajectory planning works by linearly interpolating between initial and final joint configuration, producing a smooth motion without relying on inverse kinematics.

$$\begin{aligned} x(t) &= x_0 + r\cos(t), \\ y(t) &= y_0 + r\sin(t), \\ z(t) &= z_0, \end{aligned} \qquad \mathbf{x}_d(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}.$$

Figure 11: Circular task-space Cartesian trajectory for the end-effector.

$$\Delta \mathbf{q} = \alpha\, J^{\top}(\mathbf{q})\big(\mathbf{x}_d - \mathbf{x}\big),$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta \mathbf{q}.$$

Figure 12: Jacobian-transpose inverse kinematics iteration rule.

$$\Delta \mathbf{q} = \alpha\, J^{\top}(\mathbf{q})\big(\mathbf{x}_d - \mathbf{x}\big) \;+\; \beta\,\big(\mathbf{q}_{\text{pref}} - \mathbf{q}\big),$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta \mathbf{q}.$$

Figure 13: Jacobian-transpose IK with joint preference (regularization term).

$$\mathbf{q}(t) = (1 - s(t))\,\mathbf{q}_{\text{start}} + s(t)\,\mathbf{q}_{\text{goal}}, \qquad s(t) = \frac{t}{T}.$$

Figure 14: Joint-space linear interpolation trajectory between two configurations.

5 Milestone 5

Milestone 5 focused on implementing a motor-level controller for the 7-DOF KUKA iiwa-14 manipulator. The controller computes torque commands for each joint based on position error and error rate. We implemented both a standard PD controller as a fallback, and a fuzzy logic-based controller for adaptive control.

5.1 PD Control Fallback

The fallback PD control computes torque commands as:

$$\tau_i = K_p\, e_i + K_d\, \dot{e}_i,$$

where $e_i = q_{d,i} - q_i$ is the position error, $\dot{e}_i$ is the error rate, and K_p and K_d are proportional and derivative gains.

5.2 Fuzzy Logic Controller

The fuzzy controller maps joint errors and error rates to torque commands using fuzzy sets and a rule base. For each joint, the fuzzy membership functions for error e_i and delta error $\dot{e}_i$ are:

$$
\begin{aligned}
\mu_{\mathrm{NB}}(x) &= \text{trapezoidal},\\
\mu_{\mathrm{NM}}(x) &= \text{triangular},\\
\mu_{\mathrm{NS}}(x) &= \text{triangular},\\
\mu_{\mathrm{ZE}}(x) &= \text{triangular},\\
\mu_{\mathrm{PS}}(x) &= \text{triangular},\\
\mu_{\mathrm{PM}}(x) &= \text{triangular},\\
\mu_{\mathrm{PB}}(x) &= \text{trapezoidal}.
\end{aligned}
$$

The output torque membership functions are similarly defined over the range $[-100, 100]$.

5.3 Rule Base

A 7-by-7 rule table determines the fuzzy output based on error and delta error labels:

$$
\begin{bmatrix}
NB & NB & NB & NB & NM & NS & ZE\\
NB & NB & NM & NM & NS & ZE & PS\\
NB & NM & NS & NS & ZE & PS & PM\\
NM & NS & NS & ZE & PS & PS & PM\\
NM & NS & ZE & PS & PS & PM & PB\\
NS & ZE & PS & PM & PM & PB & PB\\
ZE & PS & PM & PB & PB & PB & PB
\end{bmatrix}
$$

5.4 Defuzzification and Command Computation

The fuzzy outputs are aggregated and defuzzified using the centroid method:

$$
u_i = \frac{\sum_{u=-100}^{100} u\, \mu_{\text{aggregated}}(u)}{\sum_{u=-100}^{100} \mu_{\text{aggregated}}(u)}.
$$

Finally, the torque command for each joint is scaled by the maximum torque allowed:

$$
\tau_i = \frac{u_i}{100}\, \tau_i^{\max}, \quad i = 1, \ldots, 7.
$$

5.5 Summary

The fuzzy controller improves motor-level performance by generating torque commands that adapt to the magnitude of error and its rate, while the PD fallback ensures stability if fuzzy control is disabled. The overall algorithm can be summarized as:

1. Compute joint error e_i and delta error $\dot{e}_i$
2. Fuzzify e_i and $\dot{e}_i$

3. Evaluate fuzzy rules to determine output membership
4. Defuzzify using the centroid method to obtain u_i
5. Scale u_i by $\tau_i^{\max}$ to get final torque τ_i

$$\tau_i = f_{\text{FLC}}(e_i, \dot{e}_i), \quad i = 1, \ldots, 7$$

Figure 15: Motor-level fuzzy control law for each joint.