Robot Arm Control System

Web app

[Task 3]

Mohamad Abdulmoula,R. Rama

ramamohameed9@gmail.com

Department of Software Engineer

Under the Supervision Of:

Eng Asmaa Duramae

July 28, 2025

Table of Contents

Glossary of Terms

Term	Definition
Django	A high-level Python web framework that encourages rapid development and clean, pragmatic design.
6-DOF (Degrees of Freedom)	Refers to six independent movements of a robotic arm, typically involving rotation and translation.
Pose	A specific set of motor angles that defines a particular position or configuration of the robotic arm.
Slider	A graphical user interface element used to select a numeric value by sliding a handle along a track.
API (Application Programming Interface)	A set of endpoints allowing external systems (like robots) to interact with the web application.
SQLite	A lightweight, file-based relational database system used for local data storage.
CSRF (Cross-Site Request Forgery)	A security threat that Django protects against by verifying request origins.
Admin Interface	Django’s built-in panel for managing database content visually.
ORM (Object-Relational Mapping)	A technique Django uses to interact with databases using Python objects instead of raw SQL queries.
Command	A structured instruction issued to the robot to execute a saved pose or action.

Project Overview

This project aims to develop a **web-based robotic arm control system** using the Django web framework. The system provides a user-friendly graphical interface to manually control a 6-DOF (degrees of freedom) robotic arm, save specific poses, and execute them on demand. Each pose consists of six motor angles, which can be adjusted through sliders and stored in a database for future use.

Planning

A. Objective:

To develop a web-based control interface for a 6-DOF robotic arm that allows users to:

a) Adjust motor angles via sliders
b) Save and manage multiple pose presets
c) Send "run" commands to the robot
d) Allow the robot to fetch and consume those commands

B. Purpose:

This system serves as a bridge between the user and a physical (or simulated) robotic arm. It allows both manual pose saving and real time robot polling, enabling educational robotics, industrial simulation, or remote robot control.

C. Tools Used: Django 5.2.4 ,SQLite3,Python 3.11 ,HTML/CSS

Requirements Analysis

A. Functional Requirements (Boilerplate Style)

ID	Requirement Description	Priority
FR1	The system shall allow users to adjust 6 motor sliders	High
FR2	The system shall allow users to save the current pose with a name	High
FR3	The system shall store saved poses in a database	High
FR4	The system shall display all saved poses in a table	High
FR5	The system shall allow users to delete a pose	Medium
FR6	The system shall allow users to run a pose as a command	High
FR7	The system shall expose an API for robot to fetch pending commands	High
FR8	The system shall expose an API for the robot to mark a command as complete	High

B. Non-Functional Requirements

ID	Requirement Description	Priority
NFR1	The system shall use SQLite (or any SQL DB) for data storage	Medium
NFR2	The system shall be built with a secure backend framework	High
NFR3	The system shall run on localhost and be compatible with ngrok	Medium
NFR4	The system shall support JSON-based robot interaction	High
NFR5	The system shall be lightweight and usable on low-resource machines	Medium

Implementation Phases

1. Project and App Setup:

- django-admin startproject robotarm
- python manage.py startapp arm
- Added 'arm' to INSTALLED_APPS in settings.py

2. Data Models (models.py):

- Created Pose and Command models to store motor angles and robot instructions.

3. Migrations:

- python manage.py makemigrations
- python manage.py migrate

4. Views (views.py):

- control_panel : renders UI template
- pose_save, pose_load, pose_delete : manage pose data
- pose_run : creates new command to run
- get_run_pose: API for robot to fetch command
- update_status : API to mark command as completed

5. Templates (control_panel.html):

- Created sliders for Motor 1-6
- Save, Reset, Run buttons with fetch requests
- Table to show saved poses with Load and Remove actions

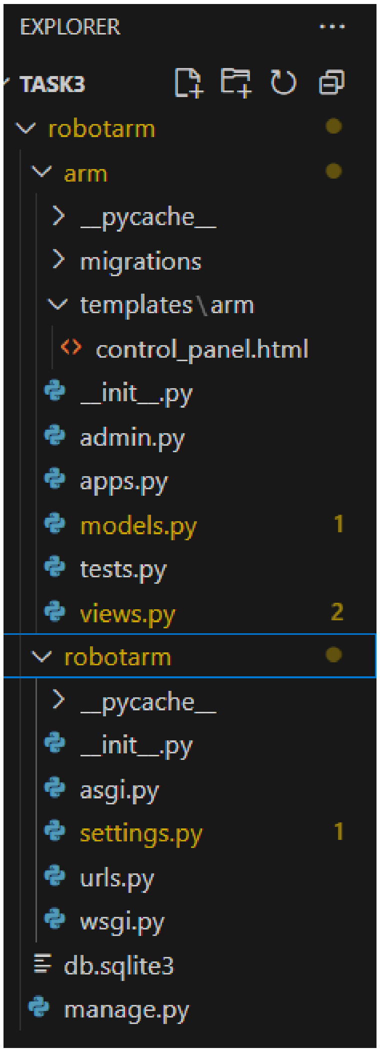
6. URLs (urls.py):

- Linked all views and APIs to proper routes

- Root route ("/") displays the control panel

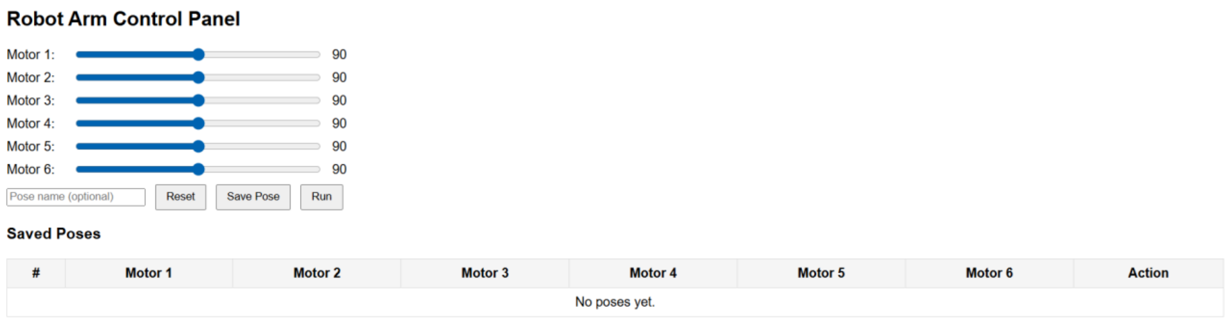
7. Admin Configuration (admin.py):

- Registered Pose and Command models



```
EXPLORER ...
TASK3
robotarm
  arm
    __pycache__
    migrations
    templates\arm
      control_panel.html
    __init__.py
    admin.py
    apps.py
    models.py        1
    tests.py
    views.py         2
  robotarm
    __pycache__
    __init__.py
    asgi.py
    settings.py      1
    urls.py
    wsgi.py
  db.sqlite3
  manage.py
```

Figure 1 Project directory structure for the robotic arm control panel web application built with Django, showing templates, views, models, and configuration files.



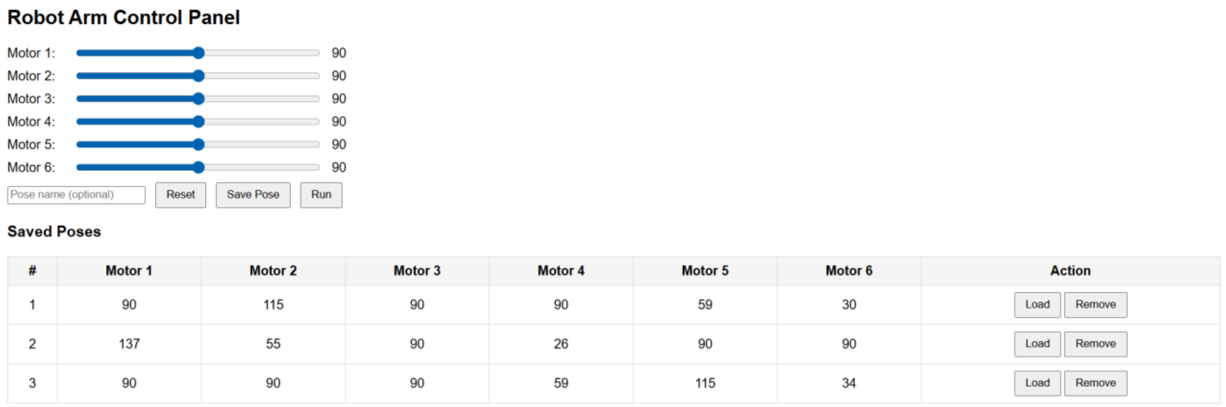
Robot Arm Control Panel

Motor 1: 90
Motor 2: 90
Motor 3: 90
Motor 4: 90
Motor 5: 90
Motor 6: 90

Saved Poses

#	Motor 1	Motor 2	Motor 3	Motor 4	Motor 5	Motor 6	Action
No poses yet.							

Figure 2 Web-based control panel interface displaying six motor sliders, input for pose name, and buttons for Reset, Save Pose, and Run, along with a dynamic table to display saved poses (currently empty).



Robot Arm Control Panel

Motor 1: 90
Motor 2: 90
Motor 3: 90
Motor 4: 90
Motor 5: 90
Motor 6: 90

Saved Poses

#	Motor 1	Motor 2	Motor 3	Motor 4	Motor 5	Motor 6	Action
1	90	115	90	90	59	30	Load Remove
2	137	55	90	26	90	90	Load Remove
3	90	90	90	59	115	34	Load Remove

Figure 3 Updated control panel interface displaying saved robotic arm poses in a table, with each pose containing six motor values and action buttons (Load, Remove) for execution or deletion.

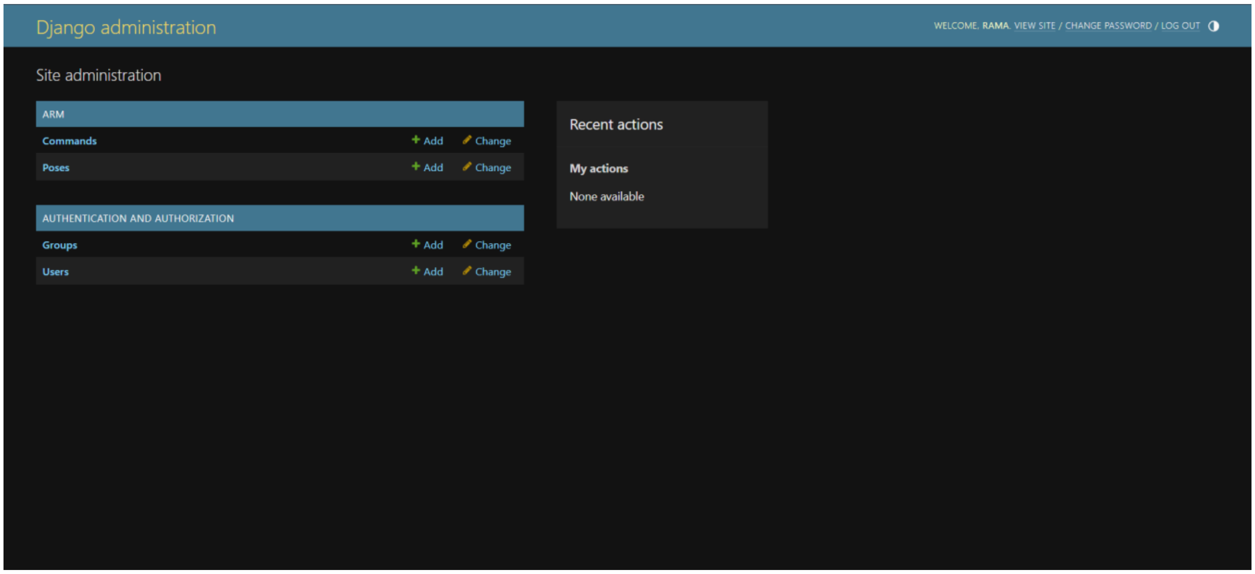


Figure 4 Django admin interface displaying registered models Commands and Poses under the ARM section, allowing administrative users to add, view, or modify robot-related data entries.

Conclusion

This project delivers a reliable web-based interface for robotic arm control with both manual (UI-based) and automated (API-based) operation modes. The decision to use Django provided a clean and robust architecture, allowing rapid implementation and future extensibility (authentication, logging, remote access).

References

A. Django Official Documentation
https://docs.djangoproject.com/en/5.2/

B. SQLite Documentation
https://www.sqlite.org/docs.html