Finite State Machine-Based Manipulator System for Teleoperation and Trajectory Control

Yohan park
20190258

***Abstract*—This project develops a finite state machine (FSM)-based manipulator system supporting teleoperation and inter-waypoint trajectory control. The system integrates solenoid-based actuation, linear interpolation for trajectory generation, and velocity mode commands via inverse kinematics (IK). A deterministic finite automaton (DFA) manages state transitions using keyboard inputs and joint state feedback. Real-time synchronization between an Ubuntu host and BeagleBone Black target ensures robust operation. The system demonstrates scalable teleoperation and waypoint navigation, showcasing FSM's practical application in distributed robotic control.**

***Keywords*—Finite State Machine (FSM), Manipulator Control, Trajectory Generation, Teleoperation, Deterministic Finite Automation (DFA).**

I. Introduction & Problem Definition

Finite state machines (FSM) enable precise control in robotic manipulators, particularly for tasks requiring navigation, teleoperation, and dynamic object interaction. FSM-based configurations offer deterministic state transitions, driven by user inputs or feedback, addressing limitations of traditional predefined or manual control approaches.

This project implements a FSM-based system combining teleoperation and inter-waypoint trajectory control. Using a deterministic finite automaton (DFA), state transitions are managed with joint state feedback between a host (Ubuntu) and target machine (BeagleBone Black). Trajectories are planned in configuration space (C-space) via linear interpolation and executed through inverse kinematics (IK) in velocity mode.

The system demonstrates scalable manipulator control, supporting task-specific state transitions and user-defined waypoints, offering a robust framework for distributed robotic systems.

II. Preliminaries

This project involves the design and implementation of a finite state machine (FSM)-based control system for a 4-degree-of-freedom (4-DoF) robotic manipulator. The system supports teleoperation and inter-waypoint trajectory control, ensuring seamless transitions between tasks through a deterministic framework.

A. System Configuration

The manipulator system is implemented using a dual-machine setup, where a host machine (Ubuntu) communicates with a target machine (BeagleBone Black). The host machine processes user inputs and relays commands to the target machine, which executes GPIO-based solenoid control and velocity mode commands. Real-time joint states are exchanged between the machines via UDP communication.

B. Manipulator Joint State Representation

The state of the 4-DoF manipulator is represented as:

$$q = [\theta_0\ \theta_1\ \theta_2\ \theta_3]^T$$

where θ_i denotes the angular position of each joint.

C. Finite State Machine Model

The FSM is modeled as a Deterministic Finite Automaton (DFA) defined by a quintuple $(\Sigma, \mathrm{S}, s_0, \delta, \mathrm{F})$, where Σ is set of input symbols (e.g., keyboard commands, joint state error thresholds), S is set of states $\{s_0,\ s_1,\ s_2,\ s_3\}$ representing Idle, Task 1, Teleoperation, and Task 3, respectively, s_0 is initial state, δ is transition function mapping $S\ \times \Sigma \rightarrow S$, and F is final states (Which is when Ctrl+C is pressed). Details will be in the methodology section.

D. Trajectory Generation

Trajectories are generated in configuration space (C-space) using linear interpolation between pre-defined waypoints. Actuator commands are derived from velocity mode outputs following inverse kinematics (IK) calculations.

E. Trajectory Generation

Users pre-define waypoints via a separate executable prior to FSM initiation. During teleoperation, keyboard inputs dictate state transitions, enabling fine-grained control of the end-effector for object manipulation.

These foundational elements provide the groundwork for the proposed FSM-based manipulator control system, ensuring precise and scalable task execution in a distributed environment.

III. Methodology

This project implements a finite state machine (FSM)-based control system for a 4-degree-of-freedom (4-DoF) robotic manipulator. The methodology focuses on integrating teleoperation and inter-waypoint trajectory control through deterministic state transitions, precise trajectory generation, and real-time communication between distributed components.

A. System Architecture

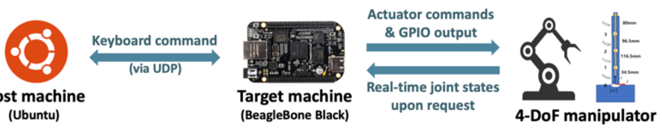


Fig. 1: Overall architecture of this project.

As in Fig. 1, the system consists of two main components, excluding the manipulator:

1) Host machine: Runs on Ubuntu and serves as the interface for user inputs. The host processes commands, such as keyboard inputs, and relays them to the target machine via UDP communication.

2) Target machine: Runs on BeagleBone Black and executes GPIO-based solenoid control for the manipulator. It processes actuator commands and manages joint state updates, enabling real-time operation.

B. FSM Configuration

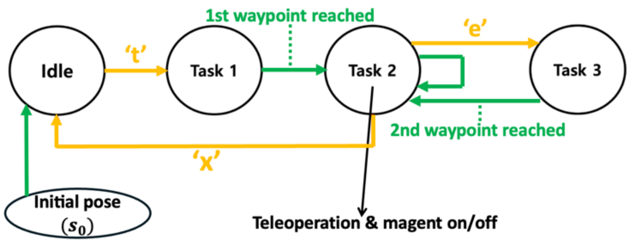


Fig. 2: Configuration of the FSM

The overall configuration of the FSM is shown in Fig. 2. It is modeled as a deterministic finite automaton (DFA) defined by a set of states and transitions:

- States: The FSM includes four states:

  (1) s_0: Initial pose (Idle state)
  (2) s_1: Task 1 – Move the end-effector (EE) to a pre-defined joint state near the target object.
  (3) s_2: Teleoperation mode with continuous user input. This state has self-loop, that let the user continue the teleoperation unless the keyboard input is present.
  (4) s_3: Task 3 – Move the EE to a pre-defined joint state near the basket.
- Transitions:
  In Fig. 2, there are two types of configurations; Transitions triggered by keyboard inputs (e.g., 'x', 't', 'e'), which is colored in orange color. And transitions triggered by the pose of the EE, which is colored in green color. Specifically, the criterion for the transition respect to the pose of the EE is $\left\|q - q_{target}\right\|_2 \leq \epsilon$.

C. Hardware implementation

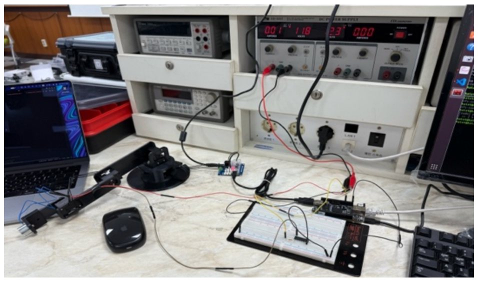


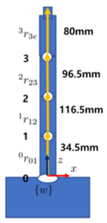
Fig. 3: Real-world hardware setup

The trajectory planning module operates in configuration space (C-space): User defines waypoints as pose data in a csv format file prior to FSM execution using a dedicated setup executable, and in the current project, two pre-defined waypoints and initial pose are used. Trajectories are generated between waypoints using linear interpolation, ensuring smooth transitions in joint space. For each pose commands, velocity mode commands are computed via inverse kinematics (IK), translating Cartesian space goals into joint space velocities for the manipulator.

The host and target machines communicate via UDP, ensuring low-latency data exchange for real-time operation. Joint state feedback from the target machine allows the host to monitor manipulator status and validate FSM conditions. The end-effector integrates a solenoid controlled via GPIO outputs on the target machine. This design enables binary on/off control for object manipulation tasks. Fig. 3 shows the whole hardware setup.

D. Inverse Kinematics (IK)

Assuming the goal position is very close to the current position in the task space, the following equation holds: $q_d - q = J^{-1}(q)(x_d - x)$. Hence, by setting the target joint q_d, the manipulator can be controlled in configuration space (C-space) using a joint space controller. As illustrated in Fig. 1, the manipulator features 4 degrees of freedom (DoF), comprising revolute joints. The axes of joints 1, 2, and 3 are parallel, and the end-effector is equipped with a solenoid magnet. The linear velocity of the end-effector results from the combined contributions of all 4 revolute joints. Consequently, the world linear velocity of the end-effector can be expressed as follows, with the red rectangle in the equation representing the Jacobian matrix for world linear velocity. This Jacobian matrix has dimensions 3 \times 4 , and its computation is shown in Fig. 4:



$$
\begin{aligned}
{}^wr_{3e} &= {}^wR_3\,{}^3r_{3e} \\
{}^wr_{2e} &= {}^wr_{3e} + {}^wR_2\,{}^2r_{23} \\
{}^wr_{1e} &= {}^wr_{2e} + {}^wR_1\,{}^1r_{12} \\
{}^wr_{0e} &= {}^wr_{1e} + {}^wR_0\,{}^0r_{01}
\end{aligned}
$$

$$
\begin{aligned}
{}^wV_e &= {}^w\Omega_0 \times {}^wr_{0e} + {}^w\Omega_1 \times {}^wr_{1e} + {}^w\Omega_2 \times {}^wr_{2e} + {}^w\Omega_3 \times {}^wr_{3e} \\
&= \boxed{\begin{bmatrix} {}^wP_0 \times {}^wr_{0e} & {}^wP_1 \times {}^wr_{1e} & {}^wP_2 \times {}^wr_{2e} & {}^wP_3 \times {}^wr_{3e} \end{bmatrix}} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \\
&= J(q)\dot{q}
\end{aligned}
$$

Fig. 4: Computation process of the Jacobian matrix J

IV. Results & Conclusion

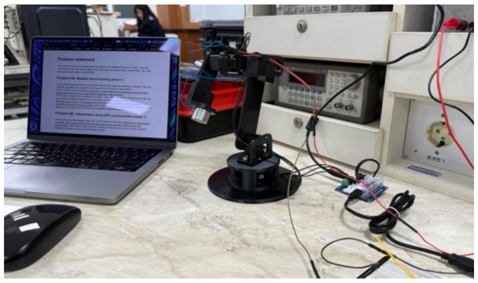


Fig. 5: Testing the setup

The whole FSM system worked as expected, but the wire connecting the target machine and the solenoid magnet was disconnected. Even though I couldn't show the magnet working in the final demo. Fig. 5 shows the testing of the system.