Project Flow

The chosen Industrial Robotic Application is a Spraying Robotic Arm.

It shall be completed in 5 milestones with each milestone going as follows:

**Milestone 1:**

**Literature Review on Industrial Robotics Applications:**

Conduct a thorough **literature review** of 5-10 recent papers (published between 2015 and the present) that focus on **industrial robotics applications**.

Review various applications of industrial robots, such as painting, welding, material handling, assembly, and more.

Summarize key insights, technological advancements, challenges, and trends from the literature review.

**Select an Industrial Robotic Application:**

Based on the literature review, **select a specific application** for the project. The selection should be aligned with industry relevance and feasibility for a desktop robotic manipulator.

State the **chosen application**, explaining why it was selected and how it addresses a particular industrial need.

**Propose a Draft Project Flow:**

Develop a **draft project flow** that serves the chosen industrial robotic application. The flow should outline the stages of development, from design and simulation to control and testing.

This draft flow will guide the upcoming phases and ensure that the project remains focused on achieving the objectives of the selected application.

**Search for Available GrabCAD Models:**

Search for available robotic manipulator models on **GrabCAD** or other CAD platforms. Focus on **desktop robotic manipulators** with **3-4 degrees of freedom** (DOF), corresponding to the number of motors.

Choose a model that can be used for simulation and further development within the project.

**Download and Install Simulation Software:**

Download and install **CoppeliaSim** software.

**Set Up GitHub Repository for Team Collaboration:**

Create a **GitHub repository** for the project, where all code, documents, and designs will be stored and shared. Set up **one repository per team** to centralize collaboration and version control.

Include necessary documentation and instructions for using the repository and collaborating efficiently.

**Milestone 2:**

**Frame Assignment on the Robotic Arm:**

Assign coordinate frames to each joint of the robotic arm. This can be done either manually on paper by visualizing the arm or through the frame assignment tools in the robotic simulator being used.

Define the position and orientation of each link relative to its neighboring links. This will lay the foundation for calculating the kinematics.

**Develop the DH Convention of the Robotic Arm:**

Develop the DH parameters for the robotic arm to define the kinematic chain systematically.

Ensure that each joint's rotation and translation parameters are accurately represented using the DH convention.

**Obtain Forward and Inverse Position Kinematics:**

Use Python to derive the **Forward Kinematics** (position and orientation of the end effector based on joint variables) and **Inverse Kinematics** (joint variables required to achieve a specific end effector position).

Validate the results through simulation or numerical computation.

**Integrate CAD Assembly into Simulation Environment:**

Import the CAD assembly of the robotic arm into the chosen simulation environment.

Use linking tools (e.g., Solidworks and MATLAB/Simulink integration or URDF files in CoppeliaSim) to ensure the CAD model is accurately transferred and represents the physical robot.

**Motion Simulation & Analysis:**

Actuate the robotic arm’s joints in the simulation environment to analyze its motion. Use constant inputs, signal builders, or sliders to drive the joints and observe the resulting limb movement.

Visualize the motion of the robotic arm in 3D to understand its movement capabilities and limitations.

**Start Building the Graphical User Interface (GUI):**

Begin creating the GUI for interacting with the robot simulation. This should allow users to control joint movement, visualize kinematics, and display essential data during simulation.

Ensure that the GUI framework is compatible with the chosen programming language (Python) and integrates smoothly with the simulation environment.

**Milestone 3:**

**Derive Forward and Inverse Velocity Kinematics:**

Formulate the **Forward Velocity Kinematics** equations to describe how the end-effector's linear and angular velocity are derived from the joint velocities.

Develop the **Inverse Velocity Kinematics** equations to determine the required joint velocities based on the desired velocity of the end effector.

Implement these equations in **Python** to compute the velocities of the robotic arm.

**Derive Forward and Inverse Acceleration Kinematics:**

Derive the **Forward Acceleration Kinematics** to relate joint accelerations to the end-effector's acceleration.

Develop the **Inverse Acceleration Kinematics** to determine the necessary joint accelerations for achieving specific end-effector acceleration.

Implement the equations in **MATLAB/Simulink** or **Python** for both forward and inverse acceleration calculations.

**Test the Kinematic Equations in Simulation:**

Validate the derived **Position, Velocity, and Acceleration Kinematic** equations using the simulation environment (CoppeliaSim).

Test the equations by inputting joint angles, angular velocities, and angular accelerations into the simulator and comparing the resulting position, velocities, and accelerations of the end effector with the calculated values from the kinematic equations.

Ensure that the results are within an acceptable tolerance range for both the forward and inverse kinematic solutions.

**Continue Building the GUI for the Simulation Environment:**

Enhance the GUI by incorporating all environmental components required for the industrial sprayer robotic arm scenario, including objects like chairs, conveyors, people, tools, and other relevant surroundings.

Ensure the GUI allows interaction with both the robotic arm and its environment, such as initiating tasks, setting trajectories, and visualizing the robot's motion.

**Derive Task-Space Trajectories:**

Formulate **Task-Space Trajectories** for the robotic arm’s end effector to follow specific paths in the workspace, such as straight lines or curved paths.

Implement these trajectories in **Python** to define the desired position, velocity, and acceleration profiles for the end effector in task space.

**Derive Joint-Space Trajectories:**

Develop **Joint-Space Trajectories** for the robotic arm’s individual joints to follow based on the desired motion of the end effector.

Implement the joint-space trajectories in **Python** by computing the joint angles, velocities, and accelerations required to follow the task-space trajectory.

Ensure smooth and continuous movement by planning the trajectory profiles appropriately.

**Milestone 4:**

**Validate Simulated Task-Space Trajectories:**

Test and validate the **Task-Space Trajectories** in the simulation environment (CoppeliaSim).

Ensure that the end effector of the robotic arm accurately follows the planned path and achieves the desired positions, velocities, and accelerations within the specified tolerance.

Analyze and fine-tune the trajectory to account for any discrepancies between the planned and simulated motion.

**Validate Simulated Joint-Space Trajectories:**

Similarly, validate the **Joint-Space Trajectories** by ensuring that each joint of the robotic arm follows the desired angles, velocities, and accelerations as planned.

Compare the simulated motion to the expected joint behavior and adjust if necessary to improve accuracy and performance.

Document any adjustments needed to better align the joint movements with the overall task-space trajectory.

**Send Joint Angles to Perform Task-Space and Joint-Space Trajectories:**

Once the task-space and joint-space trajectories have been validated, the **joint angles** required to execute the trajectories will be sent to the simulation environment via the GUI.

Implement a process within the GUI that sends the calculated joint angles and commands to the simulator, initiating the **desired motion** of the robotic arm.

Ensure that the GUI allows real-time control and monitoring of the robotic arm’s motion as it follows the task-space and joint-space trajectories.

**Start Coupling with the MCTR Team (if needed)**

**Milestone 5:**

**Derive Position Control Algorithms (Low-Level Motor Control):**

Develop **position control algorithms** at the motor level, using **PID control** to manage the robotic arm’s joint positions accurately.

Implement the **PID control system** in **Python**, allowing precise control over each joint of the robotic arm by adjusting the motor inputs.

Tune the PID controller parameters (Proportional, Integral, and Derivative gains) to minimize error in joint position control.

**Test the Closed-Loop System in the Simulation Environment:**

Integrate the PID-controlled joints into the simulation environment (CoppeliaSim) to create a **closed-loop control system**.

Visualize the robotic arm’s motion and response to the position control system in real-time using the simulation environment.

Test the **closed-loop system** for stability, accuracy, and performance under varying conditions, adjusting the controller parameters as needed to achieve smooth and accurate joint control.

**Develop a Digital Twin:**

Collaborate with MCTR team to develop a **digital twin** that mirrors the real-world hardware in the simulation environment.

Use real-time data from the physical robotic arm created by the MCTR team to synchronize the digital model in the simulator.

Ensure that the digital twin accurately replicates the behavior of the physical system, allowing for advanced testing and monitoring before applying control commands to the actual hardware.

**Perform the Desired Industrial Application:**

Once the control algorithms are validated, execute the **desired industrial application** using the robotic arm.

Utilize the simulation environment to ensure that the robotic arm performs its tasks with precision and reliability.

Use the GUI to control the application in real-time, ensuring that the position control system can handle the operational demands of the industrial setting.

**Develop Force Control Algorithm for the Robotic Manipulator (BONUS):**

As an advanced or **bonus task**, develop a **force control algorithm** for the robotic manipulator.

Implement the force control system in **Python**, ensuring that the robotic arm can apply and regulate force during interactions with the environment or objects.

Combine force control with position control to allow the robotic arm to perform tasks that require both precise movement and delicate force application (e.g., assembly, polishing, or handling fragile materials).