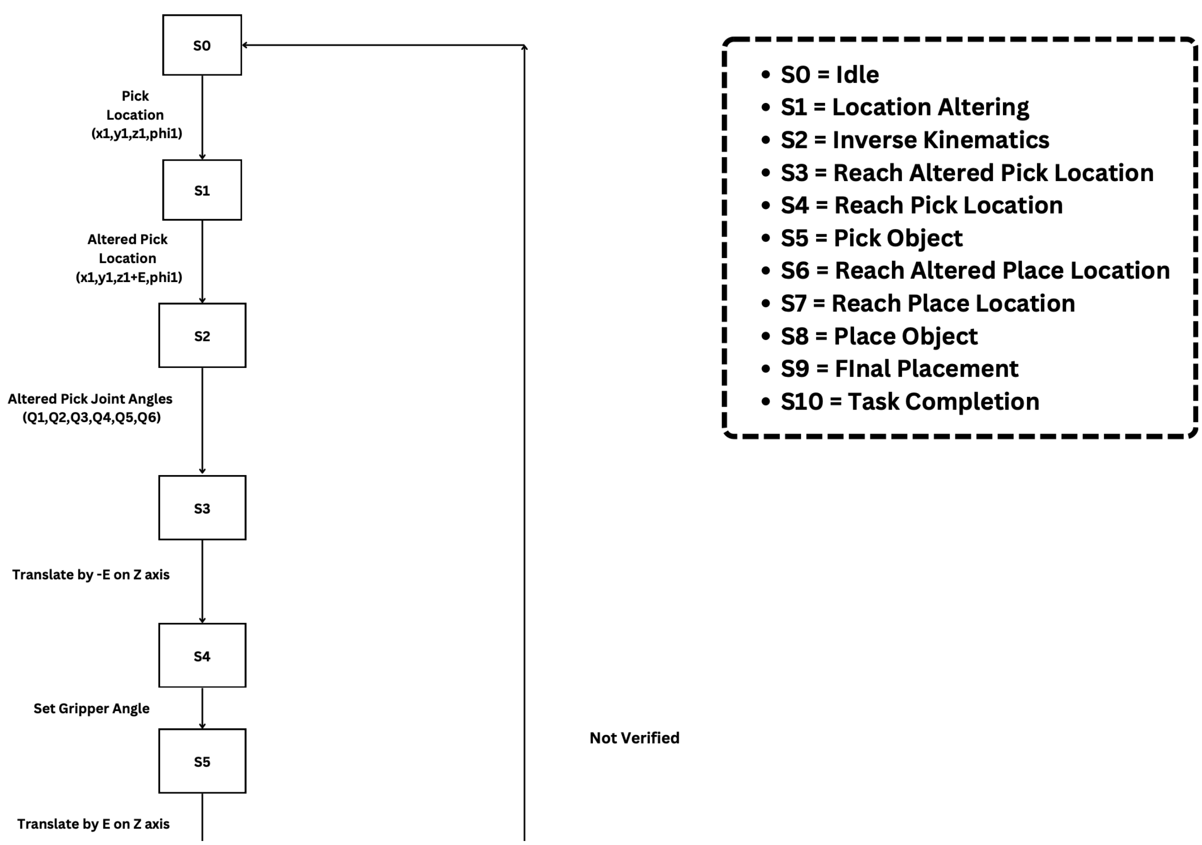
Introduction To Robotics

Lab 07

Huzaifah Tariq Ahmed - ha07151
Syed Asghar Abbas Zaidi - sz07201
Daniyal Rahim Areshia - d07605

10th May 2024

1 Task 7.1 - FSM Diagram



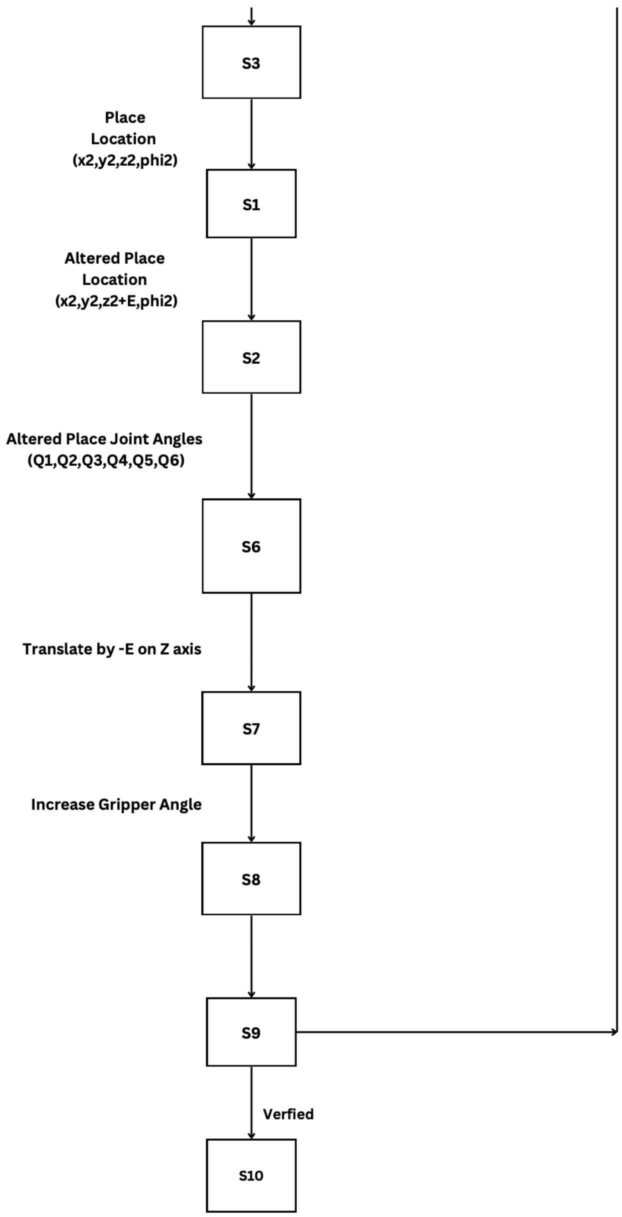


Figure 1: FSM Diagram of our Motion Control System

2 Task 7.2 - FSM Implementation

2.1 Pick and Place

```
function pick_place(pick_coords, place_coords)
    % Initialize current state
    current_state = 0;

    % Offset in z-coordinates for pick and place locations
    pick_coords(3) = pick_coords(3) + 2;
    place_coords(3) = place_coords(3) + 2;

```

```
    % Height of the block (in cm)
    height_of_block = 2.8;

    % Main loop for pick and place process
    while true
        % Pick phenomenon
        current_state
        if current_state == 0
            % Move to initial pick position
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            arb.setpos([0, 0, -pi/4, -pi/2, 0], [25, 25, 25, 25, 25]);
            pause(5);
            error_ = error_estimator([0, 0, 0, 0, 0], [arb.getpos]);
            if error_ == false
                current_state = 1;
            else
                current_state = 0;
            end
            current_state
        elseif current_state == 1
            % Adjust pick position and find optimal solution
            pick_coords(3) = pick_coords(3) + 2 * height_of_block;
            coords_1 = findOptimalsoln(pick_coords(1), pick_coords(2),
               pick_coords(3), pick_coords(4), pick_coords(5));
            current_state = 2;
            current_state
        elseif current_state == 2
            % Move to adjusted pick position
            setPosition(coords_1, 0);
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            curr_pos = arb.getpos;
            error_1 = error_estimator([coords_1, 0], [curr_pos]);
            if error_1 == false
                current_state = 3;
            else
                current_state = 2;
            end
            current_state
        elseif current_state == 3
            % Lower the gripper and find optimal solution
            pick_coords(3) = pick_coords(3) - 4 * height_of_block;
            coords_2 = findOptimalsoln(pick_coords(1), pick_coords(2),
               pick_coords(3), pick_coords(4), pick_coords(5));
            setPosition(coords_2, 0);
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
```

```
            curr_pos = arb.getpos();
            error_1 = error_estimator([coords_2, 0], [curr_pos]);
            if error_1 == false
                current_state = 4;
            else
                current_state = 3;
            end
            current_state
        elseif current_state == 4
            % Raise the gripper and check if cube is picked
            setPosition(coords_2, 1.2);
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            if arb.getpos(5) > 0.9
                current_state = 5;
            else
                current_state = 4;
            end
        % Place phenomenon
        current_state
        elseif current_state == 5
            % Adjust place position and find optimal solution
            place_coords(3) = place_coords(3) + 2 * height_of_block;
            coords_3 = findOptimalsoln(place_coords(1), place_coords(2)
               , place_coords(3), place_coords(4), place_coords(5));
            current_state = 6;
            current_state
        elseif current_state == 6
            % Move to adjusted place position
            setPosition(coords_3, 1.2);
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            curr_pos_ = arb.getpos;
            error_3 = error_estimator([coords_3, 1.1556], [curr_pos_]);
            if error_3 == false
                current_state = 7;
            else
                current_state = 6;
            end
            current_state
        elseif current_state == 7
            % Lower the gripper and find optimal solution
            place_coords(3) = place_coords(3) - 4 * height_of_block -
               1;
            coords_4_ = findOptimalsoln(place_coords(1), place_coords
               (2), place_coords(3), place_coords(4), place_coords(5));
            setPosition(coords_4_, 1.2);
```

```
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            curr_pos = arb.getpos();
            error_4 = error_estimator([coords_4_, 1.2], [curr_pos]);
            if error_4 == false
                current_state = 8;
            else
                current_state = 7;
            end
            current_state
        elseif current_state == 8
            % Raise the gripper and reset state to 0
            setPosition(coords_4_, 0.8);
            pause(5);
            arb = Arbotix('port', 'COM18', 'nservos', 5);
            current_state = 0;
        end
    end
end
```

2.2 Error Estimator - Helper Function

```
% Function to compare actual angles with motor angles and determine
   error margin
function margin_error = error_estimator(actual_angles, motor_angles)
    % Initialize margin_error flag to false
    margin_error = false;

    % Loop through each angle (assuming 5 angles)
    for k = 1:5
        % Check if the absolute difference between motor angle and
           actual angle is less than 0.05
        if abs(motor_angles(k) - actual_angles(k)) < 0.05
            % If within margin, set margin_error to false
            margin_error = false;
        else
            % If outside margin for any angle, set margin_error to true
                and exit loop
            margin_error = true;
            break; % Exit loop since error detected
        end
    end
end
```

2.3 Strategy

The provided code implements a pick and place strategy for a Phantom X Pincher robotic arm, orchestrating a sequence of actions to grasp an object and then release it at different locations. The function pick place initiates the process by setting the initial state and adjusting the z-coordinates for pick and place positions. It employs a loop to iterate through the pick and place phases continuously. In the pick phase, the arm maneuvers to an initial pick position, refines the pick position, lowers the gripper, and verifies successful pick-up. If the pick-up is confirmed, the code transitions to the place phase, where it optimizes the place position, lowers the gripper again, and ensures accurate placement. Throughout this loop, the code adjusts the arm's movements based on feedback, optimizing the process of manipulating objects with precision.

2.4 Best Execution Video

Click to Watch the execution video

2.5 Points of Improvement

While the code effectively outlines the pick and place strategy, several areas could benefit from improvement. Firstly, **enhancing error handling mechanisms** would fortify the code against unexpected scenarios like communication glitches or sensor malfunctions, ensuring robustness in operation. Secondly, **optimizing the code** for efficiency by streamlining calculations and reducing unnecessary pauses could enhance the overall speed and responsiveness of the robotic arm. Thirdly, **further modularizing** the code into distinct functions for movement, error management, and position adjustments would improve code readability and maintenance. Lastly, integrating more accurate kinematic calculations, such as forward and inverse kinematics, would elevate the arm's accuracy and reliability during pick and place maneuvers, enhancing its performance in various tasks.

Additionally, **Implement Machine Learning Algorithms** that could enable the robot to learn from experience and adapt its pick and place strategy over time. This could involve collecting data on performance metrics and using this data to continuously improve the robot's behavior through reinforcement learning or other adaptive techniques. **Addition of Path Planning and Object Detection/Avoidance**, advanced path planning algorithms can optimize trajectories, ensuring smooth, collision-free movements while minimizing energy consumption. Integrating object recognition using 3D cameras can enable the robot to adapt its strategy based on object characteristics, improving efficiency and versatility in handling various items.

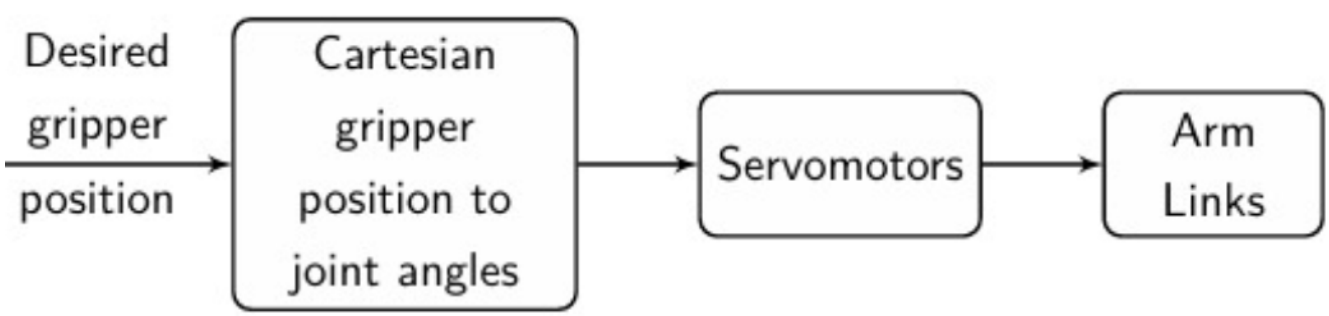


Figure 2: Block Diagram of our Motion Control System

Addition of Closed-loop control, Note that this block diagram indicates that our system is not operating in a closed loop, i.e. there is no feedback being obtained from the camera about the positioning of the gripper. We can also implement a complete visual feedback-based closed loop, but we'll operate our system in the open-loop configuration of Figure 2 for now.

Github Repository Link, click here