Introduction to Robotics

Lab Instructor: Miss Sadaf Sheikh
Group Members: Afsah Hyder, Ailiya Fatima, Maleeha Hussain
Course Instructor: Dr Basit Memon

LAB 07-Motion Control & Velocity Kinematics

April 13, 2024

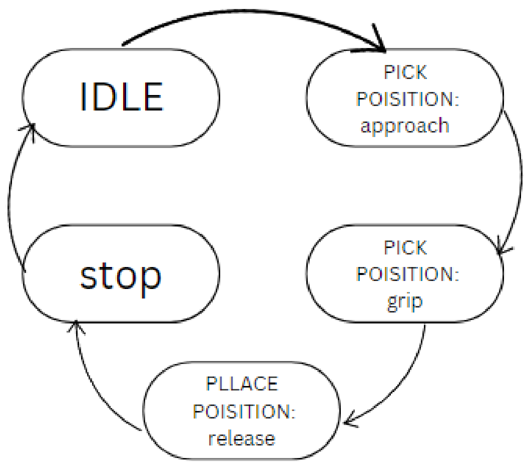
Contents

1 7.1: FSM **3**

2 7.2: FSM Implementation **4**

2.1 Explaination of the strategy 4

2.2 Functions used 5



1 7.1: FSM

The following is a state space model for the pick and place operation where we have 5 main states fn the robot arm as shown above.

- Idle: The robot arm is in its initial position, ready to receive instructions.
- Move to Pick-Up Location: The robot arm receives a command with the pick-up location coordinates. It then plans and executes the movement trajectory to reach the designated location. Sensors might be used for obstacle avoidance during movement.
- Approach Pick-Up Location: Upon reaching the pick-up location, the robot arm slows down and carefully approaches the target object. Sensors like proximity sensors provide feedback on the distance to the object. The gripper closes around to secure the object. Sensors or motor current monitoring might be used to confirm successful grasping.
- Move to Place Location: The robot arm receives a command with the placement location coordinates. It plans and executes the movement trajectory to reach the designated location. Sensors are again used for obstacle avoidance. Similar to state 3, the robot arm carefully approaches the placement location with the grasped object. The gripper opens to release the object at the desired position.
- Return to Idle: The robot arm returns to its initial position, ready for the next task.

This was the initial mindmapping for FSM. We made a slight change while implementing it.

2 7.2: FSM Implementation

Video of the execution: Video

2.1 Explaination of the strategy

The Pick and Place function automates the process of picking up an object from one location and placing it in another. Here's a breakdown of its potential functionalities:

Inputs and Outputs:

Inputs: The function takes the desired pick and place coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) as input.

Outputs: It outputs a success/failure status of the movement or an error message if the placement location is unreachable.

Function Flow:

Initialization:

- The function establishes a connection with the robot arm (Arbotix). - It defines the robot's idle state configuration (*idle*) and initializes flags (*move* and *task_end*) to track movement status and task completion.

Coordinate Validation:

- The function calculates the distance to the placement location ($place\_r$). - It checks if this distance is within the robot's reach (r). - If valid: The function sets a flag (*move*) indicating a valid movement. - If invalid: It displays an error message (optional) and sets *move* to false, halting further execution.

State Machine Loop:

The core functionality relies on a loop that continues as long as *move* is true.

Inside the loop:

- The function retrieves the current robot position using *getpos*.

State Transitions:

- A variable (*state*) determines the current action:

- **State 0 (Idle):** The robot moves to its idle configuration and waits for user input (optional). Upon receiving user input, it transitions to state 1.
- **State 1 (Approach - Pick):** The function uses inverse kinematics (*findOptSolution*) to calculate joint angles for a position above the pick location. The robot then moves to this calculated position.
- **State 2 (Grasp - Pick):** The function uses IK again to calculate the joint angles for the exact pick location and commands the robot to move down. Additionally, it closes the gripper using a specific *setpos* command.
- **State 3 (Approach - Place):** IK is used to determine the joint angles for a position above the desired placement location (with an offset for hovering). The robot moves to this calculated position.
- **State 4 (Release Object):** IK calculates joint angles for the exact place location. The robot moves down, opens the gripper using a specific *setPos* command, and returns to the idle state. The loop terminates, and *task_end* is set to true.

Loop Exit:

Once the object is placed, *move* is set to false, and a success message might be displayed (optional).

2.2 Functions used

```
function move = PickandPlace(x1,y1,z1,x2, y2, z2)

%the function is defined in such a way that it takes 6 inputs from the users where x1, y1,z1 are the pick coordinates whereas the x2, y2, z2 arethe place coordinates. We fix phi in both cases to be -pi/2 (Since that's how the gripper will be positioned to pick or place any object).

    %Idle State Defined
    idle = [pi/3 pi/3 pi/3 pi/3];
    arb = Arbotix('port','COM8', 'nservos',5);

    task_end = 0;
    move = false; % Fix variable initialization and change to boolean type
    state = 0; %here state 0 refers to the idle state
    a2 = 10.8; a3 = 10.8; %specified in cm -> also we have made a correction from prev labs (initially it was in mm)
    r = a2 + a3;  %arm fully extended config.

    %according to our fsm we need to verify if the coordinates entered by the user are valid so we run a check
    place_r = sqrt(x2^2 + y2^2)
    pick_r = sqrt(x1^2 + y1^2)

    %Check Coordinate State
    if (place_r <= r)
        disp('Coordinates are Verified');
        move = true; % Use boolean type for move, valid movement
    else
        disp('Object placed outside the range')
    end

    while move

        curr_pos = arb.getpos(); %get current position
        if (state == 0)
            setPos(arb, idle)
            pause(3)
            arb.setpos(5,0,65);
            pause(5)
            state = 1; %here we change the state to go to Approach-Pick State
            if (task_end == 1)
                setPos(arb, idle)
                pause(5);
                arb.relax()
                disp('Task Complete!')
                move = false;
            end

```

```
        %Approach-Pick State Defined
        elseif (state == 1)
            angles = findOptSolution([x1, y1, z1 + 5, -pi/2]); %hovers over the object, find the joint Angles through Inverse kinematics
            setPos(arb, angles); %setPos was defined in Lab 4 where we incorporated offset according to the real world
            pause(3);
            state = 2; %changing the state to Grasp-Pick State

        %Grasp-Pick State defined
        elseif (state == 2)
            angles_grasp = findOptSolution([x1, y1, z1, -pi/2]); %find jointAngles through Inverse kinematics for the pick point
            setPos(arb, angles_grasp); %we were just above the point and now we reach downwards
            arb.setpos(5, 1.56,64); %here we open the grasper to grasp the object, set after trial n error
            pause(3);
            state = 3;

        %Approach-Place defined
        elseif (state == 3)
            angles_place = findOptSolution([x2, y2, z2 + 5, -pi/2]);%find jointAngles through Inverse kinematics for the place point with an offset (to hover over the point)
            setPos(arb, angles_place);
            pause(3);
            state = 4;

         %Release Object State defined
        elseif (state == 4)
            angles_release = findOptSolution([x2, y2, z2, -pi/2]); %find jointAngles through Inverse kinematics for the place point
            setPos(arb, angles_release)
             arb.setpos(5,0,64)
              %we return back and state in idle state
             state = 0;
             task_end = 1;
        end
    end
end
```

We made some changes of the errorCode and set position functions of the previous labs, to make it more coherent and merge them into a ingle function instead of dealing with several small functions. This function ‘setPos’, checks if the angles are in range, converts to radians and converts to servo angles.

```

function errorCode=setPos(arb,jointAngles_deg)

% acccepts joint angles of Phantom X Pincher as argument, and sets them as goal positions for
% the respective motors in the arm.

% accepts angle in degrees for easy visualization
% the input is wrt to the DH so we convert it to servo angles
  jointAngles_deg(2) = jointAngles (2) - 90;

  % now check if any of the angles is outside [-150, 150]
  errorval = 0
  if any(jointAngles_deg > 150) || any(jointAngles_deg < -150)
        error('Vector contains values outside the range [-150, 150].');
        errorval = 1
  end

  if errorval==0
    jointAngles_rad  = arb.getpos();
     jointAngles_rad(1) = jointAngles_deg(1)* pi/180
     jointAngles_rad(2) = jointAngles_deg(2)* pi/180
     jointAngles_rad(3) = jointAngles_deg(3)* pi/180
     jointAngles_rad(4) = jointAngles_deg(4)* pi/180

     % Connecting to Robot and passing the theta information to Robot for execution with a certain speed (64 for every joint in this case)

        disp('The servo joint angles in radians are \n');
        jointAngles_rad

        arb.setpos(jointAngles_rad, [64, 64, 64, 64, 64]);
  end
end
```