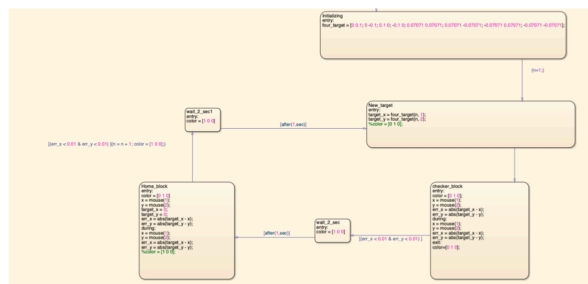
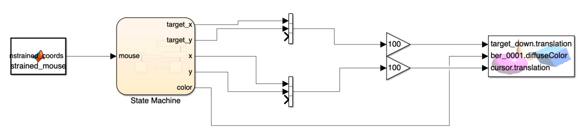
BioMedical Assignment_3 Report

Reaching Task:

The reaching task was initially implemented using **mouse coordinates**. A MATLAB function was used to extract the **X and Y positions** of the mouse, which were then sent to a **Stateflow chart**. A simple **absolute error function** guided the cursor toward the target circle. Once the target was reached, the **circle changed color** and moved to a **new position** or returned to the **home position**.

```
function constrained_coords = get_constrained_mouse_coords()
    % Declare 'get' as extrinsic to run in MATLAB workspace
    coder.extrinsic('get');

    % Initialize output to avoid size/type mismatch
    constrained_coords = [0.0, 0.0]; % Initialize as a [1x2] double vector

    % Call the extrinsic function to get pointer location
    pointer_coords = [0, 0]; % Initialize to avoid simulation issues
    pointer_coords = get(0, 'PointerLocation');

    % Cast mxArray output to double to ensure compatibility
    pointer_coords = double(pointer_coords);

    % Scale pointer coordinates to normalized values (example: [-0.1, 0.1])
    screen_width = 1440;  % Replace with your screen width or x - axis
    screen_height = 900; % Replace with your screen height or Y - axis

    % Normalize to [-1, 1]
    x_norm = (pointer_coords(1) - screen_width / 2) / (screen_width / 2);
    y_norm = (pointer_coords(2) - screen_height / 2) / (screen_height / 2);

    % Scale to [-0.1, 0.1] range
    x_scaled = x_norm * 0.1;
    y_scaled = y_norm * 0.1;

    % Constrain to [-0.1, 0.1]
    constrained_x = max(-0.1, min(0.1, x_scaled));
    constrained_y = max(-0.1, min(0.1, y_scaled));

    % Output constrained coordinates
    constrained_coords = [constrained_x, constrained_y];
end
```





In the next phase, the MATLAB function was replaced with a **Phantom robot**. Instead of using mouse coordinates, the robot's **Cartesian position (X, Y)** controlled the cursor's movement, allowing for direct, real-time interaction with the reaching task.

Force Field

Two types of force fields were implemented:

1) Attractive Force Toward the Target

An **attractive force field** was designed to pull the end-effector toward a predefined target position. This was achieved using a stiffness matrix KpK_p, which generates a force proportional to the distance between the current position and the target.

$$Fatt = -Kp(p - ptarget)F_{att} = -K_p(p - p_{target})$$

2) Viscous Damping Force (Velocity-Based)

This force opposes the velocity of the end-effector, slowing down movement proportionally to the velocity.

$$Fdamp = -KvvF_{damp} = -K_v v$$

Computing the Torque for the Phantom

Since the Phantom operates with **torque inputs**, the computed forces attractive force and viscus damping force must be converted into joint torques using the **Jacobian Transpose method**:

$$\tau = JT(Fatt + Fdamp)\tau = J^T(F_{att} + F_{damp})$$

After setting up the **force fields** in Simulink, modifying the **gain values** of the **stiffness matrix (Kp)** and **damping matrix(Kv)** leads to different effects on the **elasticity** and **repulsiveness** of the Phantom's motion.