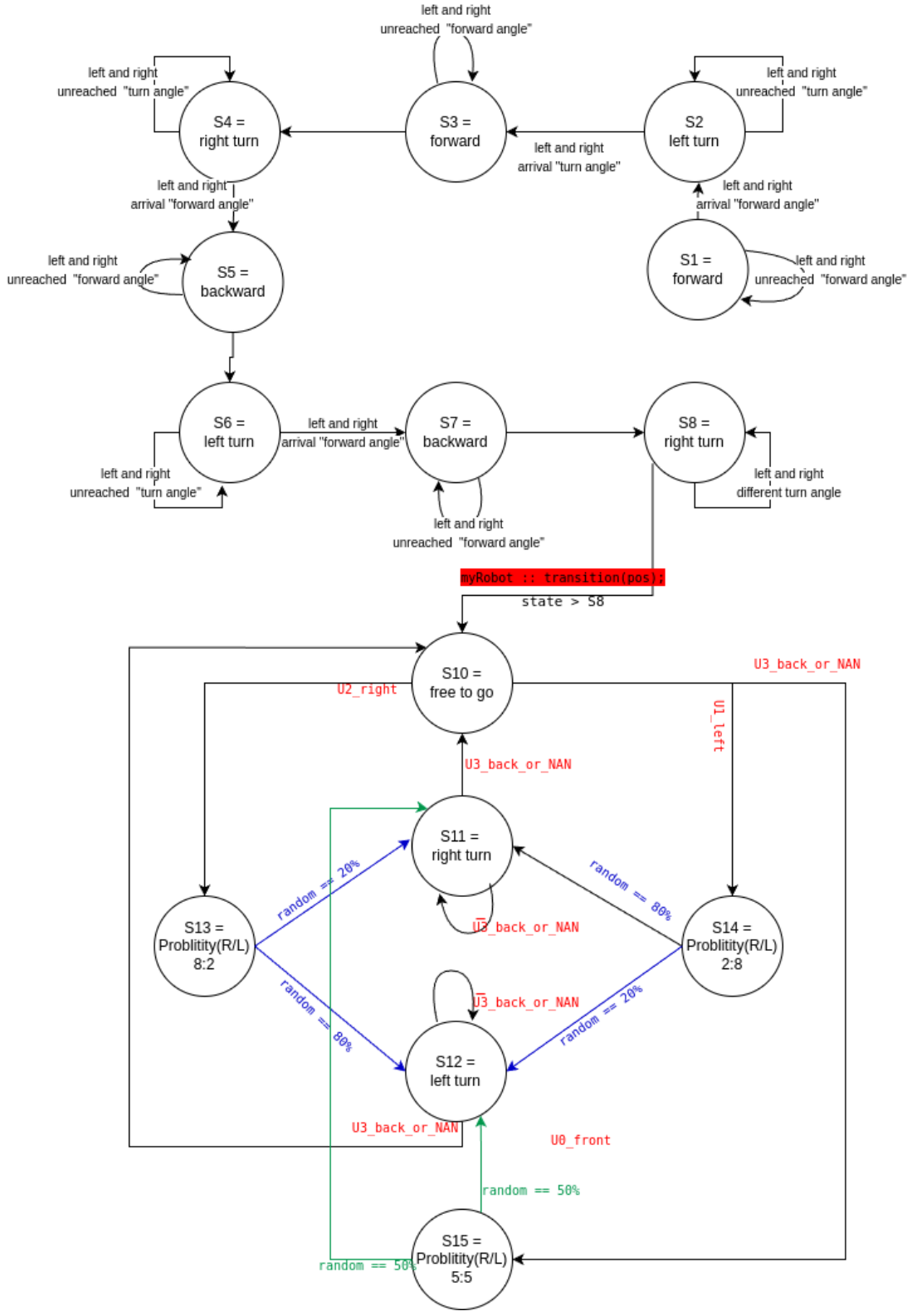
Advanced Programming in Robotic Navigation

Mid-Term report name:Tzu-Ching Chao number:111323083

state transition diagram :

The Square Navigation FSM controls the robot to trace a square path using states S1 to S8, transitioning to the Obstacle Avoidance FSM at S10.



Subclass Method for State Transition

The STATE_OA() method in the OAmyRobot class implements state transitions for the OA FSM, while invoking the transition() method from the parent myRobot class for the Square Navigation FSM.

- **Functionality**:
    - Call myRobot::transition(pos) in Square.hpp updates the Square Navigation state.
    - If state > S8 (i.e., in OA mode), evaluates the current oa_state (S10–S15).
    - Uses range() to determine the obstacle position (U0_front, U1_left, U2_right, U3_back_or_NAN).
    - Transitions to the next oa_state based on the current state and input symbol, executing actions (Forward, Backward, Left_turn, Right_turn).
    - Probabilistic transitions in S13, S14, and S15 use random numbers (rand()) to select turning directions.
    - Updates oa_state and logs state information.
- **Key Logic**:
    - S10 (free to go) moves forward unless an obstacle is detected, and then switches to the left-right turn probability state (S13, S14, S15).
    - S11 and S12 continue until no obstacle is detected (U3_back_or_NAN), i.e., the obstacle is already behind, and the state switches to U0 straight ahead.
    - The states of left-right turn probability (S13, S14, S15) utilize backward movement and random selection to avoid left-right turn collision with obstacles. The reason why I don't go backward in straight ahead, right turn and left turn (S10, S11, S12) is because I will go backward every time, which is very bad for walking efficiency, so I will put the backward part in the left/right turn probability, so that when I detect an object, I will go backward to choose the left/right turn probability.

Subclass Method for Sensor Feedback and Obstacle Detection

The range() method, supported by calculateCenterOfMass(), handles sensor feedback and obstacle detection.

calculateCenterOfMass():

1. Reads values from eight distance sensors (ps0–ps7).
2. Computes a weighted center of mass using complex numbers, where sensor values above THRESHOLD contribute to a polar vector.
3. Sensor angles (theta): [0.30, 0.80, 1.57, 2.64, -2.64, -1.57, -0.80, -0.30] radians.
4. Outputs thetac (angle of obstacle center) or NAN if no obstacle is detected (all sensors below THRESHOLD).

range():

1. Call calculateCenterOfMass() to get thetac.
2. Maps thetac to the four ranges:
    a. U0_front: Obstacle ahead , [-0.1, 0.1] .
    b. U1_left: Obstacle on left , [-1.485, -0.1) .
    c. U2_right: Obstacle on right , (0.1, 1.485].
    d. U3_back_or_NAN: Outside above ranges or NAN (back or no obstacle).
3. Returns the corresponding input symbol for the OA FSM.

Other Important Features

1. Combine Square FSM and Obstacle Avoidance FSM, the main function is to put S10 into the last state of Square FSM to trigger the next state, and then do the if condition judgment in Obstacle Avoidance FSM, if the state is bigger than S8, then it will go into Obstacle Avoidance FSM to do the state change of obstacle in different position.
2. States S13, S14, and S15 use random number generation (rand()) to determine the chance of making a right turn, so that the E-puck will not be able to go out of the corner by repeatedly spinning in the same direction all the time.