Function 1: Get Current/Read New Position

1. IMU outputs x-position & y-position of the robot in the world frame.
    - Where is the x & y IMU outputs with respect to the robots frame?
        i. Magni is in the center of the front two wheels
        ii. Rosbot is in the center of the robot
    - X-position will tell us how far along the planned trajectory the robot is
    - Y-position will tell us how far above or below the planned trajectory we are

Function 2: Calculate Desired Position/"Combined Angle"

1. Find closest point on trajectory to current position
2. Find the cross track error
3. Calculate the "Combined angle" from the stanley method
    a. Theta_error = theta_car - theta_path
    b. Combined angle(gamma) = theta_error + atan(k*CrossTrackError/ForwardVelocity)
    c. Cross track error is defined as the perpendicular distance from the path to the robot
    d. K is the "strength" of the correction. ie. larger k = faster correction

Function 3: Plug into RL to get action (prescribed wheel-power ratio)

Inputs: Combined angle/y-error variable, previous reward

Outputs: Wheel-power ratio, reward value

Receive calculated "combined angle/y-error" variable

Assign reward based upon calculated "combined angle/y-error" variable

Compare reward to previous reward to tell robot it is or isn't making progress (?)

Adjust wheel-power ratio accordingly

Send this ratio to the robot

Function 4: Apply Action

Inputs: A key that signifies which action to take (got from the Q-table)

Outputs: ROS Command/message - Power ratio (ubiquity_motor) to the wheels or linear velocity(/cmd_vel)

1. After receiving the key match it with value pair for the corresponding PR (1- “1:0”,2-” 3:1”,3-” 2:1”,4-” 1:1”,5-” 1:2”,6-” 1:3”,7-” 0:1”), Can just be the index of the array starting at 0.
2. Specify the power ratio using the ubiquity_motor package that is built in ROS and supply the corresponding amount of power to wheels.
   Alternatively, we could use /cmd_vel and give different linear velocities in the x-direction to each wheel making it turn/ go straight.

Function 5: training

Input : Combined angle value
Output: Q-weights

Initialise reward matrix such that +x values are rewarded and -x values are penalised.

Still need to decide the dimensions of the Q_matrix and Reward_matrix

```
while( iteration < 2000 )
    State = get_current_position()
    Action = random_action from possible states

    IF action is in (-)ve x axis:
            reselect the random action until +ve xaxis



    Select maximum Q value on application of Action for time dT
    Error = Input_combined_angle_after_action - Desired_combined_angle
    If (Error < threshold_waypoint)
    update the Q matrix according to the bellman equation with discount factor

    If error < threshold_goal:
            break
```