```python
import numpy as np
import skfuzzy as fuzz
import skfuzzy.control as ctrl

class JointFuzzyController:
    def __init__(self, joint_name):
        self.e = ctrl.Antecedent(np.arange(-1.5, 1.5, 0.1), f'{joint_name}_e')
        self.ec = ctrl.Antecedent(np.arange(-1.5, 1.5, 0.1), f'{joint_name}_ec')
        self.myvelocity = ctrl.Antecedent(np.arange(0, 2, 0.1), f'{joint_name}_myvelocity')
        self.velocity = ctrl.Consequent(np.arange(0, 2, 0.1), f'{joint_name}_velocity')
        self.velocityob = ctrl.Consequent(np.arange(0, 2, 0.1), f'{joint_name}_velocityob')
        self.distance_obstacle = ctrl.Antecedent(np.arange(0, 1.7, 0.1), f'{joint_name}_distance_obstacle')
        self.distance_target = ctrl.Antecedent(np.arange(0, 1.7, 0.1), f'{joint_name}_distance_target')
        self._init_membership_functions()
        self._init_rules_velocity()
        self._init_rules_ob_velocity()

        self.ctrl_vrules = ctrl.ControlSystem(self.vrules)
        self.sim_vrules = ctrl.ControlSystemSimulation(self.ctrl_vrules)
        self.ctrl_obvrules = ctrl.ControlSystem(self.obvrules)
        self.sim_obvrules = ctrl.ControlSystemSimulation(self.ctrl_obvrules)

    def _init_membership_functions(self):
        self.e.automf(names=['NB', 'NS', 'ZO', 'PS', 'PB'])
        self.ec.automf(names=['NB', 'NS', 'ZO', 'PS', 'PB'])

        self.velocity['Slow'] = fuzz.trimf(self.velocity.universe, [0, 0, 1])
        self.velocity['Medium'] = fuzz.trimf(self.velocity.universe, [0.5, 1, 1.5])
        self.velocity['Fast'] = fuzz.trimf(self.velocity.universe, [1, 2, 2])

        self.distance_obstacle['VN'] = fuzz.trimf(self.distance_obstacle.universe, [0, 0, 0.51])
        self.distance_obstacle['N'] = fuzz.trimf(self.distance_obstacle.universe, [0.4, 0.7, 1.0])
        self.distance_obstacle['F'] = fuzz.trapmf(self.distance_obstacle.universe, [0.8, 1.0, 1.4, 1.7])
        self.distance_obstacle['VF'] = fuzz.trapmf(self.distance_obstacle.universe, [1.5, 1.6, 1.7, 1.7])

        self.distance_target['VN'] = fuzz.trimf(self.distance_target.universe, [0, 0, 0.51])
        self.distance_target['N'] = fuzz.trimf(self.distance_target.universe, [0.4, 0.7, 1.0])
        self.distance_target['F'] = fuzz.trapmf(self.distance_target.universe, [0.8, 1.0, 1.4, 1.7])
        self.distance_target['VF'] = fuzz.trapmf(self.distance_target.universe, [1.5, 1.6, 1.7, 1.7])

        self.velocityob.automf(names=['VS', 'MS', 'S', 'M', 'F', 'MF', 'VF'])
        self.myvelocity.automf(names=['VS', 'MS', 'S', 'M', 'F', 'MF', 'VF'])

    def _init_rules_velocity(self):
        self.vrules = [
            ctrl.Rule(self.e['NB'] & self.ec['NB'], self.velocity['Slow']),
            ctrl.Rule(self.e['NB'] & self.ec['ZO'], self.velocity['Medium']),
            ctrl.Rule(self.e['ZO'] & self.ec['ZO'], self.velocity['Medium']),
            ctrl.Rule(self.e['PB'] & self.ec['PB'], self.velocity['Fast']),
        ]
    def _init_rules_ob_velocity(self):
        self.obvrules = [
            ctrl.Rule(self.distance_target['VN'] & self.distance_obstacle['VN'], self.velocityob['VS']),
            ctrl.Rule(self.distance_target['F'] & self.distance_obstacle['N'], self.velocityob['M']),
            ctrl.Rule(self.distance_target['F'] & self.distance_obstacle['F'], self.velocityob['F']),
        ]
    def determine_best_distance_category(self, distance_value):
        membership_values = {
            'VF': fuzz.interp_membership(self.distance_obstacle.universe, self.distance_obstacle['VF'].mf, distance_value),
            'F': fuzz.interp_membership(self.distance_obstacle.universe, self.distance_obstacle['F'].mf, distance_value),
            'N': fuzz.interp_membership(self.distance_obstacle.universe, self.distance_obstacle['N'].mf, distance_value),
            'VN': fuzz.interp_membership(self.distance_obstacle.universe, self.distance_obstacle['VN'].mf, distance_value),
        }
        return max(membership_values, key=membership_values.get)

    def compute_velocity(self, error, error_change):
        self.sim_vrules.input[f'{self.e.label}'] = error
        self.sim_vrules.input[f'{self.ec.label}'] = error_change
        self.sim_vrules.compute()
        return self.sim_vrules.output[f'{self.velocity.label}']

    def compute_velocity_ob(self, error, error_change, myvelocity, distance_target_v, distance_obstacle_value):
        best_distance_obstacle = self.determine_best_distance_category(distance_obstacle_value)
        if best_distance_obstacle == 'VN':
            return 0
        self.sim_obvrules.input[f'{self.distance_target.label}'] = distance_target_v
        self.sim_obvrules.input[f'{self.distance_obstacle.label}'] = distance_obstacle_value
        self.sim_obvrules.input[f'{self.myvelocity.label}'] = myvelocity
        self.sim_obvrules.compute()
        return self.sim_obvrules.output[f'{self.velocityob.label}']
```