Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 2

по курсу «Основы искусственного интеллекта»

на тему: «Алгоритмы нечеткой логики»

Студент ИУ7-13М (Группа) _____ (Подпись, дата) Орду М. А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Строганов Ю. В. (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

1 Теоретическая часть

1.1 Постановка задачи

В одномерном пространстве ($X = 1$) рассматриваются два автомобиля: лидер, управляемый пользователем, и автомобиль-автопилот. Автопилот должен следовать за лидером, поддерживая постоянную дистанцию D, не имея информации о скорости лидера. Известно только текущее расстояние между автомобилями. Требуется определить необходимую скорость автопилота v_{auto} на основе нечеткого логического вывода.

Определение ускорения запрещено. Входными переменными являются:

- ошибка по расстоянию $e = D - D$;
- изменение ошибки $\Delta e = \frac{de}{dt}$.

Выходная переменная — скорость автопилота v_{auto}.

1.2 Этапы нечеткого логического вывода

Нечеткий логический вывод состоит из следующих этапов:

1. **Фаззификация** — преобразование четких входных значений e и Δe в степени принадлежности нечетким подмножествам.
2. **Применение базы правил** — вычисление степени активации каждого правила на основе входных значений.
3. **Импликация** — формирование выходных нечетких множеств в соответствии с вычисленной степенью активации α.
4. **Агрегация** — объединение всех выходных множеств.
5. **Дефаззификация** — преобразование агрегированного нечеткого множества в четкое значение v_{auto}.

1.3 Функции принадлежности

Для входных и выходных переменных были выбраны следующие функции принадлежности (рис. **??**):

- Ошибка расстояния `error`: `too_close`, `normal`, `far`;
- Изменение ошибки `delta`: `approaching`, `steady`, `moving_away`;
- Скорость `speed`: `slow`, `medium`, `fast`.

Для задания функций использовались трапециевидные и треугольные формы:

$$\texttt{trapmf}(x; a, b, c, d) = \begin{cases} 0, & x < a, \\ \frac{x-a}{b-a}, & a \le x < b, \\ 1, & b \le x \le c, \\ \frac{d-x}{d-c}, & c < x < d, \\ 0, & x \ge d, \end{cases}$$

$$\texttt{trimf}(x; a, b, c) = \begin{cases} 0, & x < a, \\ \frac{x-a}{b-a}, & a \le x < b, \\ \frac{c-x}{c-b}, & b \le x < c, \\ 0, & x \ge c. \end{cases}$$

1.4 База правил

Правила нечеткого вывода описывают зависимость между ошибками и требуемой скоростью:

1. ЕСЛИ error = too_close И delta = approaching ТО speed = slow;
2. ЕСЛИ error = too_close И delta = steady ТО speed = slow;
3. ЕСЛИ error = too_close И delta = moving_away ТО speed = medium;
4. ЕСЛИ error = normal И delta = approaching ТО speed = slow;
5. ЕСЛИ error = normal И delta = steady ТО speed = medium;
6. ЕСЛИ error = normal И delta = moving_away ТО speed = fast;
7. ЕСЛИ error = far И delta = approaching ТО speed = medium;

8. ЕСЛИ error = far И delta = steady ТО speed = fast;

9. ЕСЛИ error = far И delta = moving_away ТО speed = fast.

1.5 Алгоритм импликации Ларсена

Для варианта лабораторной работы используется **импликация Ларсена**:

$$\mu_{B'}(y) = \alpha \times \mu_B(y),$$

где α — степень истинности предпосылки.

В данной работе α определяется как произведение степеней истинности входных условий:

$$\alpha = \mu_{A_1}(x_1) \cdot \mu_{A_2}(x_2),$$

что обеспечивает более плавное изменение выходного множества по сравнению с методом Мамдани, где используется минимум.

1.6 Алгоритм дефаззификации

Для получения четкого значения скорости используется метод **центра тяжести** (centroid):

$$v = \frac{\int y \cdot \mu(y)\, dy}{\int \mu(y)\, dy}.$$

В качестве альтернативного метода можно применять **метод среднего максимума (mean of maxima)**:

$$v = \frac{y_{\min} + y_{\max}}{2}, \quad y_{\min,\max} \in \{y | \mu(y) = \max(\mu)\}.$$

2 Практическая часть

2.1 Используемые инструменты

Для реализации нечеткой системы использована библиотека `scikit-fuzzy` и язык Python 3. Основные зависимости:

- `numpy` — численные вычисления;
- `matplotlib` — визуализация;
- `scikit-fuzzy` — функции принадлежности и дефаззификация.

2.2 Результаты

На рисунке **??** показана динамика движения автомобилей. Автопилот плавно регулирует скорость, удерживая требуемую дистанцию от лидера даже при изменении его скорости.

Среднеквадратичная ошибка поддержания дистанции составила:

$$MSE = 0.153.$$

Время моделирования при шаге интегрирования $dt = 0.1$ и $T = 100\,c$ составило менее 1 секунды.

2.3 Особенности реализации

- Импликация Ларсена реализована вручную как масштабирование функции принадлежности по вертикали.
- Используется произведение степеней истинности для расчета силы активации ($\alpha = \mu_1 \times \mu_2$).
- Дефаззификация выполняется методом центра тяжести (`fuzz.defuzz(..., 'centroid')`).
- Система легко расширяется на $X = 3$ при добавлении новых входных переменных (например, бокового смещения).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Исходный код программы

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# === 1. Нечёткие переменные ===
error = ctrl.Antecedent(np.arange(-20, 20.1, 0.5), 'error')  # м
delta = ctrl.Antecedent(np.arange(-10, 10.1, 0.5), 'delta')  # м/с
accel = ctrl.Consequent(np.arange(-3, 3.1, 0.1), 'accel', defuzzify_method='centroid')

# === 2. Функции принадлежности ===
error['too_close'] = fuzz.trapmf(error.universe, [-20, -20, -10, -3])
error['normal']    = fuzz.trimf(error.universe, [-4, 0, 4])
error['far']       = fuzz.trapmf(error.universe, [3, 10, 20, 20])

delta['approaching']   = fuzz.trapmf(delta.universe, [-10, -10, -2, 0])
delta['steady']        = fuzz.trimf(delta.universe, [-1, 0, 1])
delta['moving_away']   = fuzz.trapmf(delta.universe, [0, 2, 10, 10])

accel['brake']       = fuzz.trapmf(accel.universe, [-3, -3, -1.5, -0.5])
accel['hold']        = fuzz.trimf(accel.universe, [-0.5, 0, 0.5])
accel['accelerate']  = fuzz.trapmf(accel.universe, [0.3, 1.5, 3, 3])

# === 3. Правила ===
rules = [
    ctrl.Rule(error['too_close'] & delta['approaching'], accel['brake']),
```

```
27      ctrl.Rule(error['too_close'] & delta['steady'],
            accel['brake']),
28      ctrl.Rule(error['too_close'] & delta['moving_away'],
            accel['hold']),
29      ctrl.Rule(error['normal'] & delta['approaching'],
            accel['brake']),
30      ctrl.Rule(error['normal'] & delta['steady'],
            accel['hold']),
31      ctrl.Rule(error['normal'] & delta['moving_away'],
            accel['accelerate']),
32      ctrl.Rule(error['far'] & delta['approaching'],
            accel['hold']),
33      ctrl.Rule(error['far'] & delta['steady'],
            accel['accelerate']),
34      ctrl.Rule(error['far'] & delta['moving_away'],
            accel['accelerate']),
35  ]
36
37  accel_ctrl = ctrl.ControlSystem(rules)
38
39  for rule in accel_ctrl.rules:
40      rule.activation = np.multiply
41
42  accel_sim = ctrl.ControlSystemSimulation(accel_ctrl)
43
44  # === 4. Параметры симуляции ===
45  dt = 0.1
46  T = 60.0
47  steps = int(T / dt)
48
49  d_ref = 25.0     # желаемое расстояние между авто (м)
50  v_auto = 15.0     # начальная скорость авто (м/с)
51  distance = 15.0 # начальная дистанция (м)
52
53  integral_error = 0.0
54  k_i = 100
55  i_term_min = -1.5
56  i_term_max = 1.5
57
58  def v_lead_at_time(t):
59      if t < 20.0:
```

```
        return 20.0
    elif t < 40.0:
        return 15.0
    else:
        return 22.0

# История
time_hist, dist_hist, v_auto_hist, v_lead_hist = [], [], [], []
accel_fuzzy_hist, accel_total_hist, error_hist, delta_hist,
   integral_hist = [], [], [], [], []

for i in range(steps):
    t = i * dt
    v_lead = v_lead_at_time(t)

    error_val = distance - d_ref
    delta_val = v_lead - v_auto

    accel_sim.input['error'] = error_val
    accel_sim.input['delta'] = delta_val
    accel_sim.compute()
    a_fuzzy = accel_sim.output['accel']

    integral_error += error_val * dt
    i_term = k_i * integral_error

    if i_term > i_term_max:
        i_term = i_term_max
        integral_error = i_term_max / k_i
    if i_term < i_term_min:
        i_term = i_term_min
        integral_error = i_term_min / k_i

    a_total = a_fuzzy + i_term
    a_total = max(-3.0, min(3.0, a_total))  # ограничение
       физическое

    v_auto += a_total * dt
    distance += (v_lead - v_auto) * dt

    time_hist.append(t)
```

```
99      dist_hist.append(distance)
100     v_auto_hist.append(v_auto)
101     v_lead_hist.append(v_lead)
102     accel_fuzzy_hist.append(a_fuzzy)
103     accel_total_hist.append(a_total)
104     error_hist.append(error_val)
105     delta_hist.append(delta_val)
106     integral_hist.append(i_term)
107
108 steady_start = int(45.0 / dt)
109 steady_errors = np.array(error_hist[steady_start:])
110 print("Средняя ошибка: {:.3f}
        м".format(np.mean(steady_errors)))
111 print("Макс абс ошибка: {:.3f}
        м".format(np.max(np.abs(steady_errors))))
112
113 # === 7. Графики ===
114
115 # --- Окно 1: дистанция ---
116 plt.figure(figsize=(8, 5))
117 plt.plot(time_hist, dist_hist)
118 plt.axhline(d_ref, linestyle='--')
119 plt.ylabel('Дистанция (м)')
120 plt.xlabel('Время (с)')
121 plt.title('Изменение дистанции во времени')
122 plt.grid(True)
123 plt.tight_layout()
124 plt.show()
125
126 # --- Окно 2: скорости ---
127 plt.figure(figsize=(8, 5))
128 plt.plot(time_hist, v_auto_hist, label='Автопилот')
129 plt.plot(time_hist, v_lead_hist, label='Ведущее авто',
        linestyle='--')
130 plt.ylabel('Скорость (м/с)')
131 plt.xlabel('Время (с)')
132 plt.title('Скорости автомобилей во времени')
133 plt.legend()
134 plt.grid(True)
135 plt.tight_layout()
136 plt.show()
```