Метод

настройки регуляторов с помощью мысленного эксперимента

Ярмолинский Арсений,
педагог дополнительного образования

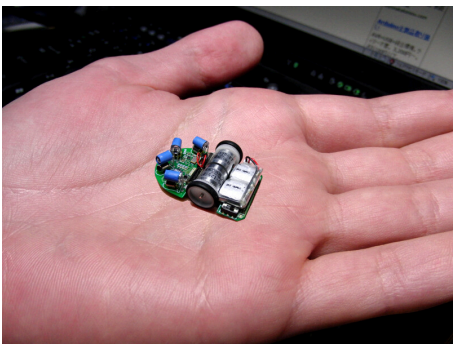
Управление положением

Мысленный эксперимент: управляем роботом с пульта

Задача: повернуться на угол

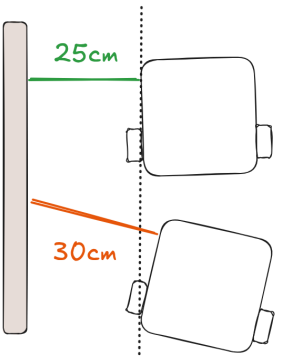
Медленный робот

Быстрый робот



Настраиваем П-регулятор

1. Представляем ситуацию, требующую коррекции
2. Оцениваем численное значение ошибки

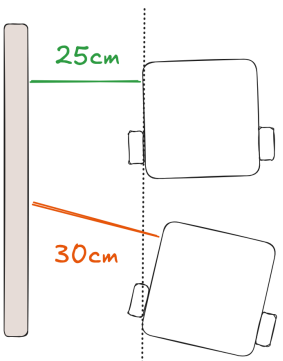


err = 25 - 30 = -5

u = ?

Настраиваем П-регулятор (продолжение)

3. Представляем какое управляющее воздействие мы бы дали если бы управляли вручную
4. Считаем коэффициент пропорциональности



err = 25 - 30 = -5

u = 80 = err * Kp

Kp = u / err = 80/-5 = -16

При необходимости оцениваем максимальное управляющее воздействие $u_{\max}$, которые мы хотим позволить роботу отрабатывать

Итоговый код регулятора


```cpp
float err = ...;
float Kp = 80.0 / -5.0;
float u = Kp * err;
u = constrain(u, -umax, umax);
drive(v - u, v + u);
```

(1) Весь код регулятора

Настройка - всего два параметра:

1. K_p - Коэффициент пропорциональности (быстродействие и точность)
2. $u_{\max}$ - Максимальное управляющее воздействие (скорость)

Важно

Всегда используйте **float** для математики!

Комбинирование регуляторов

Идея

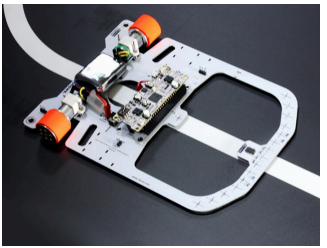
Регуляторы по разным величинам можно комбинировать и усреднять для сложных поведений

Код - движение в лабиринте Micromouse: условное выравнивание по левой и правой стенке, безусловное выравнивание по углу

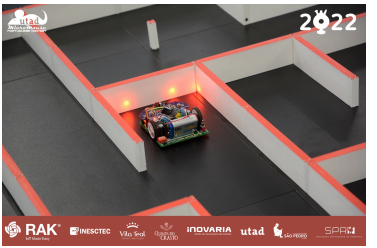
```cpp
float wf_straight_tick(SensorData data)
{
    float err_left = WF_LEFT_REFERENCE - data.dist_left;
    float err_right = WF_RIGHT_REFERENCE - data.dist_right;
    float err_angle = 0 - data.odom_theta;
    float theta_i0_left = err_left * wf_kp_left;
    float theta_i0_right = err_right * wf_kp_right;
    float theta_i0_angle = err_angle * wf_kp_angle;

    float theta_i0 = 0;
    size_t counter = 0;

    if (data.is_wall_left)
    {
        theta_i0 += theta_i0_left;
        counter++;
    }
    if (data.is_wall_right)
    {
        theta_i0 += theta_i0_right;
        counter++;
    }

    theta_i0 += theta_i0_angle;
    counter++;

    if (counter != 0)
    {
        theta_i0 /= counter;
    }

    return theta_i0;
}
```

Метод настройки регуляторов с помощью мысленного

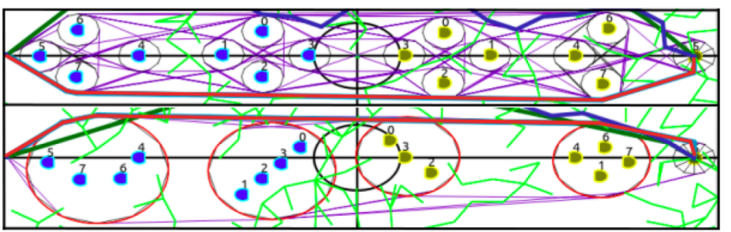
Задачи управления положением



Следование по линии



Движение вдоль стенки



Навигация в пространстве

Только П-регулятор?

? Как так?

Робот медленный и неточный

⚠ Причина

Высокая инерционность и нечуствительность на низких скоростях

Решение

Сделать полный ПИД

+ Легко объясняется
+ Есть готовые библиотеки

- Сложная настройка
- Невозможность достижения быстродействия и точности других методов

Поставить более моментные моторы

+ Простая настройка
+ Высокое быстродействие и точность

- Требуется изменение механики
- Снижение максимальной скорости робота

Добавить регуляторы скорости на моторы

+ Конкретный алгоритм настройки
+ Высокое быстродействие и точность

- Много мест, где можно ошибится при реализации

Настройка регулятора скорости

Интермедия

Вычисление скорости вращения мотора

Два способа:

- Период между тиками энкодера
- Путь за итерацию делить на время итерации

Не подходит

Сильно теряет точность и быстродействие при низких и нулевых скоростях

Подходит

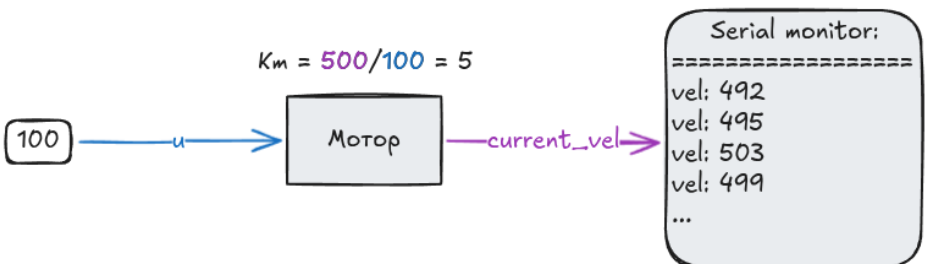
Скорость должна быть со знаком

Шумно, но можно фильтровать

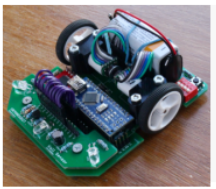
```cpp
vel = (angle - angle_old) / delta_time;
```

Измерение коэффициента усиления мотора K_m

1. Даем фиксированное управляющее воздействие
2. Измеряем получившуюся скорость
3. Считаем коэффициент усиления

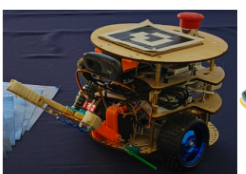


Оценка быстроты системы (*постоянной времени*) T_m



$T = 0.05 - 0.15s$

$T = 3 - 10s$



$T = 0.1 - 0.3s$

$T = 60 - 180s$

Используем ПИ регулятор

? Во сколько раз хотим ускорить систему?

Это будет наш параметр Boost: B

Коэффициенты считаются так:

$$K_p = \frac{B}{K_m}$$

$$K_i = \frac{K_p}{T_m}$$

Пишем регулятор


```cpp
float kMl = 4.5 / 150, kMr = 4.4 / 150;
float Boost = 4;
float tm = 250.0 / 1000;
float kp_speedR = Boost / kMr, ki_speedR = Boost / (tm * kMr);
float kp_speedL = Boost / kMl, ki_speedL = Boost / (tm * kMl);

void motorRPM(float rpmL, float rpmR, uint32_t iter_time_ms = 5 /*[ms]*/)
{
  static uint32_t timer = 0;
  while (millis() - timer < iter_time_ms)
    ;
  timer = millis();

  vel_est_tick();

  float errL = rpmL - getvel_left();
  float errR = rpmR - getvel_right();

  static float uIL = 0;
  static float uIR = 0;

  uIL += errL * ki_speedL * iter_time_ms / 1000.0;
  uIR += errR * ki_speedR * iter_time_ms / 1000.0;
  uIL = constrain(uIL, -255, 255);
  uIR = constrain(uIR, -255, 255);

  float uL = errL * kp_speedL + uIL;
  float uR = errR * kp_speedR + uIR;

  drive(uL, uR);
}
```

Используем регулятор


```cpp
void loop()
{
  motorRPM(vel_left, vel_right);
}
```

Спасибо за внимание

@ARSENIS