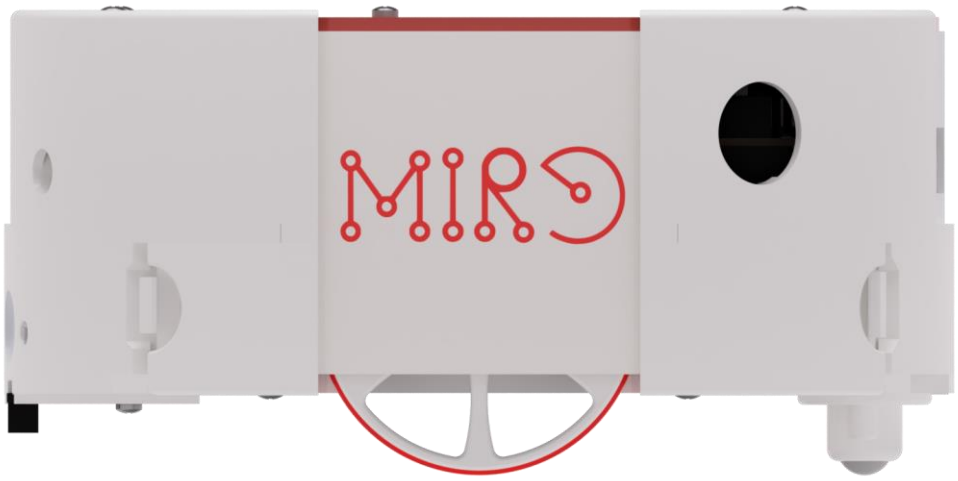
MIRO EDUCATION BOOK

LEVEL 2



УРОК 11. ОБЪЕЗД ПРЕПЯТСТВИЙ

Описание урока

Реализуем и проанализируем простой алгоритм по объезду препятствий с использованием ультразвукового дальномера.

Информационный блок

Давайте начнем с определения. Под объездом будем понимать преодоление естественного физического ограничения (предмета) для прямолинейного движения робота, связанное с выездом из текущей траектории.

Давайте начнем с разработки простого алгоритма поведения робота для объезда препятствия на пути.

Прежде чем проектировать алгоритм основного цикла, необходимо четко задать условия и ограничения.

1. Так как в роботе MIRO в модуле SENS1 установлен один ультразвуковой дальномер, то и алгоритм для объезда необходимо разработать исходя из этого ограничения.
2. Определимся, что в нашем простом алгоритме перед роботом будет стоять задача объезда препятствия известной протяженности (ширины) – не более 50 см.
3. Для визуализации процесса объезда, добавим в наш алгоритм требование звуковой или световой сигнализации при движении робота с увеличением частоты звуковых импульсов по мере приближения к препятствию.
4. Определим границы расстояния, на котором робот должен начать маневр по объезду препятствия: 30 см.

Спланируем наш алгоритм основного цикла.

1. Двигаться вперед.
2. Получить расстояние до объекта (или объектов) на пути движения, используя ультразвуковой дальномер. Если есть препятствие на расстоянии не 30 см или менее – перейти к объезду препятствия.
3. Выполнить поворот на 90 градусов влево.
4. После выполнения поворота, робот должен проверить, нет ли прямо по курсу нового препятствия на расстоянии 70 см или менее (50 см – заданная нами максимальная протяженность препятствия, 20 см – запас для маневра), не позволяющих выполнить объезд.
5. Если препятствия обнаружилось, робот должен развернуться на 180 градусов, чтобы попытаться выполнить объезд с другой стороны.
6. После выполнения разворота, робот должен снова проверить, нет ли прямо по курсу нового препятствия, не позволяющих выполнить объезд.
7. Если и в этом случае есть препятствие, значит робот находится в тупике и единственный вариант – развернуться на 180 градусов относительно начального положения и выезжать из тупика. Последний случай мы предлагаем

решить самостоятельно – он является развитием рассматриваемого нами здесь и далее алгоритма.

8. Если в любом из случаев в пунктах 4 или 6 путь для робота свободен, то робот должен проехать вперед отрезок пути 50 см (объезжая препятствие). И выполнить поворот на 90 градусов в обратную сторону, относительно той, в которую робот повернулся для объезда (если совершаем объезд препятствия «справа», то робот должен повернуться налево и наоборот).

9. Следовать вперед. Объезд выполнен.

Это примерный алгоритм объезда препятствий, который вы можете улучшить самостоятельно.

Давайте рассмотрим код, реализующий данный алгоритм.

Код программы

```
#define TONE_PIN 11   //пин пъезоизлучателя
#define US_TRIG_PIN 13 //пин TRIG УЗ-дальномера
#define US_ECHO_PIN 12 //пин ECHO УЗ-дальномера

#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

#define LIN_SPEED 128

//Коды состояний автомата объезда
#define STATE_FW 0
#define STATE_L 1
#define STATE_R 2

//Переменная состояния автомата объезда
int state = STATE_FW;

//расстояние до препятствия, ближе которого совершать объезд
#define DISTANCE_TH 20

//Функция движения вперед
void forward()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}

// Функция движения назад
void back()
{
  digitalWrite(DIR_L_PIN, HIGH);
  digitalWrite(DIR_R_PIN, HIGH);
  analogWrite(PWM_L_PIN, 255 - LIN_SPEED);
  analogWrite(PWM_R_PIN, 255 - LIN_SPEED);
}

// Функция остановки обоих двигателей
void stop()
{
```

```
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, 0);
}

// Функция проезда вперед на 50см
void forward1000()
{
  forward();
  delay(1000);
  stop();
}

// Функция поворота на 90 градусы налево
void left90()
{
  digitalWrite(DIR_L_PIN, HIGH);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 255-LIN_SPEED);
  analogWrite(PWM_R_PIN, LIN_SPEED);

  delay(600);

  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, 0);
}

// Функция поворота на 90 градусов направо
void right90()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, HIGH);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 255 - LIN_SPEED);

  delay(600);

  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, 0);
}

// Функция разворота на 180 градусов
void turn180()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, HIGH);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 255 - LIN_SPEED);

  delay(1200);

  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, 0);
}

//Функция объезда препятствия
```

```
void detour()
{
  float dist = get_distance();

  switch (state)
  {
    case STATE_FW:
      if (dist < DISTANCE_TH)
      {
        stop();
        delay(50);
        left90();
        state = STATE_L;
      }
      else
      {
        forward();
      }
      break;

    case STATE_L:
      if (dist < DISTANCE_TH)
      {
        turn180();
        state = STATE_R;
      }
      else
      {
        forward1000();
        delay(50);
        right90();
        state = STATE_FW;
      }
      break;

    case STATE_R:
      if (dist < DISTANCE_TH)
      {
        left90();
        state = STATE_FW;
      }
      else
      {
        forward1000();
        delay(50);
        left90();
        state = STATE_FW;
      }
      break;

    default:
      break;
  }
}

// Функция считывания расстояния с УЗ-дальномера
float get_distance()
{
  digitalWrite(US_TRIG_PIN, LOW);
  delayMicroseconds(4);

  digitalWrite(US_TRIG_PIN, HIGH);
  delayMicroseconds(20);
```

```
  digitalWrite(US_TRIG_PIN, LOW);
  long i_distance = pulseIn(US_ECHO_PIN, HIGH, 23280);

  //long i_distance = pulseIn(US_ECHO_PIN, HIGH);
  //пересчет значений расстояния в сантимметры
  float distance = ((float)i_distance) / 58.2;

  if (distance >= 400)
  {
    //Serial.println("US Distance infinite");
    distance = 401;
  }
  else if (distance <= 2)
  {
    //Serial.println("US Distance 0");
  }
  else
  {
    //Для отладки
    //Serial.print("US Distance: ");
    //Serial.println(distance);
  }
  return distance;
}

//Функция сигнализации приближения к препятствию
void us_beep()
{
  //объявляем переменные для хранения моментов времени
  unsigned long current_time = millis();
  static unsigned long last_time = 0;

  // получаем расстояние до объекта
  float dist = get_distance();

  if (dist > 0)
  {
    if ((current_time - last_time) >= 25 * dist)
    {
      last_time = current_time;
      tone(TONE_PIN, 2500, 30);
    }
  }
}

void setup()
{
  //Настраиваем линии управления двигателями
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);

  //настраиваем линии управления УЗ-дальномером
  pinMode(TONE_PIN, OUTPUT);
  pinMode(US_TRIG_PIN, OUTPUT);
  pinMode(US_ECHO_PIN, INPUT);

  Serial.begin(9600);
}

void loop()
```

```
{
  us_beep();
  detour();
  delay(50);
}
```

Пояснения к коду

1. Обратим внимание на реализации функций поворота. В последующих уроках мы разберем тему «Одометрия» и сможем более четко выполнять поворот робота на необходимы угол независимо от скорости и характера движения. Однако, пока поворот в любую сторону на фиксированный угол выполняется путем запуска двигателей в противоположных направлениях на экспериментально отмеренный промежуток времени. В функциях `left90()` и `right90()` этот промежуток задается вызовом `delay(600)` – 600 миллисекунд.

2. В отличие реализации функции `get_distance()` из урока «Ультразвуковой дальномер. Подключение и работа», в текущей реализации этой функции есть небольшое отличие – в ней есть проверка попадание в диапазон возможных значений датчика. Согласно техническому паспорту, у УЗ-дальномер HC-SR04 этот диапазон составляет от 2 до 400 сантиметров. Зная эти особенности накладываем условия на интерпретацию данных с датчика.

```
if(distance >= 400 || distance <= 2)
  {
    //Serial.println("US>Distance 0 or infinite");
    distance = -1;
  }
else
  {
    //Serial.print("US>Distance: ");
    //Serial.println(distance);
  }
```

Эту функцию можно смело использовать в следующих проектах.

3. Функция `detour()` реализует так называемый *автомат состояний* (он же *конечный автомат*), реализующий алгоритм объезда препятствий. В языке программирования C/C++ подобные автоматы как правило реализуются через языковую конструкцию `switch-case`. Данная функция описывает поведение робота во всех состояниях (логических), которые могут случиться с роботом по мере выполнения задачи по объезду препятствия. Набор состояний и их логический смысл определяет разработчик. В нашем случае робот может находится в 3 возможных состояниях:

`STATE_FW` – начальное состояние, робот просто движется вперед;

`STATE_L` – робота выполняет объезд препятствия слева;

`STATE_R` – робот выполняет объезд препятствия справа.

Функция `detour()` описывает что должен делать автомат (в нашем случае – робот) в каждом из этих состояний. Также, эта функция описывает условия перехода от одного состояния автомат к другому. Согласно алгоритму, робот изначально находится в состоянии `STATE_FW` – движется вперед. Затем, при

наступлении условия – на пути появилось препятствие – роботе переходит в состояние `STATE_L` (объезд препятствия слева). Если ничего не мешает роботу совершить этот маневр, то он его совершает (находясь все в том же состоянии `STATE_L`), и затем возвращается в состояние `STATE_FW`. Если же, после поворота налево, находясь в состоянии `STATE_L`, робот обнаруживает препятствие, не позволяющее совершить объезд, то робот разворачивается на 180 градусов и переходит в состояние `STATE_R` – объезд препятствия справа. Логика состояния объезда препятствия справа очень похожа: можно совершить объезд – совершаем и возвращаемся в состояние `STATE_FW`.

4. Функция `us_beep()`, как мы видим, ничего не возвращает, а лишь служит для звуковой сигнализации при приближении к препятствию и работает абсолютно независимо от автомата `detour()`. В функции `us_beep()` применяется очень распространенный прием для выполнения каких-либо действий роботом в определенные периоды времени без блокирования основного цикла программы с помощью функций `delay()`. Каждый цикл программы, переменной `current_time` присваивается текущее время в миллисекундах с момента запуска микроконтроллера робота:

```
unsigned long current_time = millis();
static unsigned long last_time = 0;
```

И далее, каждый такт цикла `loop()`, производится проверка длительности времени, прошедшего с момента последнего выполнения события. В нашем случае событием является подача звукового сигнала. Момент времени, когда в последний раз издавался звуковой сигнал хранится в переменной `last_time`. Переменная `last_time` объявлена со спецификатором `static`, что исключает ее повторную инициализацию нулевым значением (она в течение всего времени работы программы принимает и хранит последнее присвоенное ей значение момента времени, когда подавался звуковой сигнал). Если разница `(current_time - last_time)` становится больше `(25*dist)`, подается звуковой сигнал. Коэффициент `25` при `dist` в данном случае подбирается эмпирически, чтобы получился требуемый нам звуковой эффект увеличения частоты импульсов по мере приближения к препятствию.

Дополнительное задание:

1. Реализуйте алгоритм объезда препятствия неизвестной протяженности.

УРОК 12. ДВИЖЕНИЕ ПО ЛИНИИ НА ОСНОВЕ ЦИФРОВЫХ ДАТЧИКОВ

Описание урока

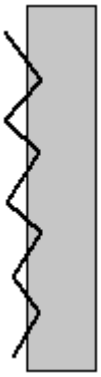
В ходе этого урока изучим алгоритмы движения робота MIRO по линии. Напишем программу для перемещения MIRO по линии по одному, двум и трем датчикам, в ходящим в состав модуля SENS1.

Информационный блок

Движение робота по линии – распространенное решение в робототехнике для задания точной траектории движения робота. Ярким примером использования такого решения являются различные логистические складские роботы (всем известные автоматизированные склады компании Amazon и Национальной Почты КНР).

Для движения по линии существует много алгоритмов, в зависимости от количества используемых датчиков и их типа. Мы рассмотрим три алгоритма, по количеству датчиков с цифровым бинарным выходом: начиная от использования одного заканчивая использованием трех датчиков линии.

1 датчик (следование по одному краю)



Для данного способа следования по линии необходим только один датчик. На самом деле робот следует не по самой линии, а по её границе, постоянно пересекая границу между темной и светлой областью разметки. Один двигатель включается, когда линия видна, но выключается, когда линия не видна, другой активируется, когда линия не видна, но выключается, когда линия видна. И таким образом робот, маневрируя из стороны в сторону, двигается вдоль границы разметки. Это отлично работает на медленных скоростях. Если датчик пересекает линию, он может развернуться и поехать в обратном направлении, если робот теряет линию, он начинает постоянно вращаться. Решение простое, но наименее эффективное и надежное. Датчик может быть в одном из двух возможных состояний 1 или 0 (см. рисунок):

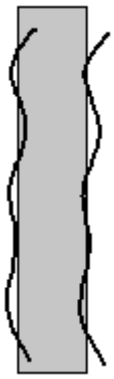
0 – линия не видна;

1 – линия видна.

Код программы 1

```
#define LINE_C A1 //пин подключения сигнальной линии датчика линии

#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

#define LIN_SPEED 150 //значение ШИМ, определяющее скорость двигателей

void setup()
{
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);

  pinMode(LINE_C, INPUT);
}

void left()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}
void right()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 0);
}
void loop()
{
  unsigned int middle = digitalRead(LINE_C);
  if (middle == 1)
  {
    right();
  }
  else
  {
    left();
  }
  delay(50);
}
```

2 датчика (исключение линии)



Принцип работы похож на первый вариант, но каждый датчик контролирует свой двигатель. Линия находится между датчиков, а они в свою очередь стараются избегать её. На большой скорости эта схема работает лучше, чем предыдущая. Но если линия будет потеряна, то робот начнёт блуждать. Это происходит потому, что робот не может отличить границы линии, и её потерю. Этот недостаток можно устранить программно, с помощью микроконтроллера. Чем меньше зазор между линией и датчиком, тем аккуратнее робот будет следовать по линии. Условия работы сенсора:

00 - граница линии потеряна;

01- линия справа;

10 - линия слева;

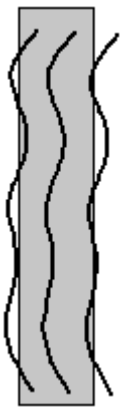
11 - не используется (если расстояние между датчиками больше чем ширина линии).

Код программы 2

```
//#define LINE_C A1 //пин подключения сигнальной линии датчика линии
#define LINE_L A2 //пин подключения сигнальной линии левого датчика линии
#define LINE_R A3 //пин подключения сигнальной линии правого датчика линии

#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

#define LIN_SPEED 120 //значение ШИМ, определяющее скорость двигателей

void setup()
{
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);

  //pinMode(LINE_C, INPUT);
  pinMode(LINE_L, INPUT);
  pinMode(LINE_R, INPUT);
}

void forward()
```

```
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}

void left()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}

void right()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 0);
}

void loop()
{
  unsigned int ls_left = digitalRead(LINE_L);
  unsigned int ls_right = digitalRead(LINE_R);
  if (ls_left > ls_right)
  {
    left();
  }
  else if (ls_left < ls_right)
  {
    right();
  }
  else
  {
    forward();
  }
  //delay(50);
}
```

3 Датчика (предвидение линии)



При добавлении третьего датчика к предыдущему дизайну, робот может, определять линии и ее края. Тем самым робот может замечать небольшой съезд

с линии. Также данная схема, легче адаптируется к меняющимся условиям, можно увеличить скорость на прямой, или настроить управление более тонко. Это одно из наиболее распространенных решений. Условия работы сенсора:

001 - линия слева;
010 -линия по центру;
011 - линия ушла немного влево;
100 - линия справа;
101 - не используется;
110 - линия ушла немного вправо;
111 - не используется (но может использоваться для слежения линии в лабиринте или на сложных трассах с перекрёстками).

Код программы 3

```
#define LINE_C A1 //пин подключения сигнальной линии датчика линии
#define LINE_L A2 //пин подключения сигнальной линии левого датчика линии
#define LINE_R A3 //пин подключения сигнальной линии правого датчика линии

#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

#define LIN_SPEED 130 //значение ШИМ, определяющее скорость двигателей

void setup()
{
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);

  pinMode(LINE_C, INPUT);
  pinMode(LINE_L, INPUT);
  pinMode(LINE_R, INPUT);
}

void forward()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}

void left_slow()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED - 40);
  analogWrite(PWM_R_PIN, LIN_SPEED + 40);
}

void left()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
```

```
  analogWrite(PWM_L_PIN, 0);
  analogWrite(PWM_R_PIN, LIN_SPEED);
}

void right_slow()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED + 40);
  analogWrite(PWM_R_PIN, LIN_SPEED - 40);
}

void right()
{
  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 0);
}

void loop()
{
  unsigned int ls_left = digitalRead(LINE_L);
  unsigned int ls_right = digitalRead(LINE_R);
  unsigned int ls_middle = digitalRead(LINE_C);

  if ((ls_left == 1) && (ls_middle == 1) && (ls_right == 0))
  {
    left_slow();
  }
  else if ((ls_left == 0) && (ls_middle == 1) && (ls_right == 1))
  {
    right_slow();
  }
  else if ((ls_left == 0) && (ls_middle == 0) && (ls_right == 1))
  {
    right();
  }
  else if ((ls_left == 1) && (ls_middle == 0) && (ls_right == 0))
  {
    left();
  }
  else
  {
    forward();
  }

  //delay(50);
}
```

Вопросы для проверки знаний

1. Какой из алгоритмов лучше? Почему?

Практическое задание

1. Замените параметры скорости так, чтобы добиться наискорейшего преодоления траектории роботом каждым из алгоритмов.

Контрольные задания

1. Напишите программу, реализующую движение по линии с одновременным объездом препятствий, размещенных на пути следования, которые необходимо объехать и вернуться на линию.

В уроке использованы материалы:
http://cxem.net/uprav/uprav40.php

УРОК 13. ДВИЖЕНИЕ ЗА ИСТОЧНИКОМ СВЕТА

Описание урока

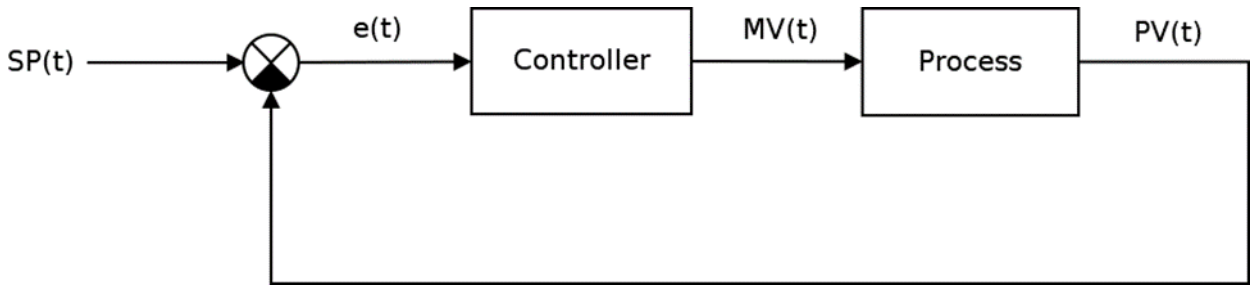
На этом уроке мы реализуем алгоритм движения робота за источником света.

Информационный блок

На прошлых занятиях вы получили знания о том, как получать информацию об освещенности с фоторезистивных датчиков света, установленных в модуле SENS1. Как вы заметили, датчики света в модуле расположены симметрично с двух сторон в переднем модуле. Мы можем получать оцифрованное значение освещенности с каждого из датчиков в диапазоне от 0 до 1023. В реальности, диапазон будет конечно же меньше, но достаточный для численного (количественного) сравнения освещенности двух датчиков: левого и правого.

Чтобы реализовать эффективный алгоритм движения за источником света, необходимо обратиться к основам теории автоматического управления и познакомится с простейшим регулятором – пропорционально-интегрально-дифференциальным регулятором, или, как его называют сокращенно, ПИД-регулятором. Несмотря на свою относительную простоту, данный тип регулятора в полной или неполной конфигурации применяется повсеместно в окружающих нас системах автоматического управления самыми разными процессами.

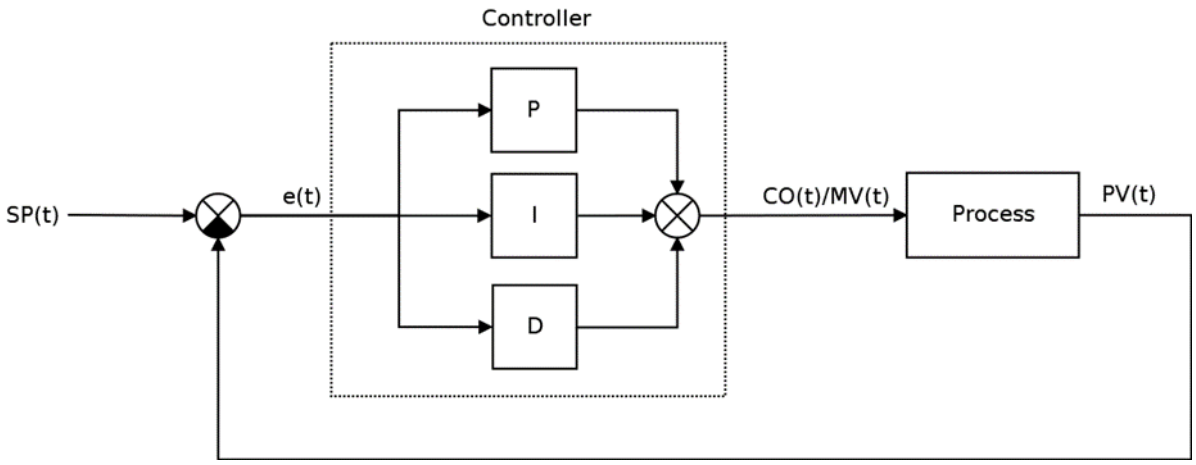
Классическая система автоматического управления представлена на следующем рисунке.



Ключевым элементом любой система автоматического управления является *регулятор*, представляющий из себя устройство, которое следит за состоянием объекта управления и обеспечивает требуемый закон управления. Процесс управления включает в себя: вычисление ошибки управления или сигнала рассогласования $e(t)$ как разницы между желаемой *уставкой* (set point или *SP*) и текущим значением управляемого параметра (process value или *PV*). Чтобы скомпенсировать эту разницу, регулятор вырабатывает управляющие сигналы (manipulated value или *MV*).

Одной из разновидностью регуляторов является *пропорционально-интегрально-дифференцирующий (ПИД) регулятор*, который формирует

управляющий сигнал, являющийся суммой трёх слагаемых: пропорционального, интегрального и дифференциального.



Где,

$e(t)$ - ошибка рассогласования,

$P = Kp \cdot e(t)$ - пропорциональная,

$I = Ki \cdot \int e(\tau)d\tau$ — интегральная,

$D = Kd \cdot (\Delta e(t))/dt$ — дифференциальная

составляющие (термы) закона управления, который в итоговом виде описывается следующими формулами:

$$e(t) = SP(t) - PV(t),$$

$$MV(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt},$$

Пропорциональная составляющая P — отвечает за т. н. пропорциональное управление, смысл которого в том, что выходной сигнал регулятора, противодействует отклонению регулируемой величины (ошибки рассогласования - *невязкой*) от заданного значения. Чем больше ошибка рассогласования, тем больше командное отклонение регулятора. Это самый простой и очевидный закон управления. Недостаток пропорционального закона управления заключается в том, что регулятор никогда не стабилизируется в заданном значении, а увеличение коэффициента пропорциональности всегда приводит к автоколебаниям. Именно поэтому совместно с пропорциональным законом управления приходиться использовать интегральный и дифференциальный.

Интегральная составляющая I накапливает (интегрирует) ошибку регулирования, что позволяет ПИД-регулятору устранять статическую ошибку (установившуюся ошибку, остаточное рассогласование). Или другими словами: интегральное звено всегда вносит некоторое смещение и если система подвержена некоторыми постоянным ошибкам, то оно их компенсирует (за счет своего смещения). А вот если же этих ошибок нет или они пренебрежительно малы, то эффект будет обратным — интегральная составляющая сама будет

вносить ошибку смещения. Именно по этой причине её не используют, например, в задачах сверхточного позиционирования. Ключевым *недостатком* интегрального закона управления является эффект насыщения интегратора (Integrator windup).

Дифференциальная составляющая *D* пропорциональна темпу изменения отклонения регулируемой величины и предназначена для противодействия отклонениям от целевого значения, которые прогнозируются в будущем. Примечательно то, что дифференциальная компонента устраняет затухающие колебания. Дифференциальное регулирование особенно эффективно для процессов, которые имеют большие запаздывания. Недостатком дифференциального закона управления является его неустойчивость к воздействую шумов (Differentiation noise).

Таким образом, в зависимости от ситуации могут применятся П-, ПД-, ПИ- и ПИД-регуляторы, но основным законом управления в основном является пропорциональный (хотя в некоторых специфических задачах и могут использоваться исключительно только звенья дифференциаторов и интеграторов).

В предлагаемом вам решении задачи движении робота на источник света применяется простой пропорциональный регулятор. Однако, вы можете поэкспериментировать и разработать более продвинутое решение с более высоким качество регулирования.

Алгоритм работы робота следующий:

1. Считать данные с двух датчиков света;
2. Вычислить значение ошибки – разницы между освещенностью левого и правого датчиков;
3. Вычислить среднее арифметическое освещенности двух датчиков;
4. Вычислить абсолютное значение пропорциональной составляющей, умножив модуль ошибки на коэффициент пропорциональности (определяется эмпирически);
5. Если модуль пропорциональной составляющей больше предельного значения разницы скоростей двигателей, ограничить модуль пропорциональной составляющей этим предельным значением;
6. Проверить, превышает ли пороговое значение среднее арифметическое освещенности двух датчиков (пороговое значение задается эмпирически – служит служи для того, чтобы робот не двигался в отсутствии акцентированного источника света, направленного на датчики);
7. Если порог освещенности превышен (присутствует акцентированный источник света), двигаться на источник света, увеличив значение ШИМ соответствующего двигателя на величину модуля пропорциональной составляющей (т. о. совершая поворот в сторону источника света).
8. Если в шаге 6 порог не превышен – остановить оба двигателя (акцентированного источника света нет «в поле зрения» робота).

Код программы

```
#define LDR_L_PIN A2
#define LDR_R_PIN A3
#define LDR_ANG_PWM 65
#define LDR_LIN_PWM 105
#define Kp 0.4
#define MIN_LDR 190

#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

void setup()
{
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);

  pinMode(LDR_R_PIN, INPUT);
  pinMode(LDR_L_PIN, INPUT);
}

void move_to_light()
{
  unsigned int right = analogRead(LDR_R_PIN);
  unsigned int left = analogRead(LDR_L_PIN);

  float error = left - right;
  float P = Kp * abs(error);

  if ((unsigned int)P > LDR_ANG_PWM) P = LDR_ANG_PWM;
  float mean = (right + left) / 2.0;

  if (mean < MIN_LDR)
  {
    if (mean >= left)
    {
      digitalWrite(DIR_L_PIN, LOW);
      digitalWrite(DIR_R_PIN, LOW);
      analogWrite(PWM_L_PIN, LDR_LIN_PWM);
      analogWrite(PWM_R_PIN, (unsigned int)P + LDR_LIN_PWM);
    }

    if (mean >= right)
    {
      digitalWrite(DIR_L_PIN, LOW);
      digitalWrite(DIR_R_PIN, LOW);
      analogWrite(PWM_L_PIN, (unsigned int)P + LDR_LIN_PWM);
      analogWrite(PWM_R_PIN, LDR_LIN_PWM);
    }
  }
  else
  {
    digitalWrite(DIR_L_PIN, LOW);
    digitalWrite(DIR_R_PIN, LOW);
    analogWrite(PWM_L_PIN, 0);
    analogWrite(PWM_R_PIN, 0);
  }
```

```
}
void loop()
{
  move_to_light();
}
```

Практическое задание

1. Реализуйте код программы, при котором робот отъезжает от источника света.
2. Вы могли заметить, что датчики освещенности дают немного разные показания при одной и той же освещенности. В результате, при превышении порога MIN_LDR, робот может достаточно круто поворачивать в какую-либо сторону и двигаться не совсем так, как нам бы хотелось. Реализуйте функцию калибровки датчиков освещенности перед запуском основного цикла программы.

В уроке использованы материалы:
https://habr.com/post/345972/