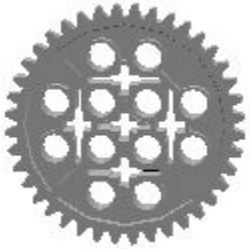
Программирование в среде RobotC



Сергей Александрович Филиппов

Президентский Физико-математический лицей №239

План занятий

- Введение в RobotC
- Вывод данных
- Графика
- Датчики
- Моторы
- Применение регуляторов
- Воспроизведение по памяти


Загрузка операционной системы

1. Тип контроллера

Меню Robot -> Platform Type->Lego Mindstorms->EV3

2. Ядро

Меню Robot -> Download EV3 Linux Kernel-> Standart File

Выполнять при включенном EV3, в процессе загрузки не трогать (до 4 мин)

3. Прошивка

Меню Robot -> Download Firmware> Standart File

Выполнять при включенном EV3, занимает 1-2 секунды


Простейшая программа

```
task main()
{
  displayTextLine(0, "Hello, world!");
  wait1Msec(10000);
}
```

Меню File -> Save as... — сохранение

F7 — проверка

F5 — загрузка на контроллер


Загрузка и отладка программы

F7 — компиляция и проверка

F5 — загрузка программы

Start — запуск программы на NXT/EV3

Step — пошаговое выполнение

Не отключать кабель USB при открытом окне дебаггера!


Форматированный вывод

```
task main()
{
  float a=5, b=4, c=1;
  int a=5, b=4;
  displayTextLine(0, "a=%d b=%d",a,b);
  displayTextLine(1, "%d+%d=%d",a,b,a+b);
  displayTextLine(4, "%f/%f=%4.2f",a,b,a/b);
  for(int i=1;i<=b;i++)
    c=c*a;
  displayTextLine(5, "%d^%d=%d",a,b,c);
  wait1Msec(10000);
}
```


Команды ожидания

- **`wait1Msec(1);`**
- **`sleep(1);`**

- Жди 1 миллисекунду (синонимы)

- **`wait1Msec(N);`**
- **`sleep(N);`**

- Жди N мс

- **`wait10Msec(N);`**

- Жди N*10 мс

- ***`while(УСЛОВИЕ);`***
- **`while(УСЛОВИЕ) sleep(1);`**

- Жди, пока выполняется условие


Управление моторами

```
task main()
{
  motor[motorB]=100;  // полный вперед
  motor[motorC]=100;
  wait1Msec(2000);    // по времени

  motor[motorB]=-50;                // поворот налево
  motor[motorC]=50;

  nMotorEncoder[motorB]=0;          // по энкодеру
  while(nMotorEncoder[motorB]>-239) sleep(1);

  motor[motorB]=0;    // остановка
  motor[motorC]=0;
}
```


Поворот с помощью гироскопического датчика

```
task main()
{
  int angle=SensorValue[Gyro];  // Запомнили текущее
  while (true)                  // значение угла
  {
     motor[motorLeft] = 20;
     motor[motorRight] = -20;
     angle=angle+90;     // Увеличим угол по часовой
     while (SensorValue[Gyro] < angle)
       sleep(1);
     motor[motorLeft] = 40;
     motor[motorRight] = 40;
     sleep(2000);
  }
}
```


Управление скоростью

```
task main()
{
  for (int i=1; i<=100; i++) // разгон 1 секунду
  {
    motor[motorB]=i;
    motor[motorC]=i;
    wait1Msec(10);
  }
  wait1Msec(1000);

  // Добавить плавное торможение

}
```


Параллельное управление скоростью

```
int mB=0, mC=0, step=5; //Скорости моторов и шаг
task motors()
{
  while(true)
  {
    int b=mB-motor[motorB];
    motor[motorB]=motor[motorB]+sgn(b)*step;
    // То же с мотором C – добавить самостоятельно
    wait1Msec(10);
  }
}
task main()
{
  startTask(motors);  // Запуск параллельной задачи
  mB=mC=100;          // Задаем любую скорость
  wait1Msec(2000);
  mB=mC=-100;
  wait1Msec(2000);
  ...
  stopTask(motors);
}
```

Контроль управления скоростью

- Необходимо ограничение модуля скорости не более 100
- На малых отклонениях необходимо повышение точности

```
int mB=0, mC=0, step=25;
task motors()
{
  while(true)
  {
    if (abs(mB)>100) mB=sgn(mB)*100;
    int b=mB-motor[motorB];
    if (abs(b)>step)
      motor[motorB]=motor[motorB]+sgn(b)*step;
    else
      motor[motorB]=mB;
    // То же с мотором C – добавить самостоятельно
    wait1Msec(10);
  }
}
```


Доступ к энкодерам без обнуления

- К энкодерам и моторам нельзя обращаться из разных задач
- Задаем глобальные переменные, которые содержат актуальные значения энкодеров

```
int mB=0, mC=0, step=25, enB=0, enC=0;
task motors()
{ ...
  { enB=nMotorEncoder[motorB];
    // То же с мотором C – добавить самостоятельно
    ...
    wait1Msec(10);
  }
} task main()
{ ...
  int enB_cur=enB;
  mB=50;
  mC=-50;
  while(enB < enB_cur + 239) sleep(1); // Поворот по энкодеру
  ...
}
```

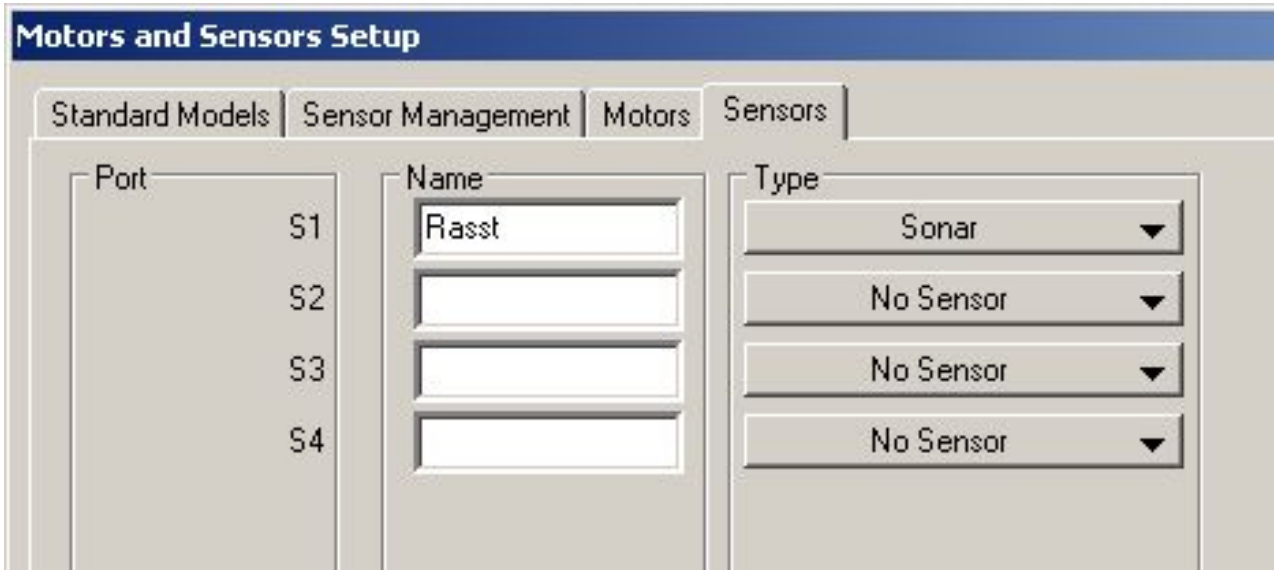

Доступ к энкодерам с обнулением

- В основной задаче для обнуления задаем enB_null=1

```
int mB=0, mC=0, step=25, enB=0, enC=0, enB_null=0, ...;
task motors()
{ ...
  { if (enB_null)
    {
      nMotorEncoder[motorB]=0;
      enB_null=0;
    }
    enb = nMotorEncoder[motorB];
    // То же с мотором C – добавить самостоятельно
    sleep(10);
  }
}
task main()
{  ...
   enB_null=1; sleep(11);
   while(enB<239) sleep(1);
```

Подключение датчика

Меню Robot -> Motors and Sensors Setup -> Sensors



#pragma config(Sensor, S1, Rasst, sensorSONAR)

```
while(SensorValue[S1]>25) // или while(SensorValue[Rasst]>25)
```

Путешествие по комнате

```
#pragma config(Sensor, S1, Rasst, sensorEV3_Ultrasonic)
task main()
{
  while(true) {
    motor[motorB]=100;  // полный вперед
    motor[motorC]=100;
    while(SensorValue[Rasst]>25) sleep(1);
    motor[motorB]=-50; // отъезд с разворотом
    motor[motorC]=-10;
    nMotorEncoder[motorB]=0;         // по энкодеру
    while(nMotorEncoder[motorB]>-400) sleep(1);
  }
}
```

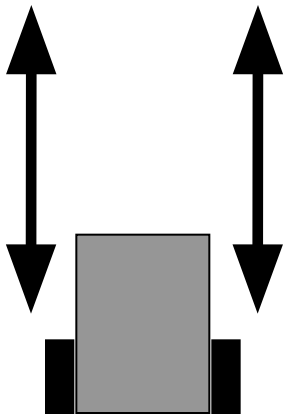

Вывод показаний датчиков на экран

```
...
while(SensorValue[Rasst]>25)
{
  displayBigTextLine(0, "%d", SensorValue[Rasst]);
  sleep(10);
}
...
while(nMotorEncoder[motorB]>-400)
{
  displayBigTextLine(2, "%d", nMotorEncoder[motorB]);
  sleep(10);
}
```


Пропорциональный регулятор: синхронизация моторов

- Пусть e2 и e3 – показания датчиков оборотов моторов B и C. Их надо будет обнулить перед началом движения. Регулятор определяется следующим образом:

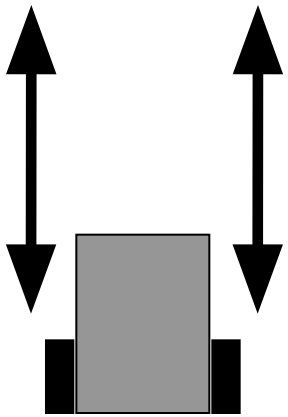
```
int v=50, k=2, u;
nMotorEncoder[motorB]=0;
nMotorEncoder[motorC]=0;
while(true)
{
  int e2=nMotorEncoder[motorB];
  int e3=nMotorEncoder[motorC];
  u=k*(e3-e2);
  motor[motorB]=v+u;
  motor[motorC]=v-u;
  wait1Msec(1);
}
```



Синхронизация при путешествии по комнате

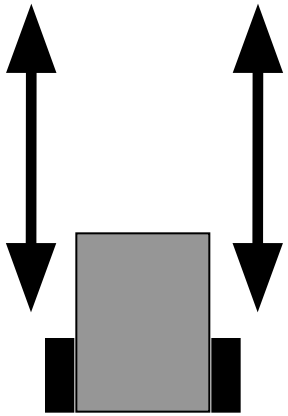
- Для синхронизации движения вперед необходимо перед циклом ожидания объекта обнулить энкодеры:

```
int v=50, k=2, u;
while(true){
  nMotorEncoder[motorB]=0;
  nMotorEncoder[motorC]=0;
  while(SensorValue[Rasst]>25)
  {
    int e2=nMotorEncoder[motorB];
    int e3=nMotorEncoder[motorC];
    u=k*(e3-e2);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
...
}
```



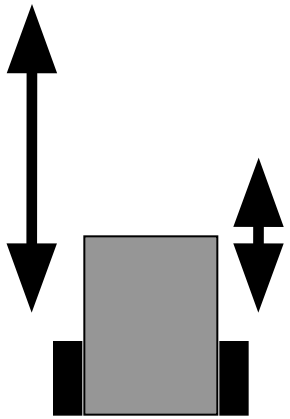
Параллельное управление моторами

```
int v=50, k=2, u;
task preg()  // Объявление задачи
{
  nMotorEncoder[motorB]=0;
  nMotorEncoder[motorC]=0;
  while(true){
    int e2=nMotorEncoder[motorB];
    int e3=nMotorEncoder[motorC];
    u=k*(e3-e2);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
task main()  // Основная задача
{
  startTask(preg); // Запуск параллельной задачи
  wait1Msec(10000); // Здесь могут быть полезные действия
  stopTask(preg);  // Остановка параллельной задачи
}
```



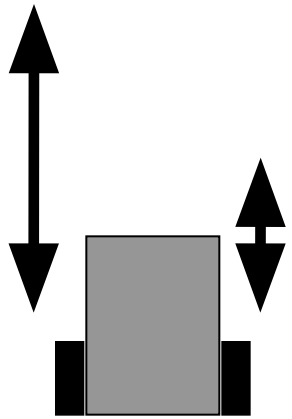
Параллельное управление моторами

```
int v=50, k=2, u, DELTA=0;
task preg()  // Объявление задачи
{
...
    u=k*(e3-e2 + DELTA);
...
  }
}
task main()  // Основная задача
{
  startTask(preg);
  wait1Msec(2000);
  DELTA=DELTA+450; // Изменение разности энкодеров
  wait1Msec(2000);
  DELTA=DELTA+450;
  stopTask(preg);
}
```



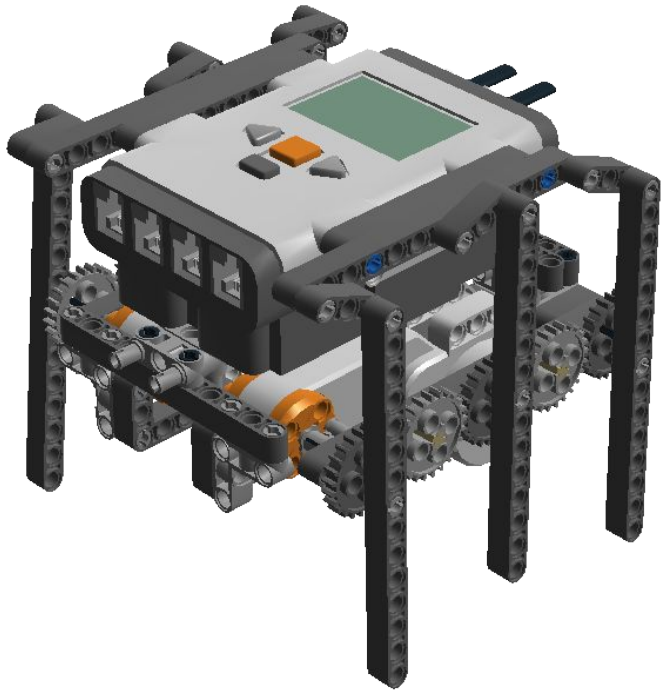
Параллельное управление моторами

```
int v=50, k=2, u, DELTA=0;
task preg()  // Объявление задачи
{
...
    u=k*(e3-e2 + DELTA);
...
  }
}
task main()  // Основная задача
{
  startTask(preg);
  while(true)
  {
    wait1Msec(2000);
    DELTA=DELTA+450; // Изменение разности энкодеров
  }
}
```



Управление шагающим роботом

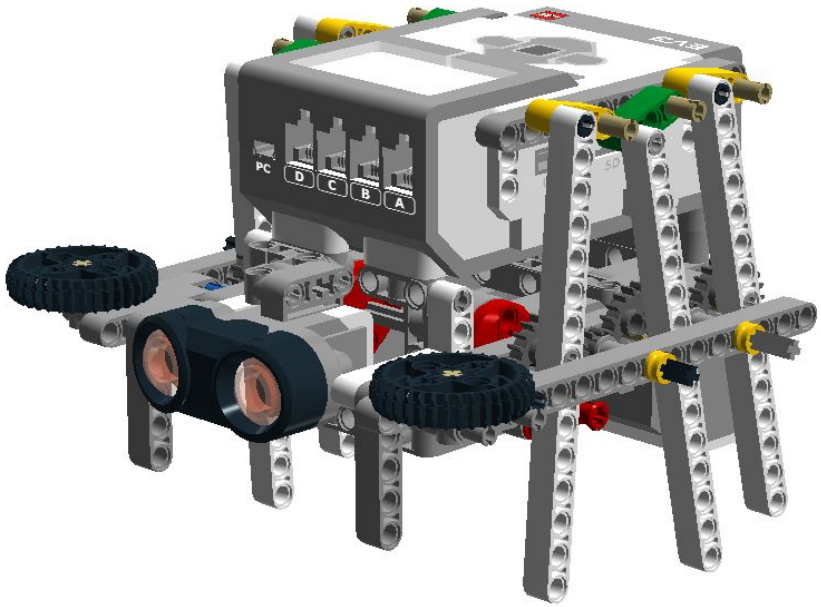
```
int v=50, k=2, u, DELTA=0, i=1;
task preg()  // Объявление задачи
{
...
    u=k*(e3-e2 + DELTA*i);
...
  }
}
task main()  // Основная задача
{
  ...
  {
    wait1Msec(4000);
    DELTA=DELTA+360*4; // Изменения с учетом периода
  }
}
```



Управление шагающим роботом с датчиком расстояния

- Робот двигается до препятствия
- На поворот выделяется время
- Для синхронизации соблюдается период обращения моторов
- Строится сценарий движения

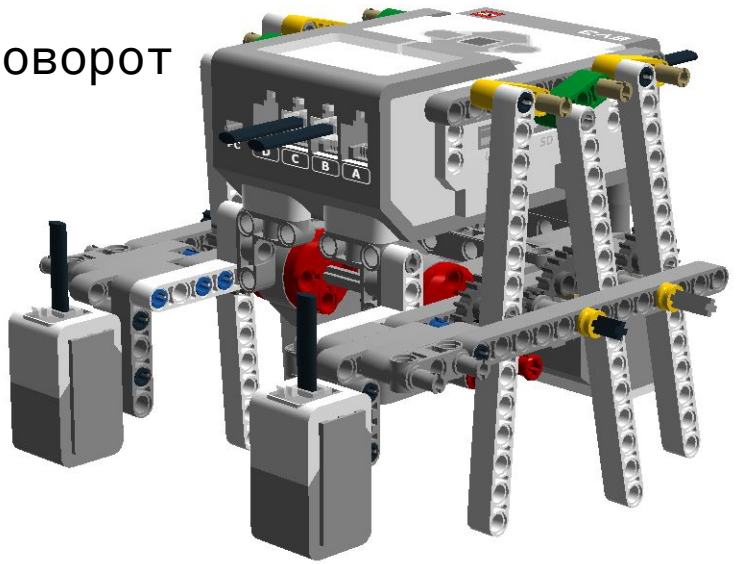
```
task main()  // Основная задача
{
  ...
  {
    while(SensorValue[S1]>25)
      sleep(1);
    delta=delta+360*4;
    sleep(2000);
  }
}
```



Шагающий робот на линии

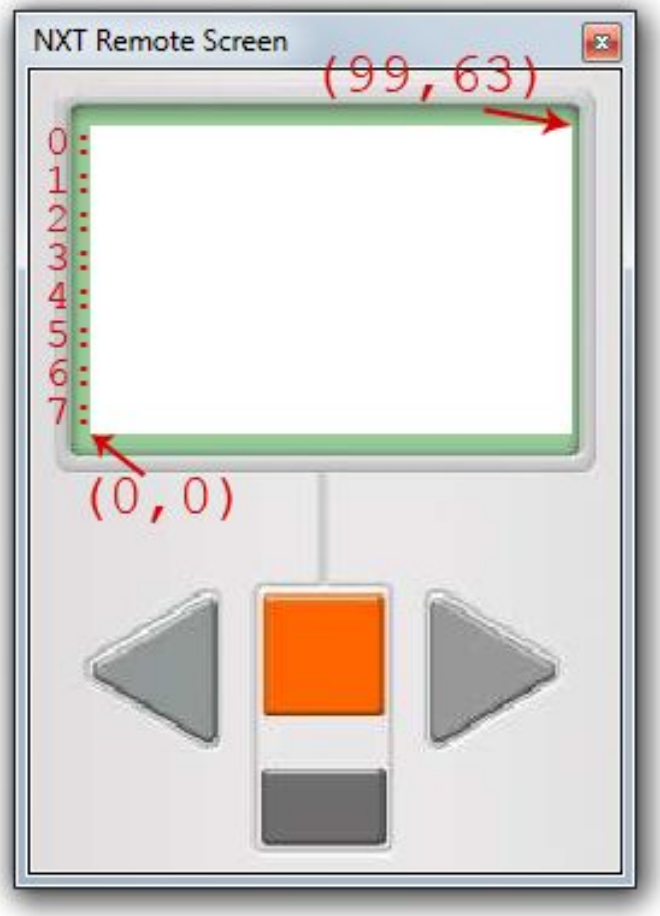
- Калибровка на старте
- Определение динамической ошибки как коэффициента периода поворота

```
task main()
{
   startTask(preg);
   int es=SensorValue[S1]-SensorValue[S2];
   while(true){
     int e=(SensorValue[S1]-SensorValue[S2]-es)/15;
     delta=delta+360*e;
     sleep(abs(e)*500+1); //Время на поворот
   }
   stopTask(preg);
}
```



Графика на экране

- **NXT**:
  - 100x64 пикселя
  - 8 текстовых строк (0..7)
- **EV3**:
  - 178x128 пикселей
  - 16 текстовых строк (0..15)
- Идентичные команды



Отображение громкости звука на экране NXT

```
#pragma config(Sensor, S1,   Zvuk,     sensorSoundDBA)

task main()
{
  int d=0, x,y;
  while(true)
  {
    d=SensorValue[Zvuk];
    x=50-d/2;
    y=32+d/2;
    drawCircle(x,y,d);
    wait1Msec(40);
    eraseRect(x,y,x+d+1,y-d-1);
  }
}
```

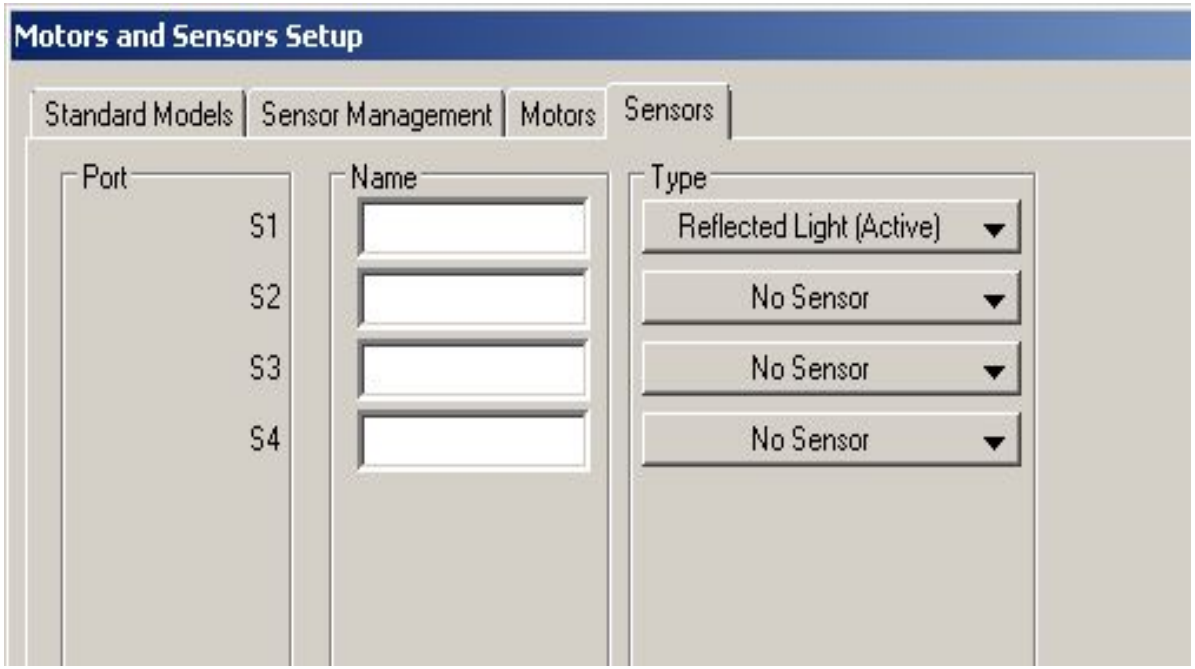
Составьте аналогичный алгоритм с использованием функций

fillEllipse и

eraseEllipse

Подключение датчика

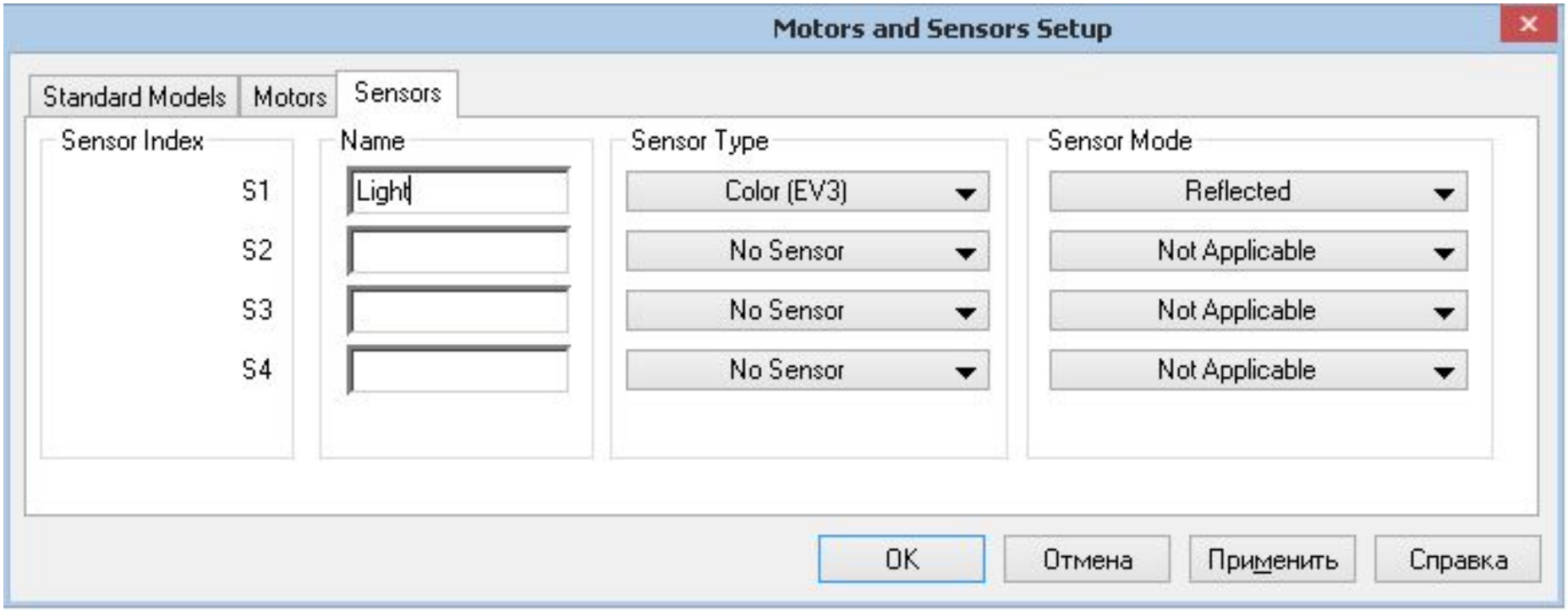
Меню Robot -> Motors and Sensors Setup -> Sensors



#pragma config(Sensor, S1, , sensorLightActive)


Подключение датчика EV3

Меню Robot -> Motors and Sensors Setup -> Sensors



#pragma config (Sensor, S1, Light, sensorEV3_Color)

График показаний датчика

- Составьте алгоритм вывода на экран графика показаний датчика света.
  - Частота 10 замеров в секунду
  - Длительность 17,8 секунд (178 замеров)
  - Масштабирование 127/100
  - Используйте цикл
  - Вывод точки
  - Вывод линии

```
drawLine(x1,y1,x2,y2);
```

```
for(int x=0; x<178; x++)
{ ...
  setPixel(x,y);
}
```


Отображение показаний датчика в виде изменяющегося эллипса

```
#pragma config(Sensor, S1, Light,    sensorEV3_Color)

task main()
{
  int d=0, x,y;
  while(true)
  {
    d=SensorValue[Light];
    x=88-d/2;
    y=63+d/2;
    drawCircle(x,y,d);
    sleep(40);
    eraseRect(x,y,x+d+1,y-d-1);
  }
}
```

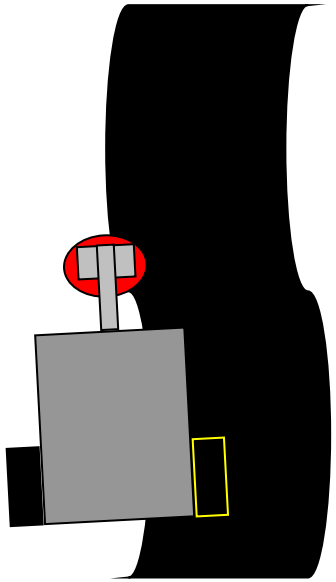
Составьте аналогичный алгоритм с использованием функций

fillEllipse и

eraseEllipse

Релейный регулятор: движение вдоль границы черного и белого с помощью датчика освещенности

```
int grey=15;  // Приближенное значение серого
task main()
{
  while (true)  // Бесконечное повторение
  {
    if (SensorValue[S1]>grey) // Проверка
    {
      motor[motorB]=100; // Направо по дуге
      motor[motorC]=0;
    }
    else
    {
      motor[motorB]=0;   // Налево по дуге
      motor[motorC]=100;
    }
    wait1Msec(1);
}
```



Пропорциональный регулятор

В задачах автоматического регулирования управляющее воздействие u(t) обычно является функцией динамической ошибки – отклонения e(t) регулируемой величины x(t) от ее заданного значения $x_0(t)$:

$$\mathbf{e(t)=x_0(t)-x(t)}$$

Пропорциональный регулятор – это устройство, оказывающее управляющее воздействие на объект пропорционально его отклонению от заданного состояния.

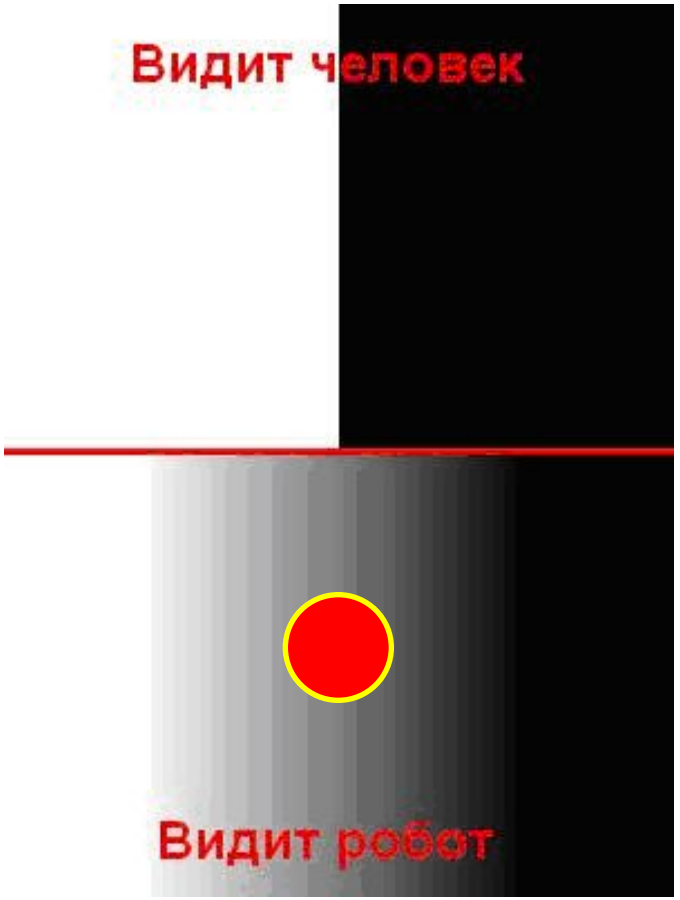
$$\mathbf{u_0(t)=ke}$$

Здесь k – это коэффициент усиления регулятора.


Пропорциональный регулятор: движение по линии

Также как и в релейном регуляторе, необходимо определить среднее значение grey между черным и белым. Это будет то состояние датчика освещенности s1, к которому должна стремиться система.

```
while(true)
{
  u=k*(sensorValue[S1]-grey);
  motor[motorB]=50+u;
  motor[motorC]=50-u;
  wait1Msec(1);
}
```



Пропорциональный регулятор: вычисление коэффициента усиления

- Базовая скорость робота ***v***
- Максимальная скорость ***vmax***
- Минимальная ***vmin***
- Минимальное значение скорости влияет на крутизну поворотов

1. Найти максимальное управляющее воздействие ***umax***

для получения предельной скорости на моторе - это наибольшее из чисел ***vmax-v*** и ***v-vmin***

2. Найти максимальную ошибку

emax = (white - black) / 2

3. Найти ориентировочное значение коэффициента усиления ***k***.

k = umax / emax

Пример

Дано:

v = 50

vmax = 100

vmin = 0

white = 50

black = 10

Решение:

umax = 100 - 50 = 50

emax = (50 - 10) / 2 = 20

k = 50 / 20 = 2.5

Ответ: коэффициент усиления k = 2.5.

Параллельные задачи

```
task line()  // Объявление задачи
{
  while(true)
  {
     // Здесь должен быть регулятор для движения по линии
  }
}

task main()  // Основная задача
{
  startTask(line); // Запуск параллельной задачи

  wait1Msec(17800); // Здесь могут быть полезные действия

  stopTask(line);  // Остановка параллельной задачи
}
```


Параллельные задачи - 2

```
task line()  // Объявление задачи
{
  while(true)
  {
     // Здесь должен быть регулятор для движения по линии
  }
}

task main()  // Основная задача
{
  startTask(line); // Запуск параллельной задачи
  for(int x=0; x<178; x++)
  {
    ...               // Рисование графика 17,8 с
    wait1Msec(100);
  }
  stopTask(line);  // Остановка параллельной задачи
  wait1Msec(30000); // Посмотреть график
}
```


Параллельные задачи - 3

```
int svalue=0;    // Глобальная переменная
task line()
{
  while(true)
  {
     svalue=SensorValue[S1];  // Показания датчика в переменную
     // Здесь должен быть регулятор для движения по линии
  }
}
task main()  // Основная задача
{
  StartTask(line); // Запуск параллельной задачи
  for(int x=0; x<178; x++)
  {
    y=svalue;     // Защита от коллизий
    ...
  }
  StopTask(line);  // Остановка параллельной задачи
  motor[motorB]=motor[motorC]=0;   // Остановить моторы

  wait1Msec(30000); // Посмотреть график
}
```

Параллельные задачи – 4 – массивы

```
int mas[178];    // Массив в RobotC объявляется глобально
task line()
  ...
task main()  // Основная задача
{
  StartTask(line); // Запуск параллельной задачи
  for(int x=0; x<178; x++)
  {
    mas[x]=svalue;     // Запись в массив без рисования
    sleep(100);
  }
  StopTask(line);  // Остановка параллельной задачи
  motor[motorB]=motor[motorC]=0;   // Остановить моторы

  for(int x=0; x<178; x++)
  {
    y=mas[x];     //  Рисование графика после остановки
    ...
  }

  wait1Msec(30000); // Посмотреть график
}
```

Параллельные задачи – 5 – массивы

```
int mas[178];    // Массив
task line()
  ...
task main()  // Основная задача
{
  StartTask(line); // Запуск параллельной задачи
  for(int x=0; x<178; x++)
  {
    mas[x]=SensorValue[S1];     // Запись в массив без рисования
    sleep(100);
  }
  StopTask(line);  // Остановка параллельной задачи
  motor[motorB]=motor[motorC]=0;   // Остановить моторы

  while(!getButtonPress(buttonEnter)) sleep(1); // Жди нажатия
  for(int x=0; x<178; x++)
  {
    y=mas[x];     //  Рисование графика после остановки
    ...
  }
  wait1Msec(30000); // Посмотреть график
}
```

Параллельное управление моторами

```
int v=50, delta=0;          // Глобальные переменные
task preg()             // Параллельная задача
{
  float e, u, k=2;
  while(true)  {          // Синхронизация моторов на П-регуляторе
    e=nMotorEncoder[mC]-nMotorEncoder[mB]+delta;
    u=e*k;
    motor[mB]=v+u;
    motor[mC]=v-u;
    wait1Msec(1);
  }
}
task main()                 // Основная задача
{
  nMotorEncoder[motorB]=nMotorEncoder[motorC]=0;
  startTask(preg);      // Запуск параллельной задачи
  for (int i=0;i<4;i++) {  // Движение по квадрату
    wait1Msec(2000);
    delta=delta+500;
  }
  v=0;
}
```

```
task line()
{
  while(true) {
    // П-регулятор
    //движения по линии
  }
}
task preg()
{
  while(true) {

   e=alpha-nMotorEncoder[mo
   torB];
    // П-регулятор
    // положения моторов
  }
}
```

```
task main()
{
  // Обнулить энкодеры
  startTask(line);
  for(int i=0;i<178;i++)  {
  mas1[i]=nMotorEncoder[motorB];
  mas2[i]=nMotorEncoder[motorC];
  sleep(100); // Запись массивов
  }
  stopTask(line);
// Стоп моторы, (обнул. энк.???)
  // Жди нажатия
  startTask(preg);
  for(...) {
    alpha=mas1[i];
    // Воспроизведение
}
```

Запоминание и воспроизведение пути

```
int alpha=0, beta=0;
float kp=0.5;
task preg()
{
  while(true) {
    e=alpha-nMotorEncoder[motorB];
    motor[motorB]=e*kp;
    e=beta-nMotorEncoder[motorC];
    motor[motorC]=e*kp;
    sleep(1);
  }
}
```

П-регулятор положения моторов

```
task line()
{
  while(true) {
    // П-регулятор
    //движения по линии
  }
}
task preg()
{
  while(true) {

   e=alpha-nMotorEncoder[mo
   torB];
    // П-регулятор
    // положения моторов
  }
}
```

```
task main()
{
  // Обнулить энкодеры
  startTask(line);
  for(int i=0;i<178;i++)  {
  mas1[i]=nMotorEncoder[motorB];
  mas2[i]=nMotorEncoder[motorC];
  sleep(100);// Запись массивов
  }
  stopTask(line);
  startTask(preg);
  for(int i=177;i>=0;i--)  {
    alpha=mas1[i];
    beta=mas2[i];
    sleep(100);
}  // Воспроизведение
```

Запоминание и воспроизведение пути

```
for(int j=0;j<100;j++)
{  // По энкодерам
  int eB = alpha - nMotorEncoder[motorB];
  int eC = beta - nMotorEncoder[motorC];
  mb = eB * k;
  mc = eC * k;
   // По датчикам
  int u=(SensorValue[Light1]-SensorValue[Light2]-est)*k;
  mb = mb+u;
  mc = mc-u;
  if (mb>100) mb=100;  // Ограничение скорости
  if (mc>100) mc=100;
  motor[motorB]=mb;
  motor[motorC]=mc;
  wait1Msec(1);
}
```

Совмещенный регулятор воспроизведения

```
  while(i<size)
  {
    int eB=nMotorEncoder[motorB];
    int eC=nMotorEncoder[motorC];
    if ((e1[i]<=eB+delta) &&
        (e2[i]<=eC+delta))
      i++;
// Уровень отставания энкодеров
erm=((e1[i]-eB)-(e2[i]-eC))*km;
    if (erm>0){  //B otstaet
      mb=v;
      mc=v - erm;
    }
    else{   //C otstaet
      mc=v;
      mb=v + erm;
    }
```

```
// Корректировка по датчикам
int
 u=(SensorValue[Light1]-SensorV
 alue[Light2]-est)*k;
  mb=mb+u;
  mc=mc-u;

  if (mb>100) mb=100;
  if (mc>100) mc=100;

  motor[motorB]=mb;
  motor[motorC]=mc;
  wait1Msec(1);
}
```

Воспроизведение на предельной скорости

Благодарю за внимание!

Сергей Александрович Филиппов
Президентский физико-математический лицей № 239
Санкт-Петербург
safilippov@gmail.com