Сергей Александрович Филиппов

ОСНОВЫ РОБОТОТЕХНИКИ НА БАЗЕ КОНСТРУКТОРА LEGO MINDSTORMS NXT. ЗАНЯТИЕ 6. ОРИЕНТАЦИЯ НА МЕСТНОСТИ: ОБЪЕЗЖАЕМ СТЕНЫ

Когда-нибудь в недалеком будущем роботы выйдут в коридоры, на улицы и станут нашими соседями не только по лабораториям. Движение по дорожной разметке (черной линии на белом фоне) было рассмотрено в предыдущей статье. Теперь научим робота объезжать стены.

ДВИЖЕНИЕ ВДОЛЬ СТЕНКИ

Решим такую задачу. Робот должен двигаться вдоль стенки на заданном расстоянии L (рис. 1), которое определяется в момент старта. Предположим, что левое колесо робота управляется мотором B, правое – мотором C, а датчик расстояния, подключенный к порту 1, закреплен несколько впереди корпуса тележки (это важно!) и направлен на стенку справа по ходу движения (рис. 5–7).

Расстояние до стенки в настоящий момент времени, которое показывает датчик, обозначим $S1$. Измеряется оно в сантиметрах.

Моторы двигаются со средней скоростью 50% от максимума, но при отклонении от заданного курса на них осуществляется управляющее воздействие u. Обозначим это следующим образом:

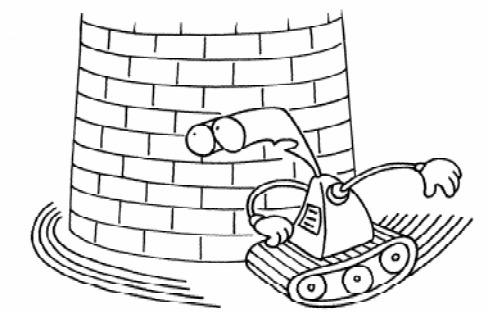
```
Motor[MotorB]=50+u;
Motor[MotorC]=50-u;
```

Осталось определить, чему будет равно управляющее воздействие. Это нетрудно сделать с помощью пропорционального регулятора:

$$u = k*(S1 - L).$$

Таким образом, при $S1 = L$ робот не меняет курса и едет прямо. В случае отклонения его курс корректируется. Здесь k – это некоторый усиливающий коэффициент, определяющий воздействие регулятора на систему. Для робота NXT средних размеров коэффициент k может колебаться от 1 до 10, в зависимости от многих факторов. Предлагаем подобрать его самостоятельно (рис. 3).

Та же программа на языке RobotC. Не забудьте объявить датчик ультразвука (Sonar) через меню **Robot → Motor and Sensors Setup**. Желательно задать датчику имя, которое будет использоваться вместо $S1$.



Робот должен двигаться вдоль стенки...

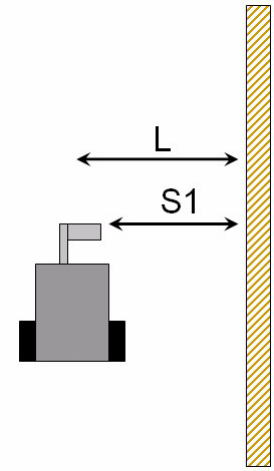


Рис. 1. Задача движения вдоль стенки на расстоянии *L*

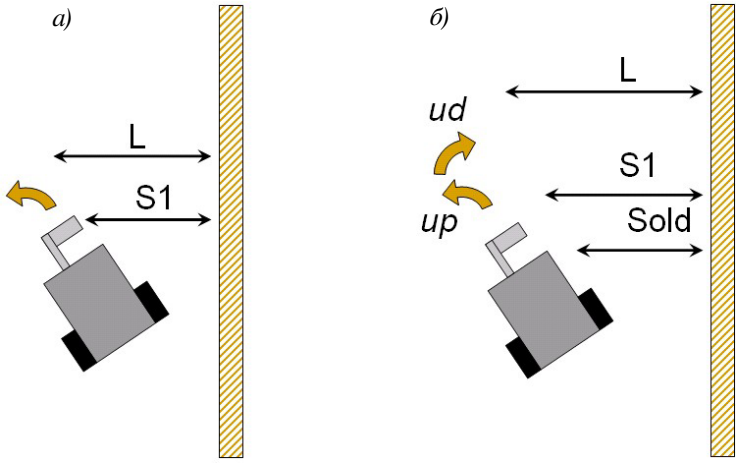


Рис. 2. Проблема пропорционального регулятора – потеря контакта со стенкой (*а*). Необходима дифференциальная составляющая (*б*)

```
task main()
{
  float u, k=3;
  int L=SensorValue[S1];
  while(true)
  {
     u=k*(SensorValue[S1]-L);
     motor[motorB]=50+u;
     motor[motorC]=50-u;
     wait1Msec(1);
  }
}
```

В данном случае П-регулятор будет эффективно работать только при малых углах отклонения. Кроме того, движение практически всегда будет происходить по волнообразной траектории. Сделать регулирование более точным позволит введение новых принципов, учитывающих отклонение робота от курса.

ПРОПОРЦИОНАЛЬНО-ДИФФЕРЕНЦИАЛЬНЫЙ РЕГУЛЯТОР

В некоторых ситуациях П-регулятор может вывести систему из устойчивого состояния (рис. 2*а*). Например, если робот направлен от стенки, но находится по отношению к ней ближе заданного расстояния, на моторы поступит команда еще больше повернуть от стенки, в результате чего с ней может быть утерян контакт (датчик расстояния получает отраженный сигнал практически только от перпендикулярной поверхности).

Для защиты от подобных ситуаций добавим в регулятор дифференциальную составляющую, которая будет следить за направлением движения робота (рис. 2*б*). Иными словами, вектор скорости будет влиять на управляющее воздействие.

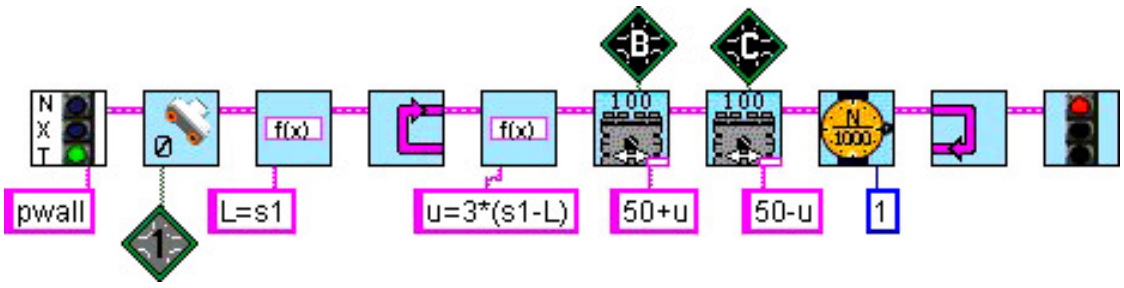


Рис. 3. Алгоритм движения вдоль стенки на основе пропорционального регулятора

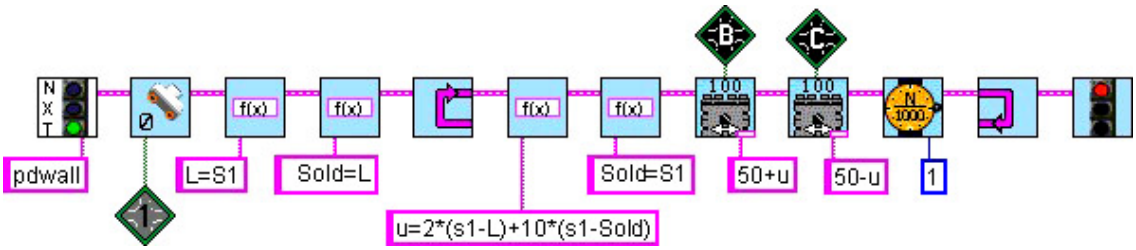


Рис. 4. Алгоритм движения вдоль стены, основанный на ПД-регуляторе

Известно, что скорость находится как

$$v = \Delta s/\Delta t,$$

где Δs – это изменение расстояния за промежуток времени Δt. Определим дифференциальную составляющую через скорость отклонения робота от заданного положения:

$$ud = k*(S1 - Sold)/\Delta t,$$

где $S1$ – текущее расстояние до стенки, $Sold$ – расстояние на предыдущем шаге.

Поскольку замеры производятся через равные промежутки времени, то Δt можно принять за константу, взяв $k2 = k*\Delta t$.

$$ud = k2*(S1–Sold)$$

Таким образом, ПД-регулятор описывается формулой из двух слагаемых

$$u = up+ud = k1*(S1 - L) + k2*(S1 - Sold).$$

Можно доказать математически, что для устойчивого достижения цели коэффициент $k2$ при дифференциальной составляющей должен превышать $k1$.

Алгоритм движения вдоль стенки на ПД-регуляторе в целом будет выглядеть так:

```
task main()
{
  float u, k1=2, k2=10;
  int Sold, L;
  Sold=L=SensorValue[S1];
  while(true)
  {
    u= k1*(SensorValue[S1]-L) +
        k2*(SensorValue[S1]-Sold);
    motor[motorB]=50+u;
    motor[motorC]=50-u;
    Sold=SensorValue[S1];
    wait1Msec(1);
  }
}
```

КОНСТРУКЦИЯ ДЛЯ ДВИЖЕНИЯ ВДОЛЬ СТЕНКИ

Базовая конструкция тележки описана в статье «Управление мобильным роботом» (№ 3, 2010). Остается только правильно прикрепить датчик расстояния и подключить его на 1 порт (рис. 5–7).

Рис. 5. Датчик расстояния слегка выносится вперед

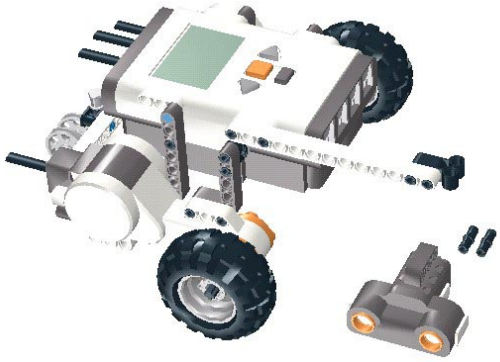


Рис. 6. Ориентацию датчика можно варьировать

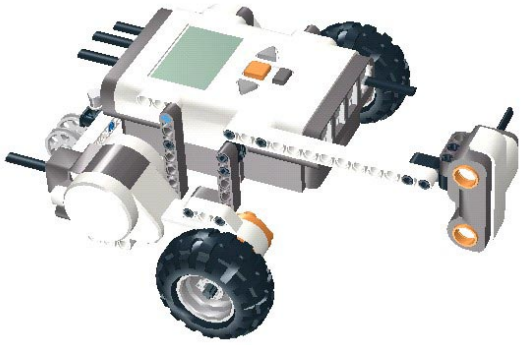
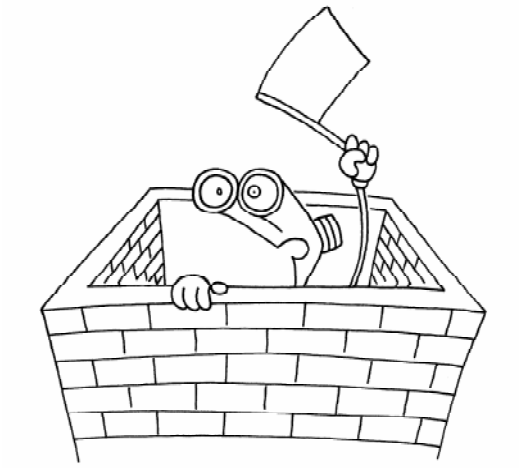


Рис. 7. Оптимальный вариант – вертикальное расположение датчика



...на пути движения будут возникать серьезные повороты, вплоть до прямых углов.

ДАТЧИК ПОД УГЛОМ

Описанный выше робот сможет объезжать стены только при малых отклонениях от прямой линии. Рассмотрим вариант, при котором на пути движения будут возникать серьезные повороты, вплоть до прямых углов. Потребуется внести модификации и в конструкцию и в программу (рис. 8).

Во-первых, робот должен будет смотреть не только направо, но и вперед. Ставить второй дальномер довольно затратно. Однако можно воспользоваться эффектом того, что ультразвуковой датчик имеет расширяющуюся область видимости (рис. 8). Это напоминает угловое зрение человека: кое-что он может увидеть краем глаза. Стоит воспользоваться таким свойством и разместить датчик расстояния не перпендикулярно курсу движения, а под острым углом (рис. 9–11). Так можно «убить сразу двух зайцев». Во-первых, робот будет видеть препятствия спереди, во-вторых, более стабильно будет придерживаться курса вдоль стены, постоянно находясь на грани видимости. Таким образом, без добавления новых устройств будет получено более эффективное использование возможностей дальномера.

Важное замечание. При старте робота его надо будет направлять датчиком строго на стену, чтобы процесс считывания начального значения прошел без помех.

Очевидно, что изменение конструкции влечет изменение коэффициентов регуля-

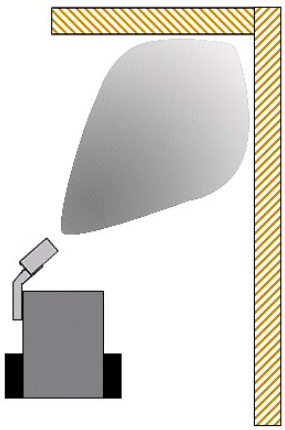


Рис. 8. Датчик расстояния устанавливается под острым углом к направлению движения

Рис. 9. Крепление для датчика размещается на левой стороне

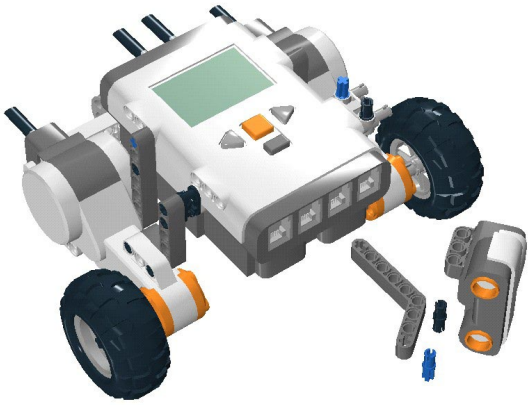


Рис. 10. Как и в первой конструкции, датчик располагается вертикально

Рис. 11. Увеличенное за счет корпуса робота расстояние до стены способствует расширению области обзора

тора $k1$ и $k2$. Обычно подбор начинается с пропорционального коэффициента при нулевом дифференциальном. Когда достигнута некоторая стабильность на небольших отклонениях, добавляется дифференциальная составляющая.

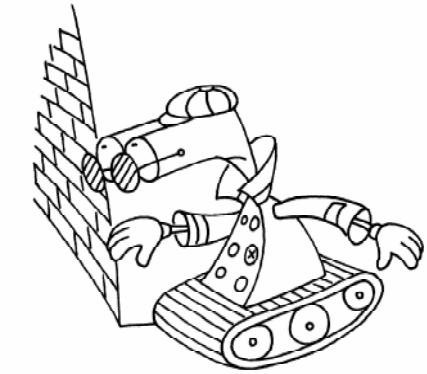
ПОВОРОТ ЗА УГОЛ

Следующим шагом необходимо ограничить реакцию робота на «бесконечность». Как известно, когда в поле видимости нет объекта, показания датчика расстояния NXT равны 250 или 255 см. Если это число попадает на пропорциональный регулятор, робот начинает крутиться на месте. А в ситуации, когда роботу следует завернуть за угол, именно это и произойдет.

Для объезда предметов потребуется ввести контроль показаний датчика расстояния: при резком изменении робот должен делать вывод о возможном повороте, который надо будет производить с другими коэффициентами или просто с постоянным значением управляющего воздействия.

Рассмотрим пример поворота направо «за угол». Если робот движется на расстоянии L от стены, то и поворот, очевидно, он будет выполнять по окружности с радиусом L (рис. 12).

Нетрудно рассчитать, каким должно быть отношение скоростей колес, чтобы радиус поворота оказался равен L. Для этого достаточно измерить расстояние между передними колесами. Пусть в нашем роботе оно будет равно $k = 16$ см, а его половина $d = 16/2 = 8$ см. Тогда левое и правое колеса движутся по окружностям радиусов, соответственно, $R_1 = L + d$



Рассмотрим пример поворота направо «за угол».

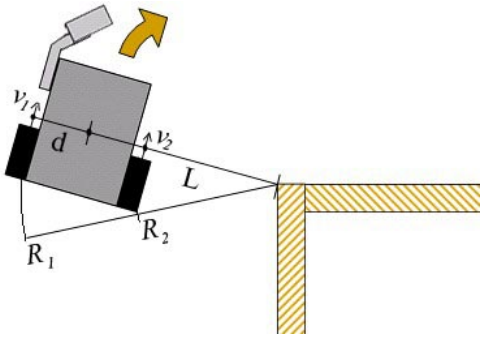


Рис. 12. Выполнение поворота при потере контакта со стенкой

и $R_2 = L - d$. Пройденные ими пути за единицу времени должны быть пропорциональны радиусам, следовательно, скорости точек крепления колес v_1 и v_2 связаны следующим отношением

$$\frac{v_1}{v_2} = \frac{R_1}{R_2}.$$

Выражая скорости перемещения колес через базовую скорость v и неизвестную x, а радиусы через L, получаем

$$\frac{v+x}{v-x} = \frac{L+d}{L-d},$$

$$vL + xL - vd - xd = vL + vd - xL - xd,$$

$$2xL = 2vd,\ 2xL = 2vd,\ 2xL = 2vd,$$

$$v_2 = v - \frac{vd}{L} = v(1 - \frac{d}{L}).$$

Линейная скорость v пропорциональна угловой скорости колеса ω, которая, в свою очередь, пропорциональна мощности, подаваемой на моторы (в режиме торможения). Мы привели закон управления к стандартному виду, что позволяет задать управляющее воздействие на время поворота за угол. Таким образом, получаем расчет для управления моторами нашего робота.

```
u=50*8/L;
motor[motorB]=50+u;
motor[motorC]=50-u;
```

Когда расстояние до стены становится больше $2L$ (используем такой порог видимости), то есть открывается поворот за угол, управляющее воздействие начинает вычисляться по приведенным формулам (рис. 13).

В программе на RobotC добавим переменных, чтобы сделать ее более строгой и наглядной. При этом не требуется вводить переменную $L2$, поскольку, в отличие от Robolab, здесь имеется возможность писать в условиях формулы.

```
task main()
{
  float u, k1=2, k2=10;
  int v=50, d=8, Sold, L;
  Sold=L=SensorValue[S1];
  while(true)
  {
    if (SensorValue[S1]>L*2) {
      u=v*d/L;
      Sold=L*2;
    }
    else {
      u = k1*(SensorValue[S1]-L) +
        k2*(SensorValue[S1]-Sold);
      Sold=SensorValue[S1];
    }
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
}
}
```

Надо заметить, что мы слегка обманываем робота, подставляя ему ограниченные сверху значения. Такой алгоритм будет стабильно работать на расстоянии L от 25 до 125 см.

ФИЛЬТРАЦИЯ ДАННЫХ

Наконец, для окончательной стабилизации робота следует ввести защиту от помех. В силу особенностей работы ульт-

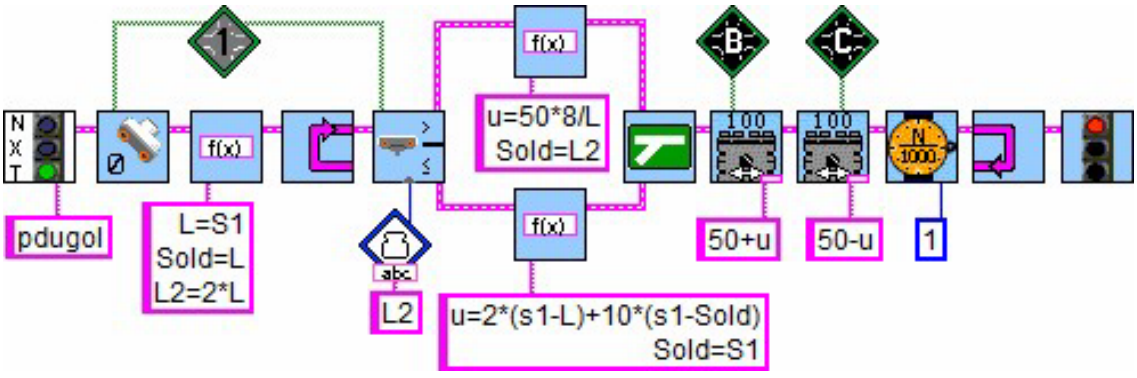


Рис. 13. Объезд предметов на заданном расстоянии по правилу правой руки

развукового датчика, сигнал время от времени не попадает на глазок-приемник, и в результате не всегда поступают адекватные значения, что может внести серьезные возмущения в наш алгоритм. Например, небольшая щель в стене для робота выглядит целым проемом. Поскольку требования к скорости пока что невысоки, можно несколько замедлить реакцию датчика на изменения расстояния, установив программный фильтр его значений. Простейшая реализация такого фильтра состоит в усреднении очередного показания и предыдущего отфильтрованного значения:

$$Snew = (Snew + S1)/2.$$

Среднее арифметическое несколько сглаживает резкие скачки показаний датчика. Во всех формулах вместо $S1$ будет использоваться отфильтрованное значение $Snew$. Однако и этого может не хватить при слишком резких изменениях. Тогда следует ввести весовые коэффициенты $c1$ и $c2$ для усредненной и новой величины.

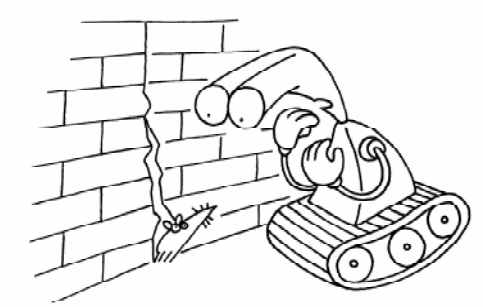
$$Snew = (c1*Snew + c2*S1)/(c1 + c2).$$

Раскрываем первые скобки

$$Snew = \frac{c1}{c1+c2} \cdot Snew + \frac{c2}{c1+c2} \cdot S1.$$

Легко заметить, что сумма полученных коэффициентов равна 1. Поэтому, заменив их, соответственно, на $1 - a$ и a, получим

$$Snew = (1 - a)*Snew + a*S1, \text{ где } 0 \le a \le 1.$$



...небольшая щель в стене для робота выглядит целым проемом.

Подбирая значение a, можно регулировать степень фильтрации. Для начала примем $a = 0.2$. Сравнение результатов фильтрации показаний датчика ультразвука в течение 1 с приведено на рис.д 14. Синий график показывает показания датчика, пурпурный – фильтр по среднему арифметическому, желтый – фильтр с параметром $a = 0{,}2$.

По графикам видно, что при малых значениях параметра фильтрация происходит эффективнее, и случайные скачки показаний практически нивелируются. С другой стороны, более-менее постоянные высокие значения достигаются в доли се-

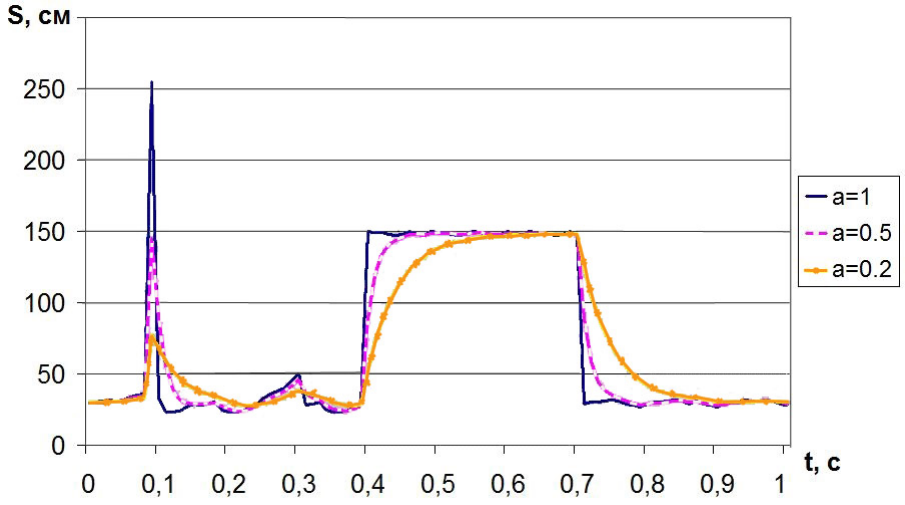


Рис. 14. Результаты фильтрации показаний датчика ультразвука с различными коэффициентами

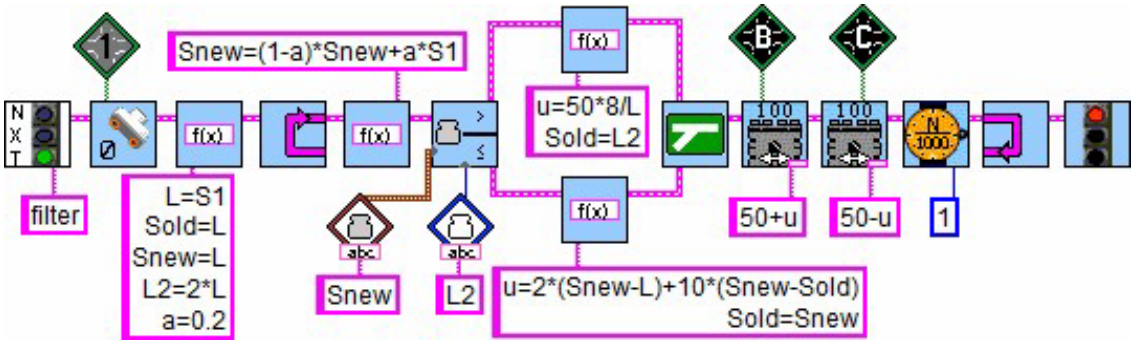


Рис. 15. Алгоритм движения вдоль стенки с фильтрацией данных

кунды, причем переход становится достаточно плавным.

Перепишем программы под новые отфильтрованные показания датчика. Поначалу результат может оказаться неожиданным, однако подбор коэффициентов все расставит по своим местам (рис. 15).

```
task main()
{
  float u, k1=2, k2=10, a=0.2, Snew;
   int v=50, d=8, Sold, L;
   Snew=Sold=L=SensorValue[S1];
   while(true)
   {
    Snew=(1-a)*Snew+a*SensorValue[S1];
     if (Snew>L*2) {
       u=v*d/L;
       Sold=L*2;
     }
     else {
      u = k1*(Snew-L) + k2*(Snew-Sold);
       Sold=Snew;
     }
     motor[motorB]=v+u;
     motor[motorC]=v-u;
     wait1Msec(1);
   }
}
```

Фильтрация данных становится особенно актуальной, если на их основе требуется принимать решение о дальнейших действиях в долгосрочной перспективе. Например, увидев проем, остановиться или повернуть назад. Достаточно одной помехи, чтобы робот остановился не в том месте. Поэтому фильтры, хоть и затормаживают реакцию робота, но делают ее более стабильной и предсказуемой.

Литература

1. *С.А. Филиппов.* Робототехника для детей и родителей. Под ред. *А.Л. Фрадкова.* СПб.: Наука, 2010.

2. *М.С. Ананьевский, Г.И. Болтунов, Ю.Е. Зайцев, А.С. Матвеев, А.Л. Фрадков, В.В. Шиегин.* Санкт-Петербургские олимпиады по кибернетике. Под ред. *А.Л. Фрадкова, М.С. Ананьевского.* СПб.: Наука, 2006.

3. Сайт подразделения Lego Education: http://www.lego.com/education/.

4. Среда трехмерного моделирования Lego Digital Designer: http://ldd.lego.com/.

5. Среда программирования RobotC: http://www.robotc.net/.

6. Сайт поддержки пользователей Lego Mindstorms, Robolab 2.9.4 и пр.: http://www.legoengineering.com/.

7. Сайт о роботах, робототехнике и микроконтроллерах: http://www.myrobot.ru/.

Филиппов Сергей Александрович,
учитель информатики
физико-математического лицея
№ 239, методист.