

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»



МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Программирование встроенных систем и микроконтроллеров»
для студентов направления 09.03.04 Программная инженерия
(направленность(профиль) «Инженерия сложных программных систем»)

Ставрополь

2026

Лабораторная работа № 1.

Введение в устройство стенда «Мехатроника»

Цель и содержание: знакомство со стендом «Мехатронные системы» и техникой безопасности при работе с ним.

Общие сведения

Стенд «Мехатронные системы» предназначен для организации учебного процесса в высших учебных заведениях по курсам «Электроприводы роботов», «Элементы электроники микропроцессорной техники», «Промышленная робототехника», «Управление роботами» и другим дисциплинам, рассматривающим проектирование, конструирование, управление и взаимодействие с мехатронными и робототехническими системами.

Помимо решения задач моделирования стенд «Мехатронные системы» может использоваться как универсальный программный комплекс для проведения различных исследований и обработки результатов.

Стенд «Мехатронные системы» представляет собой стационарное рабочее место и предназначен для отработки начальных навыков общения с роботизированными системами, а также для проверки и отработки вновь созданных алгоритмов управления перед опробованием их на сложных роботизированных системах.

Стенд состоит из следующих электронных и механических узлов:

Источник Gen-50-30 TDK Lambda служит для обеспечения стенда низковольтным питанием. Для корректной работы стенда необходимо

выставить напряжение питания $48\text{В} \pm 0,2\text{В}$ и отсечку по силе тока на уровне 5А .

Видеокамера «Axis» располагается над рабочим полем манипулятора и может служить для работы в составе комплекса ПО для автономной работы комплекса с предметами.

Манипулятор робота – роботизированная система имеет 4 силовых степени подвижности (2 в плече и 2 в локте) и 6 вспомогательных (поворот, захват и движение структурных групп).

Каждый силовой привод оснащается двумя датчиками положения. Один датчик абсолютного движения, разрядность которого 12 бит или 4096 точек на оборот. Второй – инкрементальный, установлен на валу электродвигателя, его точность 500 точек на оборот.

На кончиках структурных групп захвата манипулятора располагаются тактильные датчики чувствительностью 160 грамм. В корпусе захвата располагаются тактильные датчики расстояния с зоной работы 1-20 см. Манипулятор представлен на рисунке 1.1.

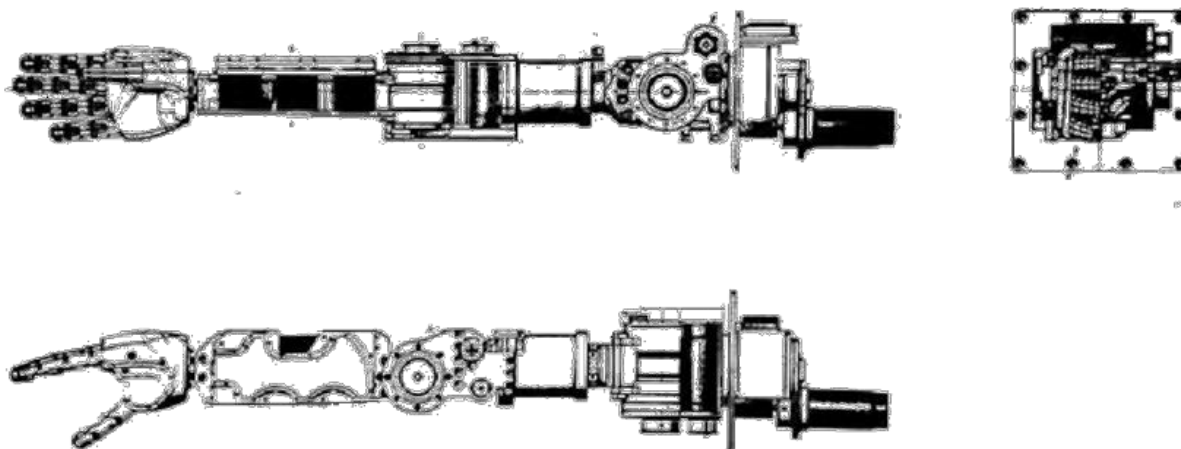


Рисунок 1.1 – Манипулятор «рука»

Кронштейн регулируемый с рабочим полем предназначен для подачи рабочего поля с находящимся на нем предметом манипулирования в необходимую зону работы манипулятора. Кронштейн регулируется не только в плоскости рабочей зоны, но и по высоте. Рабочее поле разлинено для облегчения позиционирования манипулятора при работе по видеокамере.

Порядок включения и работы со стендом

Рабочее место оператора стенда представлено на рисунке 1.2. В левой части расположены кнопки управления питанием стенда и электрический предохранитель для защиты системы от перепадов напряжения в сети, а также две розетки с напряжением сети 220 вольт. В правой, кнопки управления питанием материнской платы, экстренного отключения и две розетки 220 вольт (рис.1. 3)



Рисунок 1.2 – Рабочее место оператора стенда

Кнопки управления питанием маркированы цветами: зеленая для включения, красная для отключения стенда (рисунок 3). Включение стенда

подразумевает подачу напряжения на все вмонтированные в стол розетки и на блок низковольтного питания. Именно с включения питания стенда следует начинать работу с ним.

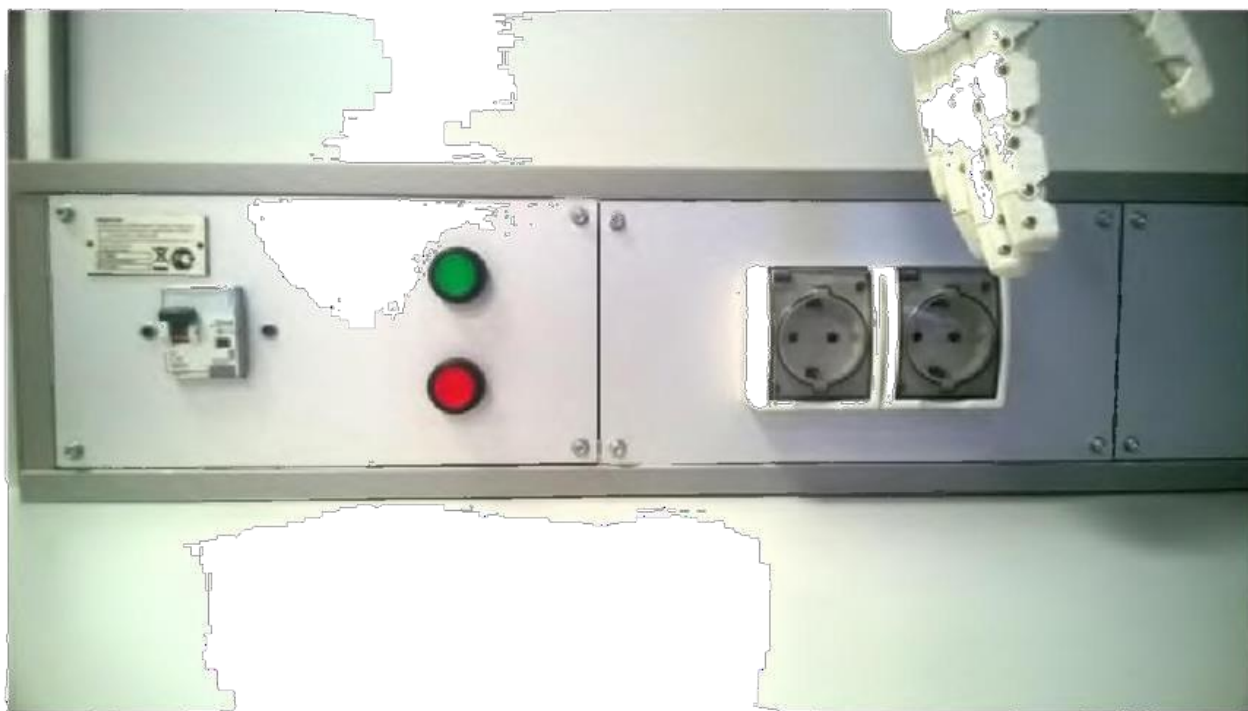


Рисунок 1.3 – Левая сторона стенда, кнопки управления питанием стенда.



Рисунок 1.4 – Блок низковольтного питания материнской платы

После подачи напряжения на блок низковольтного питания (рисунок 4) следует его включить, повернув ручку тумблера (крайний слева) в положение «1» и нажав на кнопку вывода напряжения.

Далее следует подать напряжение на материнскую плату стенда. Для этого необходимо нажать и удерживать кнопку включения питания (зеленая). В этот момент кнопки загораются зеленым и желтым цветами (кнопка включения и выключения соответственно). Когда кнопка выключения тухнет, можно отпустить кнопку включения, питание подано (рисунок 5).



Рисунок 1.5 – Правая сторона стенда, кнопки управления питанием материнской платы и экстренного отключения

Порядок отключения стенда выполняется в обратной последовательности.

Теперь можно включить программу управления стендом (рис. 1.6). Рабочая область программы представлена тремя зонами: зона управления двигателями, панелью запуска запрограммированных движений и панелью студией. Управление каждым мотором происходит поочередно. Перед

оператором представлен список из десяти моторов, которые нумеруются начиная с номера 16. На таблице 1 приведена таблица нумерации моторов.

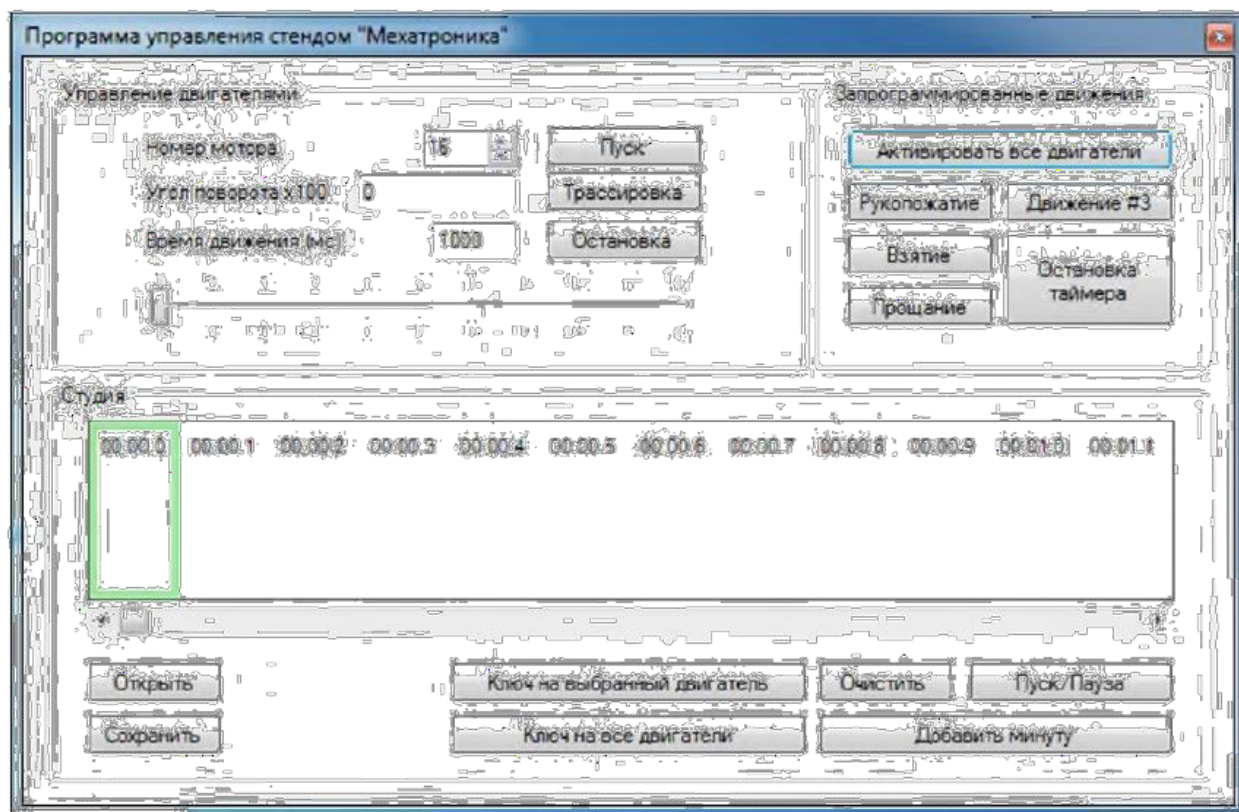


Рисунок 1.6 – Программа управления стендом

Каждый мотор имеет два режима работы – обычный и режим трассировки.

Для управления одним из моторов в обычном режиме необходимо: выбрать его номер, указать угол, на который он должен повернуться и время, за которое это должно произойти и нажать кнопку «Пуск» на панели управления двигателями.

*Внимание! Для большей точности углов используется формула «угол в градусах*100», т.е. чтобы повернуть мотор на угол 60 градусов, необходимо ввести число 6000. Таким образом, для угла 30,25 градусов вводится число 3025. А время указывается в миллисекундах.*

Таблица 1.1 – Список моторов, расположение и их диапазоны поворотов

№ мотора	Расположение	Диапазон поворота
16	Локоть	0 – 11000
17	Вращение предплечья	-4000 – 4200
18	Поворот плеча (влево, вправо)	-1000 – 8500
19	Поворот плеча (вперед, назад)	-2000 – 7500
20	Вращение кисти	-11000 – 9300
21	Большой палец	-100 – 7100
22	Указательный палец	-100 – 8300
23	Средний палец	-100 – 8500
24	Безымянный палец	-100 – 8500
25	Мизинец	-100 – 7900

Для управления в режиме трассировки необходимо выбрать номер двигателя, нажать на кнопку «Трассировка». После этих манипуляций появляется возможность управлять мотором в реальном времени посредством ползунка на панели управления двигателями.

Важно каждый раз после окончания работы с мотором отключать его с помощью кнопки «Остановка».

В блоке «Запрограммированные движения» расположены кнопки, запускающие движение руки робота по заранее заданному алгоритму. Они служат для демонстрации и тестирования работы стенда.

Блок «Студия» служит для создания микроалгоритмов движения руки – скриптов. С его помощью можно запрограммировать стенд выполнять определенные движения в определенные моменты времени. Таким образом, стенд становится подобием промышленных роботов.

Основная идея работы студии – ключевые позиции. Оператору необходимо перемещать руку в необходимую позицию и задавать ее как ключевую в определенный момент времени. В ключевые позиции записываются углы поворотов двигателей, поэтому оператору представлена возможность указывать ключевые позиции любого количества моторов в тот или иной момент на временной шкале. Это сделано для экономии времени выполнения и простоты работы. Поэтому если необходимо повернуть только локоть, нет необходимости ставить ключевые позиции всех моторов.

Студия представлена в виде временной шкалы и кнопок управления процессом создания скриптов:

- **Ключ на выбранный двигатель** – создает ключевую позицию на временной шкале для мотора, выбранного в блоке управления двигателями;
- **Ключ на все двигатели** – создает ключевую позицию на временной шкале для всех моторов;
- **Очистить** – удаляет выбранную ключевую позицию на временной шкале;
- **Добавить минуту** – добавляет к временной шкале одну минуту;
- **Пуск/Пауза** – запускает алгоритм движения с текущей выбранной позиции на временной шкале;
- **Открыть** – открыть заранее созданный скрипт из файла;
- **Сохранить** – сохранить текущий скрипт в файл;

В блоке «Студия» видно текущее положение на временной шкале (момент времени выделен зеленой окантовкой). Изменить текущее положение можно единожды нажав на любой другой момент времени на временной шкале. Также для перемещения по шкале необходимо воспользоваться скроллингом, расположенным ниже шкалы. Если щелкнуть

дважды на один из моментов времени, то рука примет положение, которое соответствует данному моменту в движении.

Методика и порядок выполнения работы

Включите стенд, запустите программу управления стендом. Убедитесь, что рука находится в начальном положении. Чтобы это сделать, необходимо поочередно повернуть все двигатели в положение 0 градусов с помощью блока управления двигателями.

Когда рука окажется в начальном положении, выделите самый первый момент времени на временной шкале и нажмите на кнопку «Ключ на все двигатели». Таким образом, с этого положения и будет начинаться алгоритм движения руки. Далее выделите момент «00:02,0» на временной шкале (он будет соответствовать 2 секундам с начала работы) и с помощью блока управления двигателями установить руку в позицию, показанную на рисунке 1.8. Теперь примените ключи для всех двигателей, задействованных в изменении положения.



Рисунок 1.8 – Вторая ключевая позиция руки

Те же манипуляции проделайте и для третьей ключевой позиции (рис. 1.9), заблаговременно выделив момент времени «00:04.0».

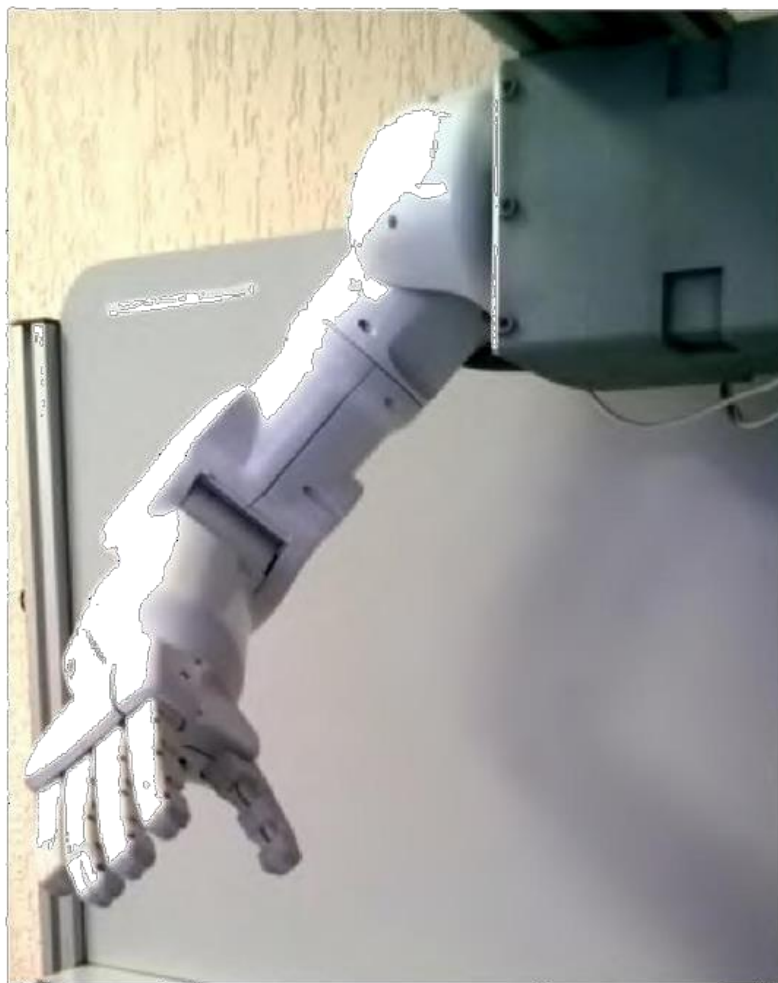


Рисунок 1.9 – Третья ключевая позиция руки

После данных манипуляций верните курсор на временной шкале в самый первый момент времени (00:00.0) и нажмите кнопку «Пуск/Пауза». Проверьте правильность выполнения заданного скрипта и сохраните его в файл.

Варианты занятий

Создать скрипт движений руки в соответствии с вариантом задания.

Таблица 1.2 – Варианты заданий

Номер вар-та	Условия задачи
1	«Прощание» (махать рукой)
2	«Рукопожатие» (необходимо выделить время для человека между протягиванием руки и рукопожатием)
3	Взятие теннисного мяча с определенной точки на расчерченной площадке кронштейна
4	Движение пальцами, как при сжимании и разжимании эспандера
5	Движение всей рукой, как при упражнении с гантелями
6	Движение «прямой удар рукой»
7	Движение «апперкот»
8	Движение, при котором рука положит заблаговременно взятый мяч на площадку кронштейна

Содержание отчета, его форма и правила оформления

Структура отчета по лабораторной работе:

1. Название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.
4. Таблица положений моторов по временной шкале.

Вопросы для защиты работы

1. Что представляет собой лабораторный стенд «Мехатроника»?
2. Из каких электронных и механических узлов состоит стенд?
3. Какой порядок включения и отключения стенда?
4. Опишите принцип работы с моторами руки.
5. Опишите принцип работы со студией.

Лабораторная работа 2.

Простейшая программа управления стендом «Мехатроника»

Цель и содержание: изучить основы программирования движений манипулятора руки стенда «Мехатроника».

Общие сведения

Высокоуровневое программирование движений манипулятора производится на языке C# при использовании библиотеки функций «AR-600».

Функции подразделяются на несколько категорий:

- Управление питанием электронных узлов стенда;
- Контроль питания электронных узлов стенда;
- Управление и контроль моторов стенда;
- Контроль сенсоров руки;

Питание стенда осуществляется многоканальной схемой, каждый канал которой питает свой агрегатный узел. Всего каналов семь:

- 5В – питание контрольной платы;
- 6В (первый канал) – питание сервоприводов пальцев кисти;
- 6В (второй канал) – питание IP-камеры, установленной на стенде;
- 8В (первый канал) – питание сенсоров руки;
- 8В (второй канал) – питание плат управления сервоприводов;
- 12В – питание бортовой материнской платы робота;
- 48В – питание всех моторов руки.

Список функции управления питанием:

void ON_6V1() – включение первого канала питания 6В;

void ON_6V2() – включение второго канала питания 6В;

void ON_8V1() – включение первого канала питания 8В;

void ON_8V2() – включение второго канала питания 8В;

void ON_12V() – включение канала питания 12В;

void ON_48V() – включение канала питания 48В;

void OFF_6V1() – выключение первого канала питания 6В;

void OFF_6V2() – выключение второго канала питания 6В;

void OFF_8V1() – выключение первого канала питания 8В;

void OFF_8V2() – выключение второго канала питания 8В;

void OFF_12V() – выключение канала питания 12В;

void OFF_48V() – выключение канала питания 48В;

void POWER_OFF() –отключение питания 5В;

Важно знать, что для безопасного включения и отключения стенда необходимо соблюдать строгий порядок включения каналов питания.

Для включения питания существует следующий порядок: 12В, 48В, 8В (1 канал), 8В (2 канал), 6В (1 канал), 6В (2 канал). А для отключения: 48В, 8В (1 канал), 8В (2 канал), 6 (1 канал), 6 (2 канал), 12В.

Список функции контроля питания:

int U5V – напряжение на канале 5В;

int U6V1 – напряжение на первом канале 6В;

int U6V2 – напряжение на втором канале 6В;

int U8V1 – напряжение на первом канале 8В;

int U8V2 – напряжение на втором канале 8В;

int U12V – напряжение на канале 12В;

int U48V – напряжение на канале 48В;

int I6V1 – потребляемый ток на первом канале 6В;

int I6V2 – потребляемый ток на втором канале 6В;

int I8V1 – потребляемый ток на первом канале 8В;

int I8V2 – потребляемый ток на втором канале 8В;

int I12V – потребляемый ток на канале 12В;

int I48V – потребляемый ток на канале 48В;

Функции контроля питания возвращают значения в формате $x*100$. Таким образом напряжение 47,95В будет возвращено в виде 4795В.

Внимание! Крайне желательно использовать в программе функции контроля потребляемого тока. Его внезапное повышения на одном или нескольких каналах питания может свидетельствовать о коротком замыкании в электронной цепи. В этом случае следует незамедлительно аварийно отключить стенд и обратиться к преподавателю.

Список функций управление и контроля моторов стенда:

`void MOT_CMD(int NumberOfMotor, int NumberOfCommand)` – отправляет команду по номеру `NumberOfCommand` мотору под номером `NumberOfMotor`;

`void MOT_TIME(int NumberOfMotor, int Value)` – отправляет значение времени `Value` мотору `NumberOfMotor`, за которое он должен повернуться;

`void MOT_TPOS(int NumberOfMotor, int Value)` – отправляет значение угла `Value` мотору `NumberOfMotor`, на который он должен повернуться;

`void MOT_PTB(int NumberOfMotor, int Value)`

`int GET_MOT_MINPOS(int NumberOfMotor)` – возвращает минимальное положение мотора `NumberOfMotor`;

`int GET_MOT_POS(int NumberOfMotor)` – возвращает текущее положение мотора `NumberOfMotor`;

`int GET_MOT_MAXPOS(int NumberOfMotor)` – возвращает максимальное положение мотора `NumberOfMotor`;

Одной из главных функций библиотеки является `MOT_CMD`. Она управляет режимами моторов стенда. Список режимов и их номера приведены в таблице 2.1.

Действия, помеченные знаком звездочки заблокированы для оператора на данном этапе обучения. Для более полной информации обратитесь к преподавателю.

Таблица 2.1 – Команды управления моторами

Номер команды	Действие
1	Сохранение низкоуровневых настроек мотора*
2	Остановка
3	Снятие муфты тормозов
4	Снятие сброса мотора
5	Сброс мотора
6	Режим трассировки
7	Пуск
8	Центровка*
11	Установка реверсивной шкалы позиций*

Действия, помеченные знаком звездочки заблокированы для оператора на данном этапе обучения. Для более полной информации обратитесь к преподавателю.

Методика и порядок выполнения работы

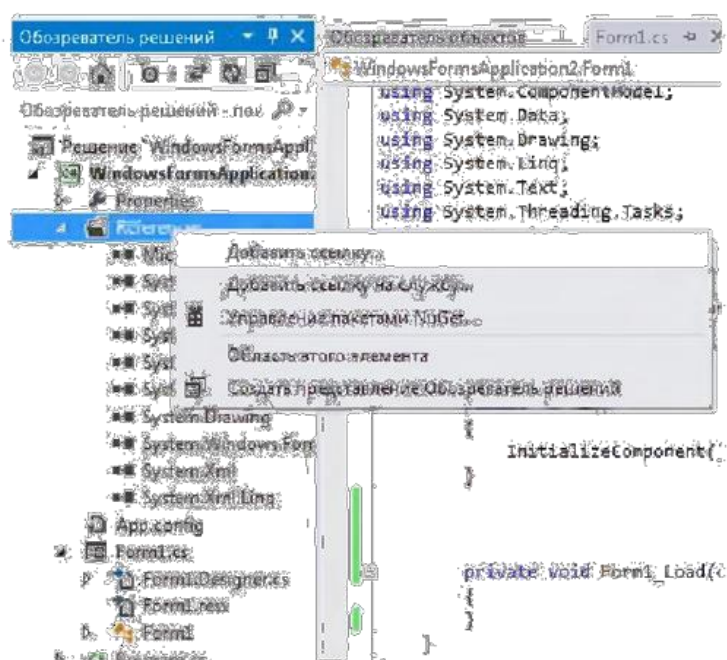


Рисунок 2.1 – Команда добавления внешней библиотеки

Создайте новый проект в Visual Studio 2012. Подключите библиотеку функций AR600.dll с помощью команды «Добавить ссылку» (рис. 2.1).

После добавления в проект библиотеки, программисту становится доступно пространство имен AR600_NS, которое необходимо добавить в число используемых в проекте:

```
using AR600_NS;
```

Теперь можно объявить объект *robot* класса *_AR600* и создать его в событии Load главной формы. Так же сюда нужно добавить команды включения питания:

```
private void Form1_Load(object sender, EventArgs e)
{
    robot = new _AR600();
    robot.START();
    robot.ON_12V(); Thread.Sleep(100);
    robot.ON_48V(); Thread.Sleep(100);
    robot.ON_8V1(); Thread.Sleep(100);
    robot.ON_8V2(); Thread.Sleep(100);
    robot.ON_6V1(); Thread.Sleep(100);
    robot.ON_6V2(); Thread.Sleep(100);
}
```

Сразу следует позаботиться о правильном завершении программы, работа которой всегда должна заканчиваться отключением всех каналов питания. Для этого пропишите в событие FormClosing главной формы следующие строки:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    robot.OFF_48V(); Thread.Sleep(100);
    robot.OFF_8V1(); Thread.Sleep(100);
    robot.OFF_8V2(); Thread.Sleep(100);
    robot.OFF_6V1(); Thread.Sleep(100);
    robot.OFF_6V2(); Thread.Sleep(100);
    robot.OFF_12V(); Thread.Sleep(100);
}
```

После этого можно приступить к непосредственному управлению двигателями. Чтобы повернуть один из двигателей стенда, программа поочередно должна отправлять на контрольную плату данные о времени движения, требуемого угла и, соответственно, команду на пуск. Следует

помнить, что время указывается в миллисекундах, а угол в градусах*100. Так же между командами необходимо производить паузы в 50-100 миллисекунд во избежание ошибок работы.

Для начала поместите на форму две кнопки «Пуск» и «Стоп». Первая должна будет запускать движение, вторая останавливать стенд в случае непредвиденных ситуаций.

В событие Click кнопки «Пуск» напишите следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
    robot.MOT_TIME(16, 100); Thread.Sleep(100);
    robot.MOT_TPOS(16, 8000); Thread.Sleep(100);
    robot.MOT_CMD(16, 7); Thread.Sleep(100);
}
```

Он повернет локтевой сустав на угол 80 градусов за 1 секунда с начальной позиции. Так же обратите внимание на порядок отправления команд на контрольную плату и паузы между командами.

И для обеспечения безопасности работы впишите в событие Click кнопки «Стоп» код остановки 16-го двигателя:

```
private void button2_Click(object sender, EventArgs e)
{
    robot.MOT_CMD(16, 2); Thread.Sleep(100);
}
```

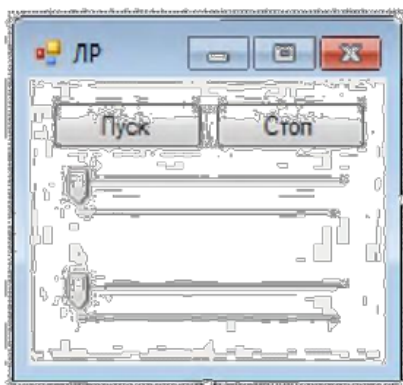


Рисунок 2.2 – Вид программы

Запустите программу. Если угол локтя на момент пуска программы уже был 80 градусов, то при нажатии на кнопку «Старт» ничего не произойдет.

Далее для более интерактивной работы добавьте на форму два объекта класса TrackBar установите их диапазоны (от 0 до 9000 и от 100 до 1000 для первого и второго соответственно) (рис. 1.2).

Далее перейдите к событию Click кнопки «Старт» и измените его код следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    robot.MOT_TIME(16, trackBar2.Value); Thread.Sleep(100);
    robot.MOT_TPOS(16, trackBar1.Value); Thread.Sleep(100);
    robot.MOT_CWD(16, 7); Thread.Sleep(100);
}
```

Запустите программу и попробуйте поворачивать локоть на разные углы и за разные отрезки времени. Так же перед каждым закрытие программы следует нажимать кнопку «Стоп», чтобы избежать сбоя системы (все двигатели должны оставаться в режиме остановки).

Варианты занятий

Напишите на языке C# программу, которая будет управлять рукой по алгоритму в соответствии с вариантом задания.

Таблица 2.2 – Варианты заданий

Номер варианта	Условия задачи
1	«Прощание» (махать рукой)
2	«Рукопожатие» (необходимо выделить время для человека между протягиванием руки и рукопожатием)
3	Взятие теннисного мяча с определенной точки на расчерченной площадке кронштейна
4	Движение пальцами, как при сжимании и разжимании эспандера
5	Движение всей рукой, как при упражнении с гантелями
6	Движение «прямой удар рукой»

Продолжение табл. 2.2

Номер варианта	Условия задачи
7	Движение «апперкот»
8	Движение, при котором рука положит заблаговременно взятый мяч на площадку кронштейна

Содержание отчета, его форма и правила оформления

Структура отчета по лабораторной работе:

1. Название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.
4. Программа, написанная по индивидуальному заданию.

Вопросы для защиты работы

1. Перечислите каналы питания стенда и их назначение.
2. Перечислите функции управления моторами.
3. Порядок включения и отключения каналов питания.
4. Как задать движение мотору?
5. Какой порядок включения и отключения стенда?

Лабораторная работа 3. Питание и сенсоры руки станда

Цель и содержание: контроль питания и сенсоры руки станда «Мехатроника».

Общие сведения

Рука станда «Мехатроника» оснащена 4 инфракрасными сенсорами-длинномерами, пятью сенсорами-кнопками на пальцах и сенсором, измеряющим давление на кисть в трех измерениях. Сенсоры-длинномеры определяют расстояние от кисти до окружающих объектов. Они работают в диапазоне от 2 до 20 см.

Снятие показаний с сенсоров осуществляется с помощью следующих свойств и атрибутов:

- `int RH_DIST0` – нулевой сенсор-длинномер;
- `int RH_DIST1` – первый сенсор-длинномер;
- `int RH_DIST2` – второй сенсор-длинномер;
- `int RH_DIST3` – третий сенсор-длинномер;
- `int RH_FX` – сенсор давления по оси X;
- `int RH_FY` – сенсор давления по оси Y;
- `int RH_FZ` – сенсор давления по оси Z;
- `int[] RH_KEYS` – массив показаний сенсоров-кнопок.

Все выше перечисленные методы возвращают значения в условных единица.

Методика и порядок выполнения работы

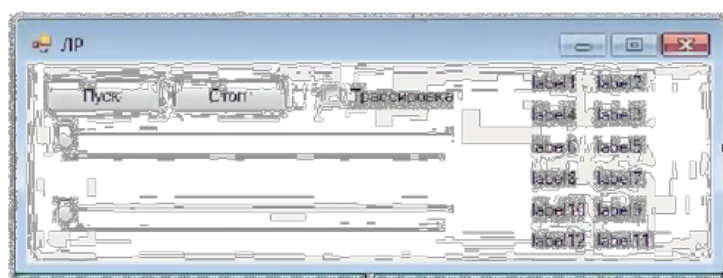


Рисунок 3.1 – Расположение объектов класса Label на форме

Откройте свой проект предыдущей лабораторной работы. Расположите на форме 12 объектов класса Label, как показано на рисунке 3.1. И подпишите их в соответствии с рисунком 3.2.

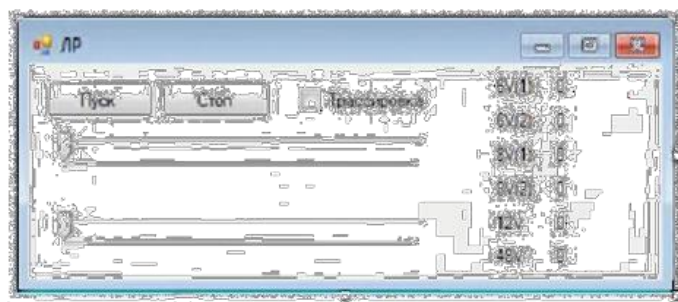


Рисунок 3.2 — Подпись объектов класса Label

Далее добавьте на форму объект класса Timer, измените интервал тиков в значение 100 миллисекунд и создайте для него обработчик события Tick, как показано ниже:

```
private void timer1_Tick(object sender, EventArgs e)
{
    label2.Text = robot.USV1.ToString();
    label3.Text = robot.USV2.ToString();
    label5.Text = robot.USV1.ToString();
    label7.Text = robot.USV2.ToString();
    label9.Text = robot.D12V.ToString();
    label11.Text = robot.D48V.ToString();
}
```

Теперь измените обработчик события Load главной формы следующим образом:

```
private void Form1_Load(object sender, EventArgs e)
{
    robot = new _Robot();
    robot.START();
    robot.ON_12V(); Thread.Sleep(100);
    robot.ON_48V(); Thread.Sleep(100);
    robot.ON_5V1(); Thread.Sleep(100);
    robot.ON_5V2(); Thread.Sleep(100);
    robot.ON_6V1(); Thread.Sleep(100);
    robot.ON_6V2(); Thread.Sleep(100);

    timer1.Enabled = true;
}
```

А обработчик события Closing следующим:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    timer1.Enabled = false;

    robot.OFF_48V(); Thread.Sleep(100);
    robot.OFF_8V1(); Thread.Sleep(100);
    robot.OFF_8V2(); Thread.Sleep(100);
    robot.OFF_5V1(); Thread.Sleep(100);
    robot.OFF_5V2(); Thread.Sleep(100);
    robot.OFF_12V(); Thread.Sleep(100);
}
```

Запустите и протестируйте программу.

Снятие показаний сенсоров

Добавьте на форму еще 16 объектов класса Label, расположите и подпишите, как показано на рисунке 3.3.

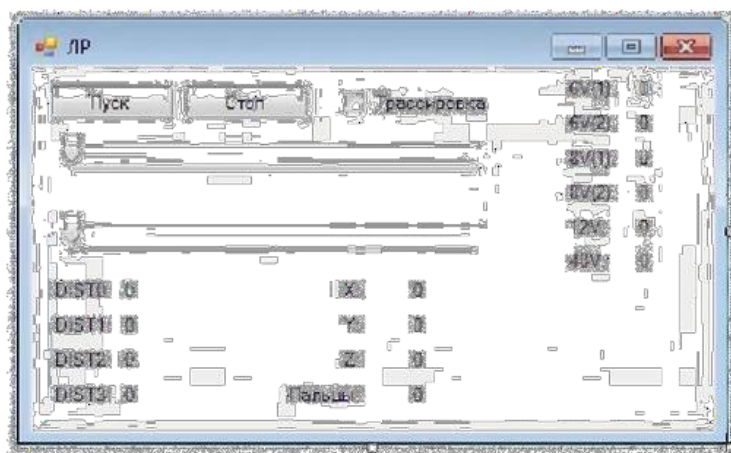


Рисунок 3.3 – Расположение объектов для индикации показаний сенсоров

Далее измените обработчик события Tick объекта timer1 в соответствии с приведенным ниже кодом:

```

private void timer1_Tick(object sender, EventArgs e)
{
    label2.Text = (robot.U5V1/1000.0f).ToString();
    label3.Text = (robot.U5V2/1000.0f).ToString();
    label5.Text = (robot.U8V1/1000.0f).ToString();
    label7.Text = (robot.U8V2/1000.0f).ToString();
    label9.Text = (robot.U12V/1000.0f).ToString();
    label11.Text = (robot.U48V / 100.0f).ToString();

    label19.Text = robot.RH_DIST0.ToString();
    label17.Text = robot.RH_DIST1.ToString();
    label15.Text = robot.RH_DIST2.ToString();
    label13.Text = robot.RH_DIST3.ToString();

    label27.Text = robot.RH_FX.ToString();
    label25.Text = robot.RH_FY.ToString();
    label23.Text = robot.RH_FZ.ToString();

    string resFingers = "";
    for (int i = 0; i < 5; i++)
        resFingers += robot.RH_KEYS[i].ToString();
    label21.Text = resFingers;
}

```

Запустите программу и проверьте правильность ее работы. При необходимости скорректируйте команды присвоения.

Варианты занятий

Создать функции движений руки в соответствии с вариантом задания.

Таблица 1.1 – Варианты заданий

Номер варианта	Условия задачи
1	Реакция пальцев (каждый палец должен одергиваться в момент нажатия на кнопку сенсор, расположенную на нем)
2	Рукопожатие (задайте программу рукопожатия, которая реагирует на показания сенсора)
3	Произвольные циклические движения руки по оси X (с остановками в моменты обнаружения препятствия сенсором)
4	Произвольные циклические движения руки по оси Y (с остановками в моменты обнаружения препятствия сенсором)
5	Произвольные циклические движения руки по оси X (с остановками в моменты повышения потребления тока)
6	Произвольные циклические движения руки по оси Y (с остановками в моменты повышения потребления тока)

Номер варианта	Условия задачи
7	Произвольные циклические движения руки по оси Z (с остановками в моменты повышения потребления тока)
8	Алгоритм, при котором рука хватается мяч, вложенный ей в ладонь, и кладет его на стол

Содержание отчета, его форма и правила оформления

Структура отчета по лабораторной работе:

1. Название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.
4. Выписка зависимости уровней тока на каналах от работы двигателей.
5. Зависимость показаний сенсоров-длинномеров, от расстояний на которые они реагируют;
6. Блок схема программы.

Вопросы для защиты работы

1. Какие типы сенсоров установлены на лабораторный стенд «Мехатроника»?
2. В каком формате возвращаются показания напряжения и тока на каналах питания?
3. Какие существуют каналы питания и за что они отвечают?
4. Почему важно следить за показаниями электропитания на каналах? Какие последствия этим можно избежать?

Лабораторная работа 4. Режим трассировки

Цель и содержание: знакомство с режимом трассировки моторов руки станда «Мехатроника».

Общие сведения

Одним из режимов работы моторов станда является трассировка. В этом режиме, в отличие от режима, рассмотренного в предыдущей лабораторной работе возможно управление мотором в реальном времени. То есть, указав угол поворота мотора, он изменит свою позицию с максимально возможной скоростью без дополнительных функций управления временем и командами запуска. Чтобы изменить угол поворота двигателя в этом режиме, достаточно указать требуемый угол для выбранного мотора.

Внимание! Разница между текущим углом поворота и указываемым не должна превышать 5-7 градусов. В противном случае может произойти слишком быстрое движение двигателей, что приведет к повреждению конструкции станда.

Режим трассировки используется во всех роботизированных системах для решения задач адаптивных алгоритмов. Например, сохранение равновесия двуногим роботом, стоя на одной ноге.

Для работы в режиме трассировки моторов используется метод MOT_PTB(int NumMot, int Pos). Он указывает мотору позицию, на которую необходимо переместится в текущий момент времени. Принимает две атрибуты: NumMot – номер мотора, Pos – нужная позиция.

Методика и порядок выполнения работы

Откройте свой проект предыдущей лабораторной работы. Добавьте на форму объект класса CheckBox и подпишите его «Трассировка» (рис. 4.1).

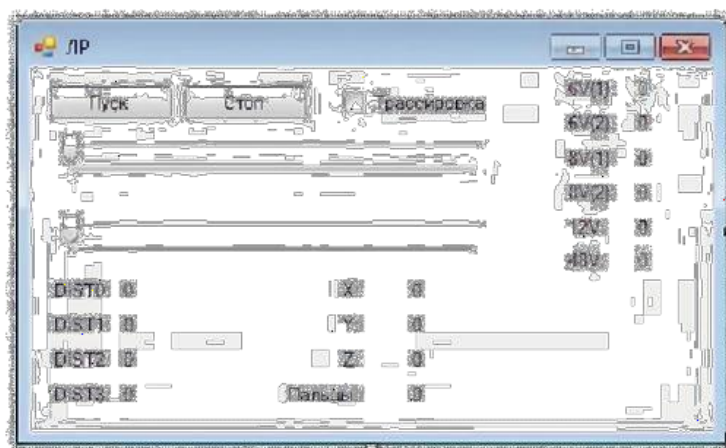


Рисунок 4.1 – Флажок «Трассировка» на главной форме

Далее следует объявить переменную `trace` типа `bool` и сразу присвоить ей значение `false`:

```
bool trace = false;
```

Она пригодится для переключения между обычным режимом работы и режимам трассировки. Определите обработчик события `CheckedChanged` для объекта `checkBox1`:

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        trackBar1.Value = robot.GET_MOT_POS(16);
        robot.MOT_TIME(16, 10); Thread.Sleep(100);
        robot.MOT_IPOS(16, robot.GET_MOT_POS(16)); Thread.Sleep(100);
        robot.MOT_CMD(16, 7);
        trace = true;
    }
    else
    {
        robot.MOT_CMD(16, 2); Thread.Sleep(100);
        trace = false;
    }

    button1.Enabled = !trace;
    button2.Enabled = !trace;
    trackBar2.Enabled = !trace;
}
```

Далее создайте обработчик события `Scroll` для объекта `trackBar1`:

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    if (trace)
        robot.MOT_PT8(16, trackBar1.Value);
}
```

Теперь в случае установки флажка «Трассировка», будет активирован 16 двигатель руки (локтевой сустав) и появится возможность управлять им напрямую. Запустите приложение и протестируйте правильность работы нововведений.

Варианты занятий

Создать функции движений руки в соответствии с вариантом задания.

Таблица 4.1 – Варианты заданий

Номер вар-та	Условия задачи
1	Реакция пальцев (каждый палец должен одергиваться в момент нажатия на кнопку сенсор, расположенную на нем)
2	Следование по оси X (рука должна двигаться в сторону давления на кисть)
3	Следование по оси Y (рука должна двигаться в сторону давления на кисть)
4	Следование по оси Z (рука должна изгибаться в соответствии давления на кисть)
5	Отстранение по оси X (рука должна двигаться в сторону от приближения предметов к сенсорам-длинномерам)
6	Отстранение по оси Y (рука должна двигаться в сторону от приближения предметов к сенсорам-длинномерам)
7	Отстранение по оси Z (рука должна изгибаться в сторону от приближения предметов к сенсорам-длинномерам)
8	Хватание предметов, вложенных в руку

Содержание отчета, его форма и правила оформления

Структура отчета по лабораторной работе:

1. Название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.

4. Блок схема программы
5. Вывод

Вопросы для защиты работы

1. Принцип работы режима трассировки.
2. Для каких целей может использоваться режим трассировки моторов?
3. Какой порядок активации режима трассировки?
4. Назовите методы, используемые для управления в режиме трассировки.
5. Каким образом происходит деактивация режима трассировки?

Лабораторная работа 5. Групповое управление моторами

Цель и содержание: изучить групповое управление моторами руки стенда «Мехатроника».

Общие сведения

Одновременное управление группой моторов не предусмотрено конструкцией и штатным программным обеспечением, поэтому для этой задачи была разработаны отдельные функции, позволяющие обойти это ограничение.

Управление группой моторов осуществляется следующими методами:

- GROUP_PTB(int[] data) – групповая трассировка моторов;
- GROUP_STOP(int[] data) – останавливает группу моторов;
- GROUP_RELAX(int[] data) – снимает тормозные муфты с моторов;

- GROUP_TIME(int[] data) — задает время движения для группы моторов;
- GROUP_TPOS(int[] data) — задает положения, на которые должна повернуться группа моторов;
- GROUP_GO(int[] data) — запускает движение группы моторов в обычном режиме;
- group_setVal(int[] values) — формирует массивы команд для всех методов группового управления.

Методика и порядок выполнения работы

Откройте свой проект предыдущей лабораторной работы. Добавьте на форму две кнопки, расположите и подпишите их, как показано на рисунке 5.1.

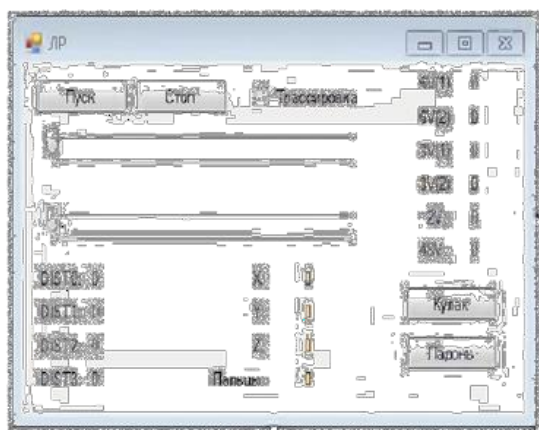


Рисунок 5.1 – Две новых кнопки на форме приложения

Создайте для них обработчики события Click. Для первой обработчик будет выглядеть следующим образом:

```

private void button3_Click(object sender, EventArgs e)
{
    int[] poses = { 21, 7000, 22, 7000, 23, 7000, 24, 7000, 25, 7000 };
    int[] times = { 21, 100, 22, 100, 23, 100, 24, 100, 25, 100 };
    int[] go = { 21, 0, 22, 0, 23, 0, 24, 0, 25, 0 };

    robot.GROUP_TIME(robot.group_setVal(times));
    robot.GROUP_TPOS(robot.group_setVal(poses));
    robot.GROUP_GO(robot.group_setVal(go));
}

```

Также создайте обработчик события Click для второй кнопки:

```

private void button4_Click(object sender, EventArgs e)
{
    int[] poses = { 21, 0, 22, 0, 23, 0, 24, 0, 25, 0 };
    int[] times = { 21, 100, 22, 100, 23, 100, 24, 100, 25, 100 };
    int[] go = { 21, 0, 22, 0, 23, 0, 24, 0, 25, 0 };

    robot.GROUP_TIME(robot.group_setVal(times));
    robot.GROUP_TPOS(robot.group_setVal(poses));
    robot.GROUP_GO(robot.group_setVal(go));
}

```

Из написанного кода видно, что все методы принимают переработанные массивы с помощью метода group_setVal(). А group_setVal() принимает в свою очередь массивы формата { номер мотора, позиция/время_движения/0, номер мотора, позиция/время_движения/0, ... }.

Внимание! Крайне важно, чтобы номера двигателей для всех трех массивов управления совпадали, как показано в коде выше. В противном случае, поведение стенда может быть непредсказуемо.

Варианты занятий

Создать функции движений руки в соответствии с вариантом задания,

Таблица 5.1 – Варианты заданий

Номер варианта	Условия задачи
1	Постановка позиции руки, чтобы получился символ "OK"
2	Постановка позиции руки, чтобы получился символ "Victoria"
3	Взятие теннисного мяча с определенной точки на расчерченной площадке кронштына

Номер варианта	Условия задачи
4	Движение пальцами, как при сжимании и разжимании эспандера
5	Движение всей рукой, как при упражнении с гантелями
6	Движение «прямой удар рукой»
7	Движение «апперкот»
8	Рука должна показывать пальцами число введенное оператором

Содержание отчета, его форма и правила оформления

Структура отчета по лабораторной работе:

1. Название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.
4. Выписка зависимости уровней тока на каналах от работы двигателей.
5. Блок схема программы

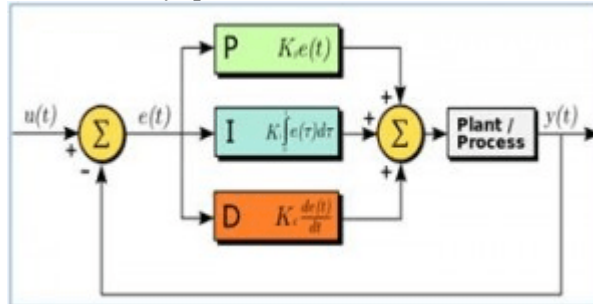
Вопросы для защиты работы

1. Перечислите методы группового управления моторами.
2. По какому принципу вводятся команды управления?
3. В чем разница команд STOP и RELAX?
4. Какие проблемы могут последовать при несоблюдении правил ввода команд?

Лабораторная работа 6

PID контроллер для выполнения множественных задач

ПИД-регуляторы находятся в широком диапазоне применений для управления промышленными процессами. Приблизительно 95% операций с замкнутым контуром в промышленной автоматизации используют ПИД-регуляторы. PID обозначает Пропорционально-интегральная-дифференциальная составляющая. Эти три контроллера объединены таким образом, что он создает управляющий сигнал.



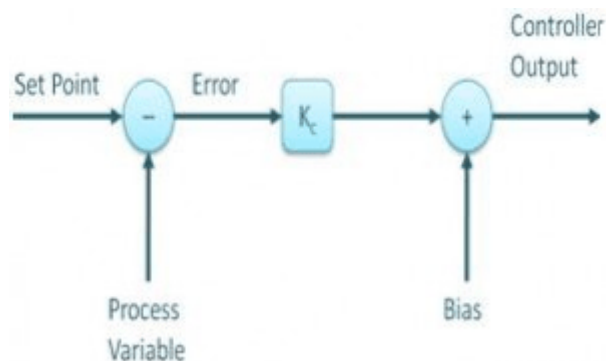
В качестве контроллера обратной связи он обеспечивает выход управления на желаемых уровнях. ПИД-регулирование осуществлялось с помощью аналоговых электронных компонентов, перед изобретением микропроцессоров. Но сегодня все ПИД-контроллеры обрабатываются микропроцессорами. Программируемые логические контроллеры также имеют встроенные настройки ПИД-регулятора. Благодаря гибкости и надежности ПИД-регуляторов, они традиционно используются в системах управления технологическим процессом.

Работа ПИД-регулятора

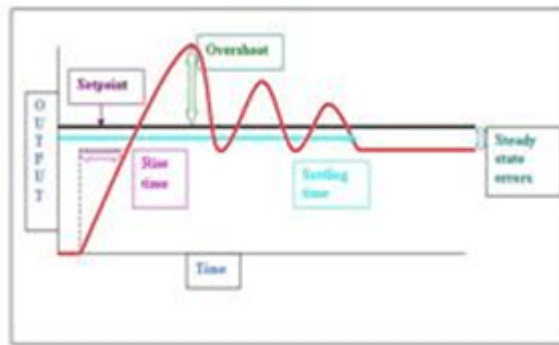
При использовании недорогого простого контроллера возможны только два состояния управления, например, полностью ВКЛ или полностью ВЫКЛ. Он используется для настроек с ограничением контроля, в котором эти два состояния управления достаточно для целей управления. Однако характер этого контроля ограничивает его использование и, следовательно, заменяется ПИД-контроллерами.

ПИД-регулятор поддерживает выход таким образом, что между переменной процесса и заданной точкой / желаемым выходом с помощью операций замкнутого контура имеется нулевая ошибка. ПИД использует три основных поведения управления, которые объясняются ниже.

П-контроллер:

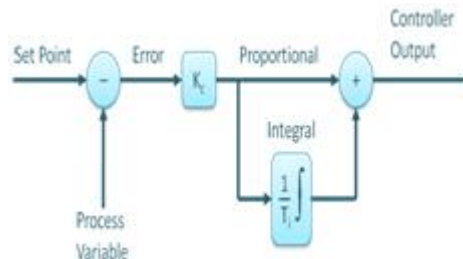


Пропорциональный или П-регулятор дает выход, который пропорционален текущей ошибке $e(t)$. Он сравнивает желаемую или заданную точку с фактическим значением или значением процесса обратной связи. Полученная ошибка умножается на пропорциональную константу, чтобы получить выход. Если значение ошибки равно нулю, то выход этого контроллера равен нулю.



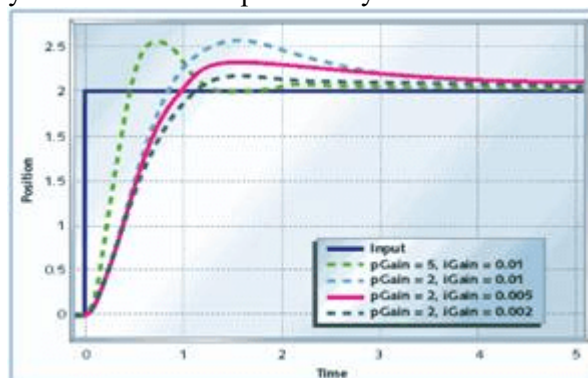
Этот контроллер требует смещения или ручной сброс при использовании отдельно. Это происходит потому, что он никогда не достигает состояния устойчивого состояния. Он обеспечивает стабильную работу, но всегда поддерживает постоянную ошибку. Скорость реакции возрастает при увеличении пропорциональной константы K_p .

И-контроллер



Из-за ограничения П-контроллера, где всегда существует смещение между переменной процесса и заданным значением, необходим И-контроллер, который обеспечивает необходимые действия для устранения ошибки установившегося состояния. Он интегрирует ошибку в течение периода времени, пока значение ошибки не достигнет нуля. Он содержит значение для конечного устройства управления, при котором ошибка становится равной нулю.

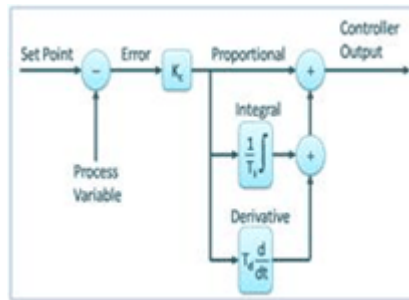
Интегральное управление уменьшает его выход, когда происходит отрицательная ошибка. Он ограничивает скорость реакции и влияет на стабильность системы. Скорость реакции увеличивается за счет уменьшения интегрального усиления K_i .



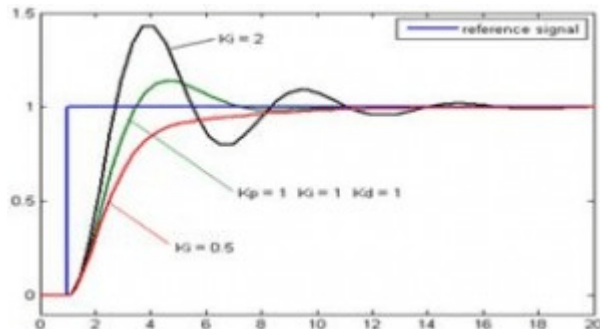
На приведенном выше рисунке, когда коэффициент усиления И-контроллера уменьшается, ошибка установившегося режима также продолжает уменьшаться. В большинстве случаев контроллер ПИ используется, в частности, когда требуется высокая скорость ответа.

При использовании ПИ-регулятора выход И-контроллера ограничен некоторым диапазоном для преодоления интегральных условий, когда интегральный выход растет даже при нулевом состоянии ошибки из-за нелинейности на установке.

Д-контроллер



И-контроллер не может предсказать будущее поведение ошибки. Поэтому он реагирует нормально после изменения заданного значения. Д-контроллер преодолевает эту проблему, ожидая будущего поведения ошибки. Его выход зависит от скорости изменения погрешности за время, умноженное на постоянную производной. Это дает начало запуска для выхода, тем самым увеличивая системный отклик.



На приведенном выше рисунке ответ контроллера Д больше, по сравнению с контроллером ПИ, а также время установления выходного сигнала уменьшается. Это улучшает стабильность системы за счет компенсации фазового запаздывания, вызванного И-контроллером. Увеличение производного усиления увеличивает скорость реакции..

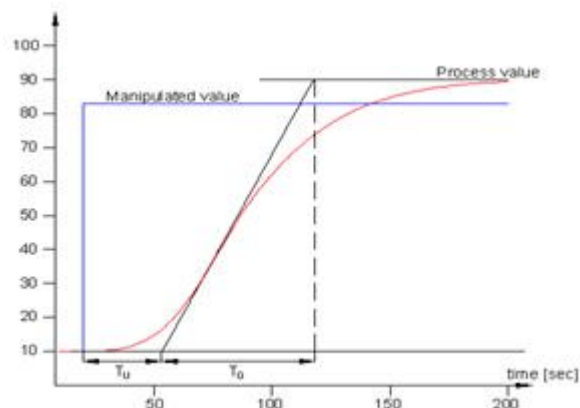
Итак, наконец, мы заметили, что, объединив эти три контроллера, мы получим желаемый ответ для системы. Различные производители разрабатывают различные алгоритмы ПИД. Методы настройки ПИД-регулятора

Перед началом работы ПИД-регулятора он должен быть настроен на динамику контролируемого процесса. Дизайнеры дают значения по умолчанию для параметров П, И, Д, и эти значения не могут дать желаемую производительность, а иногда приводят к нестабильности и медленным характеристикам управления. Разработаны различные методы настройки для настройки ПИД-регуляторов и требуют от оператора большого внимания для выбора наилучших значений пропорциональных, интегральных и дифференциальных коэффициентов. Некоторые из них приведены ниже.

Метод проб и ошибок: это простой способ настройки ПИД-регулятора. Пока система или контроллер работают, мы можем настроить контроллер. В этом методе сначала нужно установить значения K_i и K_d в нуль и увеличить пропорциональный коэффициент (K_p), пока система не достигнет колебательного поведения. Как только он осциллирует, отрегулируйте K_i (интегральный термин), чтобы колебания остановились и, наконец, отрегулировали Д, чтобы получить быстрый отклик.

Технологическая кривая технологического процесса: это метод настройки с открытым циклом. Он производит ответ, когда к системе применяется шаг ввода. Первоначально мы должны вручную вводить некоторые данные управления в систему и записывать кривую ответа.

После этого нам нужно рассчитать наклон, неподвижное время, время нарастания кривой и, наконец, подставить эти значения в уравнения П, И и Д, чтобы получить значения коэффициента усиления ПИД.



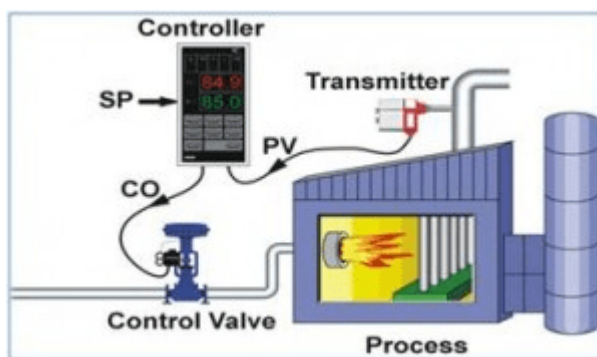
Метод Zeigler-Nichols: Zeigler-Nichols предложил методы замкнутого контура для настройки ПИД-регулятора. Это метод непрерывного циклирования и метод демпфирования колебаний. Процедуры для обоих методов одинаковы, но поведение колебаний различно. При этом сначала нужно установить постоянную р-контроллера, K_p на определенное значение, а значения K_i и K_d равны нулю. Пропорциональный коэффициент усиления увеличивается до тех пор, пока система не будет колебаться с постоянной амплитудой.

Усиление, при котором система производит постоянные колебания, называется конечным усилением (K_u), а период колебаний называется предельным периодом (P_c). Как только это достигнуто, мы можем ввести значения P, I и D в ПИД-контроллере по таблице Zeigler- Nichols, зависит от контроллера, используемого как P, PI или PID, как показано ниже.

	K_c	T_I	T_D
P	$K_u/2$		
PI	$K_u/2.2$	$P_c/1.2$	
PID	$K_u/1.7$	$P_c/2$	$P_c/8$

Структура ПИД-регулятора

ПИД-регулятор состоит из трех членов, а именно пропорционального, интегрального и дифференциального. Объединенная работа этих трех контроллеров дает стратегию управления процессом контроля. ПИД-регулятор управляет переменными процесса, такими как давление, скорость, температура, расход и т. д. В некоторых приложениях используются ПИД-регуляторы в каскадных схемах, где для достижения контроля используются два или более ПИД.



На приведенном выше рисунке показана структура ПИД-регулятора. Он состоит из блока PID, который дает свой вывод для обработки блока. Процесс / установка состоит из конечных устройств управления, таких как исполнительные механизмы, регулирующие клапаны и другие управляющие устройства для управления различными процессами промышленности / установки.

Сигнал обратной связи от технологической установки сравнивается с уставкой или сигналом $u(t)$, и соответствующий сигнал ошибки $e(t)$ подается на алгоритм ПИД-регулирования. Согласно пропорциональным, интегральным и дифференциальным расчетам в алгоритме, контроллер производит комбинированный ответ или управляемый выход, который применяется к устройствам управления установкой.

Все управляющие приложения не нуждаются во всех трех элементах управления. Комбинации, такие как элементы управления PI и PD, очень часто используются в

практических приложениях.

Лабораторная работа 7

Разработка и настройка PID контроллера для управления движением колесного робота

Абсолютное большинство современных роботов для гонок по линии высокого класса как основу алгоритма движения по линии используют ПИД-регулятор (он же ПИД-контроллер). В то же время для многих юных поклонников робогонок ПИД-регулятор остаётся сложным и малопонятным механизмом. Давайте попробуем вместе разобраться что такое ПИД-регулятор и как он работает.

Что такое «регулятор»

Само понятие ПИД-регулятор пришло из теории автоматического управления. Кто читает по-английски, то там эта дисциплина называется control theory. Эта наука занимается исследованием систем автоматического управления, а такие системы мы встречаем на каждом шагу. Автопилоты, системы наведения торпед и ракет, системы управления температурой в химическом производстве, регуляторы положения поглощающих стержней в атомном реакторе, система регулирования температуры в кондиционерах, поплавковый клапан в сливном бачке унитаза — это всё системы автоматического управления.

Общая схема системы автоматического управления нарисована ниже, а основными понятиями такой системы будут следующие:

1. Объект управления — это то, чем управляет система. В случае гоночного робота это сам робот, а точнее, положение робота относительно линии;
2. Уставка (цель, целевое значение) — это то условие, за соблюдением которого «смотрит» автомат. Для нашего случая целью является то, чтобы робот не отклонялся от линии;
3. Рассогласование (ошибка, ошибка управления) — это отклонение текущего состояния от желаемого. В нашем случае это отклонение робота от линии;
4. Датчики — то, чем мы определяем рассогласование. Для гоночного робота это обычно «линейка» — система из датчиков серого (пара светодиод/фототранзистор);
5. Обратная связь — сигнал с датчиков, ориентируясь на который автомат принимает решение об управляющем воздействии;
6. Регулятор. Ключевой элемент системы. Именно регулятор «принимает решение» о том что сделать, чтобы система вернулась к цели. В сливном бачке это механическая система, в случае гоночного робота это часть программы, реализующая ПИД-алгоритм;
7. Управление (управляющее воздействие, корректировка) — это действия, которые принимает система, чтобы вернуться к цели. В нашем случае это изменение скорости моторов робота;
8. Привод. Это та часть системы, которая реализует управляющее воздействие. В нашем случае это моторы.



Для демонстрации работы ПИД-контроллера дальше мы используем робота нашего клиента Марка из Казахстана.

Расчёт рассогласования

Первый расчётный блок в этом перечне — расчёт рассогласования. Как пример расчёта согласования можно взять такой: считаем, что в нашей линейке датчиков каждый датчик даёт выходной сигнал, например, 200 на белом поле и 1000 на чёрной линии. В этом случае мы можем считать, что если датчик показывает больше, чем 600, то он находится над линией. Меньше или равно 600 — над белым полем.

Этот алгоритм для системы из восьми датчиков линии на ардуино будет выглядеть примерно так:

```
float bot_position()
{
    float posSum = 0;
    float posMedian = 0;
    float signal[8];
    float signal_corrected;

    qtr.read(signal);

    for (int i = 0; i < 8; i++) {
        if (signal[i] > 600)
        {
            signal_corrected = 1;
        }
        else
        {
            signal_corrected = 0;
        }
        posSum += signal_corrected ;
        posMedian += signal_corrected * i * 2;
    }

    if (posSum > 0)
    {
        return posMedian / posSum - 7;
    }
    else
    {
        return 0;
    }
}
```

Здесь `signal` — массив для хранения результатов замера сигнала с каждого из восьми датчиков, а `qtr.read(signal)` — это функция, которая заполняет массив значениями с датчиков. Нетрудно убедиться, что данная функция вернёт 0 в том случае, если линия находится точно посередине линейки датчиков и положительное или отрицательное значение — в случае, если робот отклонился вправо или влево.

Это сильно упрощённый расчёт рассогласования, который обычно требует большого количества датчиков (8-14 на робогонках). Есть другой вариант — рассчитывать дробное значение положения робота на линии, пользуясь тем, что наш датчик по мере приближения с белого поля к чёрной линии будет показывать промежуточные значения между 200 и 600. В каких-то ситуациях при такой схеме даже робот с массивом из трёх датчиков линии может показывать хороший результат.

Обратная связь

Второй расчётный блок — обратная связь. Для [простого робота](#) обычно обратная связь реализована таким образом — регулятор рассчитывает величину поправки, которая добавляется к скорости одного мотора и вычитается из скорости другого мотора.

Например, задаём среднюю скорость для моторов как ШИМ 150 при 255 — максимальном

ШИМ для нашего контроллера. Далее если рассчитанное управление равно 50, то мы к правому мотору добавляем 50 (получаем ШИМ 200), от левого вычитаем 50 (получаем ШИМ 100). В случае, если рассчитанное управление будет отрицательным, -50, то у нас наоборот левый мотор будет двигаться быстрее.

П-регулятор

Буква «П» в названии П-регулятора означает слово «пропорциональный». П-регулятор берёт в расчёт текущее отклонение робота от линии и рассчитывает управление пропорционально текущему отклонению.

Например, наш расчёт рассогласования написан таким образом, что смещение робота на полдатчика в сторону от линии выдаёт ошибку рассогласования в одну единицу. Т.е. в случае если робот отклонился на полдатчика, то мы получим ошибку 1, если на один датчик, то ошибка будет 2, а если робот сместится на два датчика в сторону, то величина ошибки на выходе у нашей функции `bot_position()` будет четыре.

В этом случае мы можем написать П-регулятор так:

```
int avgSpeed = 150; // средняя скорость моторов
int kP = 10; // коэффициент пропорциональной обратной связи
int error; // Это ошибка положения
```

```
error = bot_position();
correction = kP * error;
```

```
motor1.move(avgSpeed*(1+correction));
motor2.move(avgSpeed*(1-correction));
```

Здесь функции `motor1.move()` и `motor2.move()` будут управлять скоростью левого и правого моторов, от -255 до 255. Видно, что если робот сместится в сторону на полдатчика, то скорости моторов изменятся, один мотор получит на вход ШИМ 165, а другой 135 и робот начнёт двигаться по дуге. Представим, что линия под роботом делает резкий поворот и радиус движения робота всё равно оказался недостаточным, чтобы вернуться на линию. В этом случае робот скоро сместится от линии в сторону уже на целый датчик, ошибка станет равна двум, а коррекция — 20. Скорости моторов изменятся и станут равны 180 и 120 соответственно и робот ещё активнее попытается вернуться на линию. Т.е. чем резче поворачивает линия под роботом, тем больше робот будет смещаться в сторону и тем больше будет разница скорости между моторами.

Как мы подберём коэффициент `kP`? Робот с низким `kP` называется недорегулированным. Такой робот не умеет поворачивать достаточно резко для того, чтобы оставаться на линии. Робот с высоким `kP` называется перерегулированным. Он начинает «дёргаться» и двигаться зигзагами, отклоняясь от линии то вправо, то влево. На видео выше видно, что робот немного перерегулирован — он двигается не «как влитой», а колеблется, причём бывает, что колебаниями линейку отводит влево от линии, несмотря, что робот в этот момент совершает поворот влево. Нормально отрегулированный робот с П-контроллером всегда поворачивает вправо когда нос находится слева от линии и влево — когда нос находится справа от линии, т.к. он «ищет потерянную линию».

Давайте посмотрим на робота со значительно перерегулированным П-контроллером и мы увидим, что он колеблется постоянно:

П-регулятор хорош для медленных роботов, для которых инерция не влияет на движение робота. Давайте повысим скорость робота и мы увидим, что уже настроенных значений `kP` не хватает — робот начинает сходить. Мы будем вынуждены повысить `kP`, чтобы робот не сходил с линии. При этом робот будет перерегулирован и начнёт двигаться зигзагами. Т.е. он может быть одновременно и недорегулированным — значений `kP` не хватает чтобы не сходить с линии, и перерегулированным — двигаться зигзагом. Причина этого одна — инерция. Мощности и скорости реакции двигателей и сцепления колёс с трассой уже не хватает чтобы робот моментально реагировал на команды регулятора. П-регулятор не предназначен для управления роботами с инерцией и нам надо переходить уже к следующему виду регуляторов — ПД.

ПД-регулятор

На видео с повышенной скоростью движения давайте обратим внимание на то, какие куски

трассы для робота оказались наиболее сложными. Видно, что робот оказывается недорегулированным на сопряжённых поворотах, т.е. в тот момент когда робот повернул сначала в одну сторону, а потом ему требуется повернуть в другую. В этот момент сказывается накопленная инерция и мощности двигателей уже не хватает чтобы быстро повернуть робота в другую сторону.

Д-регулятор (дифференциальный регулятор) работает таким образом — считается изменение рассогласования за некоторый промежуток времени, а дальше необходимое управление считается как произведение этого изменения на некий коэффициент, который мы обозначим kD . Мы получим ПД-регулятор добавив Д-регулятор к уже имеющемуся П-регулятору.

```
int prev_error;
```

```
void loop(void)
```

```
{
int avgSpeed = 150; // средняя скорость моторов
int kP = 10; // коэффициент пропорциональной обратной связи
int kD = 1; // коэффициент дифференциальной обратной связи
int error; // Это ошибка положения
```

```
error = bot_position();
correction = kP * error + kD * (error-prev_error);
prev_error = error;
```

```
motor1.move(avgSpeed*(1+correction));
motor2.move(avgSpeed*(1-correction));
```

```
delay(10);
}
```

Мы добавили здесь переменную `prev_error`, расчёт дифференциальной коррекции $kD * (error - prev_error)$, а также задержку в цикле — `delay(10)`. Д-регулятор как видно из формулы, борется с высокой скоростью изменений ошибки. Т.е. он борется с инерцией и с колебаниями, и является обязательным для роботов, которые двигаются с высокими скоростями.

Недостаток Д-регулятора — это то, что в нашем случае он требует задержки в цикле управления. Значение ошибки `error` в нашем случае меняется редко, надо чтобы робот проехал какое-то расстояние чтобы ошибка изменилась, поэтому если мы задержку не введём, то Д-регулятор будет действовать очень кратковременно, только в тот короткий момент когда робот будет переходить с датчика на датчик. Величина задержки в цикле должна быть достаточной, чтобы за этот момент значение ошибки могло измениться на пару датчиков. Но то, что хорошо для Д-регулятора, то плохо для П-регулятора, который потеряет возможность быстро реагировать на изменение положения робота.

Второй недостаток Д-регулятора — подверженность шумам. Случайные скачки освещённости могут привести к неожиданному его срабатыванию.

Давайте рассмотрим более профессиональный код ПД-регулятора для нашего робота:

```
int avgSpeed = 150; // средняя скорость моторов
int kP = 10; // коэффициент пропорциональной обратной связи
int kD = 5; // коэффициент дифференциальной обратной связи
```

```
int correction;
int err;
```

```
int err_arr[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int err_p = -1;
```

```
prevErr = bot_position();
```

```
void loop(void)
```

```
{
err = bot_position();
```

```

err_p = (err_p + 1) % 10;
err_arr[err_p] = err;

P = err * KP;
D = (err_arr[err_p] - err_arr[(err_p+11) % 10])*KD;

correction = P + D;

motor1.move(avgSpeed*(1+correction));
motor2.move(avgSpeed*(1-correction));

delay(2);
}

```

В данном примере последние десять ошибок хранятся в массиве `err_arr` и Д-регулятор берёт за основу вычисления коррекции разницу между текущим значением ошибки и значением ошибки 20 миллисекунд назад (время цикла $2\text{мс} * 10$, где 10 — количество элементов в массиве). Д-регулятор позволяет значительно повысить скорость движения робота.

Если значение коэффициента обратной дифференциальной связи слишком велико, то робот с ПД-контроллером начнёт «злиться» или «нервничать» — для такого робота характерны высокочастотные колебания, причём если для перерегулированной связи по «П» характерны колебания вправо-влево от желаемого направления движения, то перерегулировка по «Д» приводит к тому, что робот просто «дрожит».

Давайте уменьшим коэффициент `kD` и мы увидим, что робот стал идти намного ровнее, чем на П-контроллере, более того, теперь мы можем даже уменьшить коэффициент `kP` и робот всё-равно будет успешно проходить трассу. Этот робот может на ПД-контроллере идти более чем вдвое быстрее, чем на П-контроллере.

Видно, что робот стал ехать как будто на хорошо настроенном П-контроллере, хотя мы знаем, что на такой скорости П-контроллер с этим роботом бы не справился. Или его пришлось бы сильно перерегулировать и это было бы заметно по его колебаниям. Д-контроллер в нашем случае как-будто делает двигатели более мощными, улучшает баланс робота, и повышает точность линейки.

И-регулятор

Обратите внимание на предыдущем видео — робот в повороте движется «на крайних датчиках», сильно смещаясь от середины линии. Это особенность работы П-регулятора, для его корректной работы нужно чтобы робот двигался с ошибкой, а если нет ошибки, то нет и коррекции.

Тем не менее, есть простой регулятор, который отвечает как раз за движение робота в точности по центру линии. Это И-регулятор (интегральный регулятор). Посмотрите на видео ниже — робот с хорошо настроенным И-регулятором движется уже чётко посередине линии, даже в повороте.

Он считает управление как произведение некоторого коэффициента, который я назову `kI` на накопленную ошибку. Можно просто считать сумму всех ошибок «с начала времён». В нашем случае я посчитаю сумму последних десяти ошибок, благо мы их уже храним в массиве.

```

int avgSpeed = 150; // средняя скорость моторов
int kP = 10; // коэффициент пропорциональной обратной связи
int kD = 5; // коэффициент дифференциальной обратной связи
int kI = 5; // коэффициент интегральной обратной связи

```

```

int correction;
int err;

```

```

int err_arr[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int err_p = -1;

```

```

prevErr = bot_position();

```

```

void loop(void)

```

```

{
err = bot_position();
err_p = (err_p + 1) % 10;
err_arr[err_p] = err;

P = err * KP;
D = (err_arr[err_p] - err_arr[(err_p+11) % 10])*KD;
int err_sum = 0;
for (int i = 0; i < 10; i++) err_sum += err_arr[i];
I = err_sum/10*KI;

correction = P + I + D;

motor1.move(avgSpeed*(1+correction));
motor2.move(avgSpeed*(1-correction));

delay(2);
}

```

В этом коде вы уже видите полноценный ПИД-регулятор. Робот с настроенным ПИД-регулятором уже идёт посередине линии. А что будет если И-регулятор перенастроить, задать слишком большие значения коэффициента kI ? В этом случае И-регулятор будет мешать работать П-регулятору и робот опять начнёт дёргаться, как это видно на видео ниже. Обычно коэффициент kP несколько понижают в случае ПИД-регулятора по сравнению с ПД-регулятором, в такой схеме инерционный регулятор помогает пропорциональному.

Порядок настройки ПИД-регулятора

Обычный	порядок	настройки	робота	такой:
1.	На невысокой скорости настраиваем П-регулятор (подбираем значение kP такое, чтобы в самых крутых поворотах робот проходил держа линию близко к своим крайним датчикам).	kD и kI при этом равны нулю, т.е. пользуемся чисто П-регулятором;		
2.	Повышаем скорость, подбираем значение kD . Если при настройке П-регулятора робот шёл без инерции, то значение kP можно не менять. Если робот уже шёл с инерцией, как это обычно бывает у быстрых роботов, то значение kP надо будет понизить — мы это увидим по тому, что робот прекратит сильно отклоняться от линии благодаря помощи Д-регулятора;			
3.	Когда ПД-регулятор настроен, то можно подбирать значение I , уменьшая отклонения робота от линии. Значения коэффициентов kD и kP при этом обычно тоже несколько понижаются. И-регулятор полезен для гонок где линия может делать петли. Отклонение робота от прямой при этом чревато выбором ошибочного направления движения. При гонках на трассах без петель зачастую используется ПД-регулятор, так как в общем случае он позволяет развивать более высокую скорость.			

Лабораторная работа №8.
Программирование микроконтроллера робота

Цель работы: ознакомиться с принципом работы и основами программирования 8-Разрядных микроконтроллеров.

Методические рекомендации: самостоятельно изучить структуру микроконтроллера, его интерфейсы, работу внутренних регистров микроконтроллера, программные средства WinAVR, PonyProg, последовательность действий для программирования микроконтроллера, лабораторный макет проведения лабораторных работ.

Теоретическая часть

Микроконтроллер – компьютер на одной микросхеме, предназначенный для управления различными электронными устройствами и осуществления взаимодействия между ними в соответствии с заложенной в него программой.

В отличие от микропроцессоров, используемых в персональных компьютерах, микроконтроллеры содержат встроенные дополнительные устройства: устройства памяти, порты ввода/вывода (I/O), интерфейсы связи, таймеры, системные часы.

Эти устройства выполняют свои задачи под управлением микропроцессорного ядра микроконтроллера. На рисунке 1 изображена обобщенная структурная схема микроконтроллера.

В состав микроконтроллера входят: генератор тактового сигнала (GCK); процессор (CPU); постоянное запоминающее устройство для хранения программы, выполненное по технологии Flash (FlashROM); оперативное запоминающее устройство статического типа для хранения данных (SRAM); постоянное запоминающее устройство для хранения данных, выполненное по технологии EEPROM (EEPROM); набор периферийных устройств для ввода и вывода данных и управляющих сигналов и выполнения других функций.

Микроконтроллеры семейства AVR являются устройствами синхронного типа. Действия, выполняемые в микроконтроллере, привязаны к импульсам тактового сигнала.

В качестве генератора тактового сигнала может использоваться внутренний генератор с внешним кварцевым или керамическим резонатором (XTAL); внутренний RC-генератор (IRC); внутренний генератор с внешней RC-цепочкой (ERC); внешний генератор (EXT). Процессор формирует адрес очередной команды, выбирает команду из памяти и организует ее выполнение. Код команды имеет формат "слово" (8 бит) или "два слова" (16 бит). В состав процессора входят счетчик команд, арифметико-логическое устройство и блок регистров общего назначения.

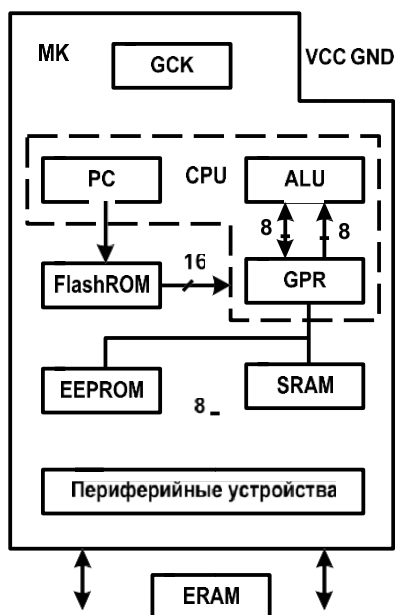


Рисунок 1 – Обобщенная структурная схема микроконтроллера

Счетчик команд представляет собой регистр, в котором содержится адрес следующей исполняемой команды. Размер счетчика команд зависит от объема имеющейся памяти программ. При нормальном выполнении программы содержимое счетчика автоматически увеличивается на 1 или на 2 в каждом такте. Этот порядок нарушается при выполнении команд перехода, вызова и возврата из подпрограмм, а также при возникновении прерываний.

После включения питания, а также после сброса микроконтроллера счетчик команд автоматически обнуляется, и первая команда выбирается из FlashROM по адресу 0, а при возникновении прерываний в него загружается адрес соответствующего вектора прерывания.

Блок регистров общего назначения служит для хранения промежуточных результатов вычислений и состоит из 32 регистров (R0...R31).

В арифметико-логическом устройстве выполняются арифметические и логические операции. Операнды поступают из регистров общего назначения. Каждая команда (16 арифметических и 16 логических) в качестве операндов использует либо содержимое двух разных регистров общего назначения, либо содержимое регистра и константу. Результат вычислений также помещается в один из регистров.

Постоянное запоминающее устройство FlashROM предназначено для хранения кодов команд программы и констант. Ячейка памяти содержит 16 разрядов. В ней могут храниться код команды формата "слово", половина кода команды формата "два слова" или коды двух констант. При чтении кодов команд адрес в FlashROM поступает из счетчика команд. При чтении констант адрес поступает из пары R30-R31 регистров общего назначения. Запись кодов в FlashROM выполняется в процессе программирования побайтно.

Оперативное запоминающее устройство статического типа SRAM предназначено для хранения данных, получаемых в процессе работы микроконтроллера. При выключении напряжения питания микроконтроллера данные в SRAM теряются.

Постоянное запоминающее устройство EEPROM предназначено для хранения данных, записанных при программировании микроконтроллера и получаемых в процессе выполнения программы. При выключении напряжения питания данные сохраняются. Ячейка памяти содержит 8 разрядов.

В группу периферийных устройств входят: параллельные порты ввода-вывода; последовательный периферийный интерфейс SPI; универсальный синхронный/асинхронный приемо-передатчик USART; двухпроводный последовательный интерфейс TWI (I2C); таймеры-счетчики общего назначения; сторожевой таймер и аналого-цифровой преобразователь; аналоговый компаратор; программируемый аппаратный модулятор; блок прерываний.

Параллельные порты ввода-вывода (PORTx) предназначены для ввода и вывода данных. Порт может иметь от трех до восьми выводов. Вывод порта может работать в режиме входа или в режиме выхода. Направление передачи бита устанавливается для каждого вывода в отдельности.

Последовательный периферийный интерфейс SPI имеет два назначения: с его помощью может осуществляться обмен данными между микроконтроллером и различными периферийными устройствами, такими как цифровые потенциометры, ЦАП/АЦП, Flash-ПЗУ, между несколькими микроконтроллерами, а также программирование микроконтроллера.

Универсальный синхронный/асинхронный приемо-передатчик USART предназначен для передачи и приема байтов данных по двухпроводным линиям связи (например, по интерфейсу RS-232). Длина посылки

может варьироваться от 5 до 9 бит. Во всех модулях USART присутствуют схемы контроля и формирования бита четности.

Интерфейс TWI или его, еще обозначают I2C, служит для передачи данных и позволяет объединить до 128 различных устройств с помощью двунаправленной шины, состоящей всего из двух линий: линии тактового сигнала и линии данных. Суммарная скорость передачи по интерфейсу – 400 Кбит/с.

Таймер-счетчик общего назначения предназначен для формирования запроса прерывания при истечении заданного интервала времени (режим таймера) или свершении заданного числа событий (режим счетчика). Основным элементом таймера-счетчика является базовый счетчик, который ведет счет на увеличение. При его переполнении формируется запрос прерывания.

Таймер-счетчик общего назначения может выполнять дополнительные функции: функцию захвата, сравнения, широтно-импульсного модулятора, счета реального времени.

Сторожевой таймер предназначен для ликвидации последствий сбоя в ходе программы путем перезапуска микроконтроллера при обнаружении сбоя.

Аналого-цифровой преобразователь формирует разрядный двоичный код числа, пропорционального величине напряжения аналогового сигнала на входе микроконтроллера. В микроконтроллерах AVR к преобразователю могут подключаться от четырех до восьми входов.

Аналого-цифровой компаратор сравнивает по величине аналоговые сигналы, поступающие на два входа микроконтроллера, и формирует запрос прерывания, когда разность их значений меняет знак. При этом также может быть выдан сигнал для выполнения функции захвата в таймере счетчика общего назначения.

Блок прерываний (Interrupt Unit, IU) организует переход к выполнению прерывающей программы при поступлении запроса прерывания, если прерывание по данному запросу разрешено, и он имеет более высокий приоритет, чем другие запросы, поступившие одновременно с ним. Приоритетность запросов задана аппаратно.

В микроконтроллере особая четырехбайтовая область памяти отведена для конфигурационных бит (fuse bits) и бит защиты (lock bits), которые служат для глобальной настройки микрочипа. Если содержание регистров памяти микроконтроллер может изменять сам в процессе выполнения программы, то изменение fuse и lock битов доступно только с использованием программатора. При программировании этих битов нужно быть предельно внимательным и постоянно сверяться с Datasheet, так как можно навсегда испортить микроконтроллер.

Программа для микроконтроллера может быть написана на языках: Assembler, C/C++, Pascal, Basic, FlowCode или Scratch и т.д. Каждый из них имеет свои преимущества и недостатки в решении определенных задач, например, Assembler позволяет наиболее полно раскрыть все возможности микроконтроллеров и получить максимальное быстродействие и компактный код, но сама программа громоздка и труднопонижаема, визуальные языки наиболее просты, однако могут быть проблемы с оптимизацией кода. Широкое распространение получил язык C/C++ благодаря огромному числу программных средств и библиотек, позволяющих просто создавать необходимый код. C/C++ сегодня стал основным языком разработки управляющих программ.

Вышеприведенные языки не понятны микроконтроллеру. Для перевода программы в машинный код, который будет вшит в память микрочипа, используют специальные программы компиляторы.

Программирование осуществляется с помощью специальных компьютерных программ-программаторов. Для сопряжения интерфейсов ПК с интерфейсами микроконтроллера применяют аппаратные программаторы устройства, зачастую на основе другого микрочипа. Программатор является промежуточным звеном между компьютером и программируемым микроконтроллером. Со стороны микроконтроллера программатор может использовать интерфейс SPI для последовательного программирования, JTAG – для программирования и отладки в режиме реального времени, или несколько выводов, специально предназначенных для параллельного программирования.

На рисунке 2 изображены структура и электрическая схема платы для выполнения лабораторной работы на основе микроконтроллера ATtiny2313.

Используемый в лабораторной работе микроконтроллер ATtiny2313 выполнен в корпусе типа DIP и имеет 20 выводов, из которых 18 приходятся на 3 параллельных порта ввода-вывода PORTA, PORTB и PORTD и на питание: к выводу VCC – подключается положительное напряжение до +5В, а вывод GND заземляется. PORTA и PORTD не полные: отсутствуют выводы PA3-PA7 и PD7 соответственно.

Как видно в схеме (рис. 2), выводы PORTB подключены через сопротивления к светодиодным индикаторам и одновременно выводятся на 10 пиновый порт №1 для подключения внешних устройств. Для отключения светодиодной панели индикации предусмотрены джамперы.

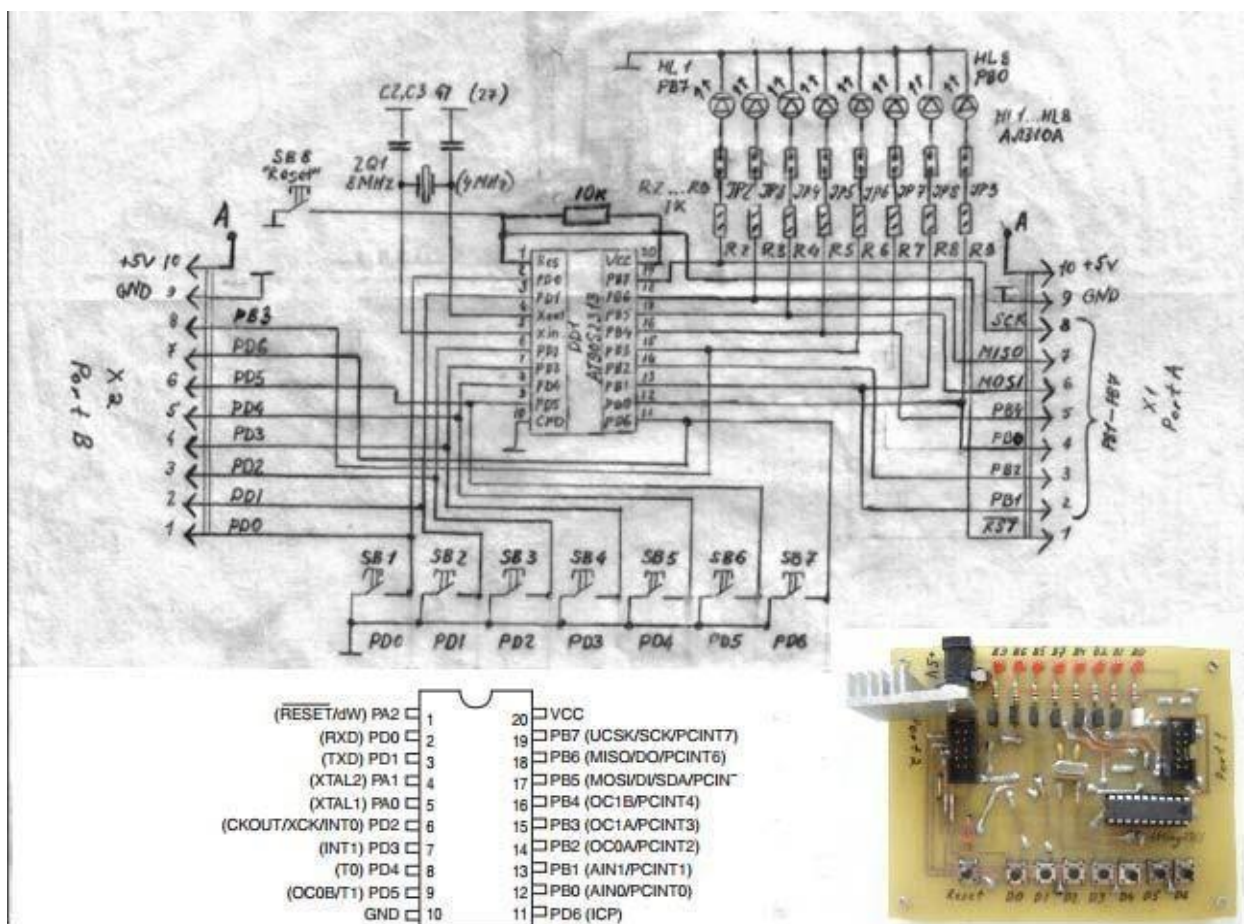


Рисунок 2 – Структура и электрическая схема платы на основе ATtiny2313

К выводам PORTD подключены кнопки, при нажатии которых происходит подключение их к земле. Одновременно выводы PORTD выводятся на 10-пиновый порт №2 для подключения внешних устройств.

Питание микроконтроллера осуществляется с помощью стабилизатора напряжения на 5 В через вывод VCC и GND.

Каждый из 18 выводов портов ввода-вывода выполняет альтернативные функции, например, для тактирования микроконтроллера к выводам PA0 и PA1 (XTAL1, XTAL2) подключен кварцевый резонатор на 8 МГц, SPI интерфейс, через который будет программироваться микроконтроллер использует выводы PB5, PB6, PB7 (MOSI, MISO, SCK соответственно). Интерфейс USART для приема/передачи данных использует выводы PD0, PD1, PD2 (RXD, TXD, XCK соответственно). Вывод PA2 (RESET) предназначен для внешнего сброса микроконтроллера и при подаче на него напряжения низкого уровня с помощью кнопки Reset, порты ввода-вывода немедленно возвращаются к их первоначальному состоянию, микроконтроллер перезагружается и активизируется только при подаче напряжения высокого уровня. В процессе программирования на выводе RESET должно присутствовать напряжение низкого уровня, а в процессе выполнения программы микроконтроллером – высокое.

Параллельные порты ввода-вывода предназначены для ввода и вывода бит данных. На каждом выводе порта может присутствовать напряжение высокого уровня, соответствующее логической единице, либо напряжение низкого уровня, соответствующее логическому нулю. Для каждого порта ввода/вывода в адресном пространстве микроконтроллера выделены 3 регистра: регистр данных порта PORTx, регистр направления данных DDRx и регистр выводов порта PINx (к примеру, для порта B – DDRB, PORTB, PINB).

Регистр DDRx устанавливает выводы порта x как вход или как выход данных. Если в него записаны 0, то порт сконфигурирован на вход, если записаны 1 на выход. Определить направление данных можно для каждого вывода порта. Например, если записать DDRB=0b00000000, то все 8 выводов порта B сконфигурированы как входы, DDRB=0b11111111, то все 8 выводов порта B сконфигурированы как выходы, DDRB=0b00001111, то первые 4 вывода порта PB0...PB3 сконфигурированы как выходы, а остальные 4 вывода PB4...PB7 – как входы.

Регистр PORTx имеет двойное назначение. Если порт настроен на выход данных, то значение регистра PORTx определяет напряжение на выводах порта. Например, если записать DDRB=0b11111111 и PORTB=0b10111000, то на выводах PB0, PB1, PB2, PB6 окажется напряжение низкого уровня, на выводах PB3,

PB4, PB5, PB7 напряжение высокого уровня.

Если же порт настроен на вход (DDRx=0b00000000), то значение регистра PORTx определяет состояние внутреннего подтягивающего резистора на каждом выводе порта. Если в регистр PORTx записать единицы, то ко всем выводам порта x будут подключены подтягивающие резисторы. Подтягивающий резистор соединяет вывод порта с питанием микроконтроллера и служит для защиты от помех, при этом сигнал, на свободно висеющей ножке, микроконтроллером расценивается как единица.

Регистр PINx указывает текущее состояние на выводах порта x и до ступен только для чтения. Фактически регистр PINx используется при программировании в качестве переменной, а его значение соответствует значениям потенциалов на входных ножках.

Используя эти регистры, напомним программу, реализующую непрерывное последовательное мигание светодиодов, подключенных к выводам PB0 и PB1, если кнопка D1 не нажата, а при нажатии и удерживании кнопки D1, подключенной к выводу PD1, будет осуществляться последовательное мигание светодиодов, подключенных к выводам PB3 и PB4. После написания программы ее нужно будет скомпилировать и записать в память микроконтроллера (последнее действие называется программированием). Итак, выполним все последовательно:

- 1) Откроем «Programmer's Notepad», входящий в пакет программ WinAVR. Перейдем по вкладкам

File → new → C/C++. Появится окно для написания программы на языке C/C++.

Сначала необходимо с помощью оператора #include подключить библиотеки для нашего микроконтроллера, в данном случае это ATtiny2313, располагающиеся в папках программы WinAVR. Понадобится библиотека iotn2313.h, в которой сопоставляются названия регистров с их адресами, и библиотека delay.h, в которой осуществлена программа функции задержки

delay(ms), пересчитывающая секунды в количество тактов GCK генератора:

```
#include <avr/io.h>
```

```
#include <util/delay.h>
```

Любая программа на языке C/C++ начинается с функции main:

```
int main (void)
```

```
{  
}
```

Слово int означает, что функция main является функцией целого типа, то есть возвращаемое ею значение – целое число. Слово void в скобках означает, что функция не принимает значений.

Порт B, на котором расположена светодиодная панель индикации, сконфигурируем как выход, а порт D с кнопочной панелью управления как вход:

```
DDRB=0xff; DDRD=0x00;
```

Для защиты от помех при нажатии/отжатии кнопок, подключим все выводы порта D к внутренним подтягивающим резисторам:

```
PORTD=0xff;
```

Для того, чтобы светодиоды изначально не горели установим на все выводы порта B напряжение низкого уровня:

```
PORTB=0x00
```

Для того, чтобы светодиоды B0...B3 загорались необходимо подать на соответствующие им выводы напряжение высокого уровня (логическая единица), потухли – напряжение низкого уровня (логический 0):

```
PORTB=0x01; PORTB=0x00; PORTB=0x02; PORTB=0x00; PORTB=0x04; PORTB=0x00; PORTB=0x08;  
PORTB=0x00;
```

Если оставить все без изменений, то светодиоды будут загораться и гаснуть с частотой работы тактирующего генератора микроконтроллера (8МГц), и мы увидим 4 горящих светодиода, так как они не будут успевать гаснуть. Для того, чтобы визуально было заметно, когда светодиод гаснет, применим функцию задержки _delay_(ms), в качестве аргумента которой указывается число, соответствующее времени задержки в мс:

```
PORTB=0x01;
```

```
_delay_(500); PORTB=0x00;
```

```
_delay_(500); PORTB=0x02;
```

```
_delay_(500); PORTB=0x00;
```

```
.....
```

Теперь светодиоды будут загораться и гаснуть с интервалом 500 мс. Первые два светодиода должны мигать при отжатой кнопке, другие два – при нажатой. При отжатой кнопке D1 на выводе PD1 присутствует напряжение высокого уровня, благодаря внутреннему подтягивающему резистору, а при нажатой происходит утечка тока на землю и на выводе PD1 напряжение становится низкого уровня. Используем операторы if-else и регистр PIND:

```
if (PIND==125) //соответствует значениям битов в регистре 01111101
```

```
{
```

```
PORTB=0x01;
```

```

_delay_(500); PORTB=0x00;
_delay_(500); PORTB=0x02;
_delay_(500); PORTB=0x00;
}
else {
PORTB=0x04;
_delay_(500); PORTB=0x00;
_delay_(500); PORTB=0x08;
_delay_(500); PORTB=0x00;
}

```

Для получения бесконечного цикла используем оператор условия while (), который выполняется до тех пор, пока значение в скобках истинно, то есть отлично от нуля.

Конечная программа имеет вид:

```

#include <avr/io.h>
#include <util/delay.h> int main (void)
{
  DDRB=0xff; DDRD=0x00; PORTD=0xff; PORTB=0x00;
  while (1)
  {
    if (PIND=125)
    { PORTB=0x01;
      _delay_(500); PORTB=0x00;
      _delay_(500); PORTB=0x02;
      _delay_(500); PORTB=0x00;
    }
    else
    {
      PORTB=0x04;
      _delay_(500); PORTB=0x00;
      _delay_(500); PORTB=0x08;
      _delay_(500); PORTB=0x00;
    }
  }
}

```

- 2) Сохраняем файл с программой как main.c.
- 3) Для того чтобы скомпилировать написанную программу, необходимо в программе Mfile, входящую в пакет программ WinAVR, создать Makefile и сохранить его в папке с основной программой main.c.
- 4) Открываем программу Mfile. Проходим по вкладкам Makefile → Main file name и прописываем имя нашего компилируемого файла main. с без расширения. Выбираем модель микроконтроллера, пройдя по вкладкам Makefile → MCU type → ATtiny → attiny2313. Выбираем расширение hex для скомпилированного файла, пройдя по вкладкам Makefile → Output for mat → ihex. Сохраняем Makefile в папке с основной программой main.c.
- 5) В программе Programmer's Notepad производим компиляцию, пройдя по вкладкам Tools → [WinAVR] Make All. Скомпилированный файл main.hex появится в той же папке с main.c.
- 6) Открываем программу PonyProg и производим ее настройку. Для начала выбираем тип программатора Setup → Interface Setup, выбираем LPT порт подключения программатора и тип программирования Avr ISP I/O. В главном окне программы выбираем тип ядра микроконтроллера AVR micro и модель ATtiny2313. Подключаем LPT-программатор к порту №1 платы, а блок питания в гнездо питания микроконтроллера. Пройдя по вкладкам Command → Read Program (FLASH), прочитаем flash память микроконтроллера, таким образом, проверив правильность выбранных настроек программатора и микроконтроллера. Если память прочитана, то можно начинать программировать микроконтроллер. Для этого загружаем файл с программой main.hex, пройдя по вкладкам File → Open Program (FLASH) File и записываем программу во flash-память микроконтроллера, пройдя по вкладкам Command → Write Program (FLASH).
- 7) Отключаем программатор от порта №1 платы. Перезагружаем микроконтроллер кнопкой Reset. Проверяем, что программа работает правильно.

Задание: на основе теоретических знаний и примеров программ, изложенных в лабораторной работе, написать программы, осуществляющие:

- 1) Непрерывное последовательное переключение светодиодов (в виде змейки), подключенных к порту PORTB микроконтроллера.

- 2) Непрерывное последовательное переключение светодиодов, подключенных к порту PORTB микроконтроллера, которое при нажатии и удерживании кнопки D0, меняет направление переключения на обратное.
- 3) При нажатии и удерживании кнопок, подключенных к D реги стру выводов, светятся соответствующие номера светодиодов. При отпускании кнопки соответствующий светодиод выключается.

Контрольные вопросы

1. Назовите основные структурные блоки микроконтроллера и их функциональное назначение?
2. Как осуществляется тактирование микроконтроллера и какие источники при этом могут использоваться?
3. Из каких функциональных блоков состоит процессор микроконтроллера? Назовите функцию каждого блока.
4. Какие устройства микроконтроллера предназначены для хранения кодов программ?
5. Какие периферийные устройства входят в состав микроконтроллеров?

Рекомендуемая литература

- 1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / В. В. Гребнев. М.: ИП РадиоСофт, 2002 □□176 с: ил. С. 9–23.
- 2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека XXI», 2007. – 432 с.: ил. С. 9–77.
- 3 ATTINY2313 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash ATMEL Corporation. Сайт документации на электронные компоненты. (Англ.). URL: <http://www.alldatasheet.com/datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html>
- 4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. □□СПб.: Наука и Техника, 2007. □□352 с.: ил.
- 5 Белов А. В. Самоучитель по микропроцессорной технике. □□СПб.: Наука и Техника, 2003. □□224 с.: ил.
- 6 Белов А. В. Создаем устройства на микроконтроллерах. □□СПб.: Наука и Техника, 2007. □□304 с.: ил.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.

ЛАБОРАТОРНАЯ РАБОТА №2. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ДВИЖЕНИЯ РОБОТА

Цель работы: получить навыки работы со средой программирования Robolab.

Объем работы: 3 астр. часа.

Методические рекомендации:изучить интерфейс Robolab

Теоретическое введение

Основные элементы интерфейса программной среды *Robolab* представлены на рис. 122.

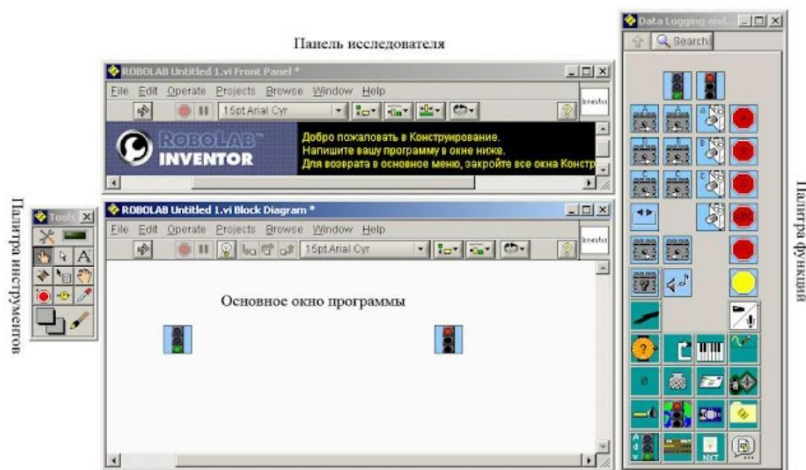


Рис. 1 Окно программной среды *Robolab*

Команды в программе задаются в виде пиктограмм, которые находятся в Палитре команд (функций). Там представлены команды действия, ожидания, управляющие структуры, модификаторы.

Соединение пиктограмм между собой имеет ряд свойств:

- щелчок в любом месте окна, кроме пиктограммы, фиксирует точку для поворота провода;
- двойным щелчком в области окна программы заканчивается прокладка начатого провода;
- при нажатии клавиши пробел провод поворачивается. Программная среда *Robolab* позволяет создавать линей-

ные алгоритмы с ветвлением и циклом. Два светофора обозначают начало и конец программы.

Цикл применяется для многократного повторения команд. Он может иметь параметр, который определяет, сколько раз вы полняете команды между началом и концом цикла. Команды Прыжок (стрелка вверх) и Метка (стрелка вниз) служат для изменения порядка выполнения команд. Они позволяют делать в программе петли (возвращаться назад) или пропускать команды (прыгать вперед).

Ветвление применяется для проверки различных условий. Когда программа достигает развилки, она проверяет значение, указанное на развилке. Если значение больше указанного, программа дальше следует вдоль провода, присоединенного к знаку

«больше» (>), если меньше или равно, то по проводу знака

«меньше или равно» (<=).

Практические задания

Задание 1. Запустите программу *Robolab*. При запуске программы пользователю предлагается три режима «Администратор», «Программист», «Исследователь». Выберите режим «Администратор» и измените прошивку *NXT*. Вернитесь к выбору режимов работы в *Robolab*, выберете режим «Программист», затем в разделе Конструирование – уровень *Invertor 4*.

Задание 2. Составьте программу для движения робота по квадрату:

- повторить 4 раза:
 - моторы *B*, *C* вперед в течение 1 секунды;
 - мотор *C* назад (*B* вперед) в течение $N/100$ секунд;
- стоп моторы.

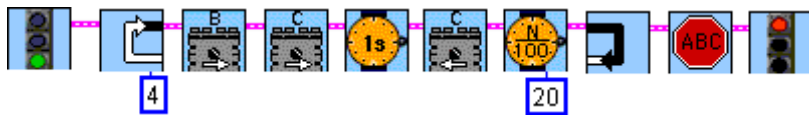


Рис. 2. Программа к заданию 2

Задание 3. Составьте программу для движения робота в круге, ограниченном черной линией.

Повторять бесконечно:

- моторы *B*, *C* вперед;
- жди темнее на датчике освещенности (среднее арифметическое между показаниями датчика освещенности на темной и на светлой областях поля);
- изменить направление моторов *B*, *C*;
- жди $N/100$ секунды;
- мотор *C* вперед (*B* назад);
- жди $N/100$ секунды.

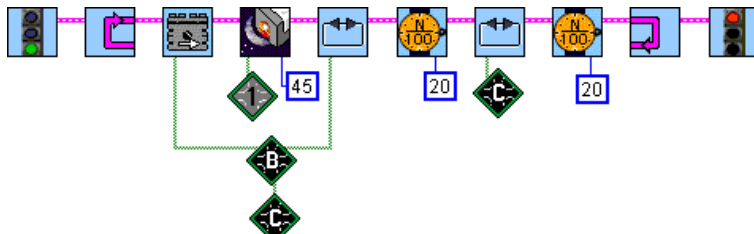


Рис. 3. Программа к заданию 3

Задание 4. Составьте программу для робота, который движется по черной линии (перед запуском программы установить робота слева от черной линии). В качестве граничного значения для датчика освещенности используется среднее значение между его показаниями на светлом участке поля и на темном. Повторяй бесконечно:

- плавный поворот направо до черного;
- плавный поворот налево до белого.

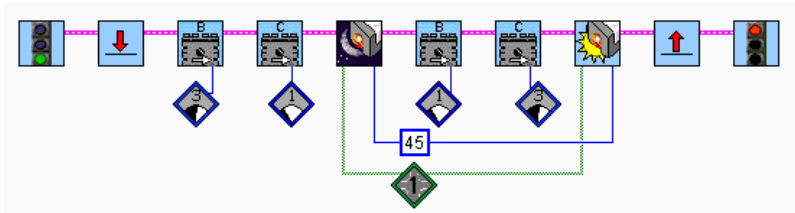


Рис. 4. Программа к заданию 4

Задание 5. Составьте программу для робота, который путешествует по комнате (датчик расстояния смотрит вперед).

Повторяй бесконечно:

- моторы *B, C* вперед;
- жди ближе 30 на датчике расстояния;
- изменить направление моторов *B, C*;
- жди $N/100$ секунды;
- мотор *C* вперед (*B* назад);
- жди $N/100$ секунды.

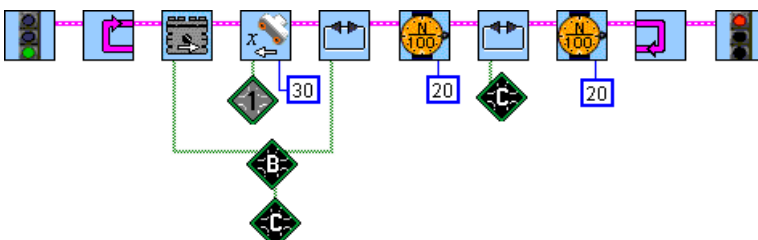


Рис. 6. Программа к заданию 5

Задание 6. Составьте программу для робота, который движется по черной линии. В качестве граничного значения для датчика освещенности используется среднее значение между его показаниями на светлом участке поля и на темном.

Повторяй бесконечно:

- если показания датчика освещенности выше среднего значения, поворачивай колесом *B*;
- если показания датчика освещенности ниже среднего значения, поворачивай колесом *C*;
- жди $N/1000$ секунды.

Перед запуском программы установить робота слева от черной линии.

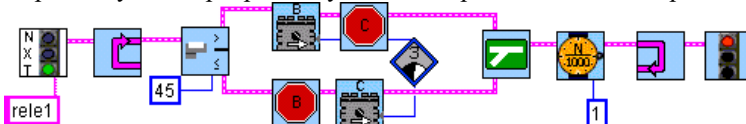
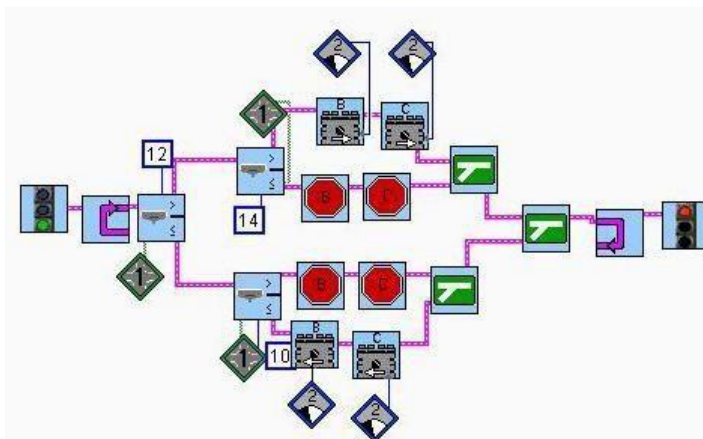


Рис. 7. Программа к заданию 6

Задание 7. Составьте программу для робота прилипалы: если расстояние до ближайшего объекта находится в заданном интервале, робот остается на месте, если это расстояние больше, — едет назад.

Рис. 8. Программа к Контрольные вопросы

1. Какие преимущества и



заданию 7

недостатки имеет

- среда программирования *Robolab* по сравнению с *Lego MindstormsNXTSoftware*?
2. Какие алгоритмические конструкции использовались Вами при выполнении заданий лабораторной работы?
 3. Какие элементы знаний по физике и математике использовались Вами при выполнении заданий лабораторной работы?

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

4. тему лабораторной работы;
5. условие задачи;
6. полный окончательный код примера; результаты ее тестирования.

Лабораторная работа №9. Управление роботом с помощью инфракрасных датчиков

Цель работы: изучить способы управления роботом с помощью датчиков.

Объем работы: 3 астр. часа.

Методические рекомендации:

Инфракрасный датчик от Lego для EV3 позволяет определять сигналы в инфракрасном диапазоне волн, посылаемые, например, с удаленного инфракрасного маяка (ИК-маяка) и посылать собственный инфракрасный сигнал и определять его отражение от других объектов.



Рис. 1. Внешний вид инфракрасного датчика и ИК-маяка для EV3

Инфракрасное излучение занимает в электромагнитном спектре диапазон между красной границей видимого спектра и микроволнами (от 750 нм до 1 мм), его часто называют тепловым излучением. Этот тип сигнала используется в большинстве телевизионных пультов управления. Как и видимый свет, инфракрасное излучение блокируется находящимися на его пути объектами, поэтому между ИК-маяком и инфракрасным датчиком должно быть свободное пространство. Солнечный свет может влиять на инфракрасные сигналы, но обычное комнатное освещение не должно оказывать влияния.

Инфракрасный датчик имеет три разных режима: «Приближение», «Маяк» и «Дистанционное управление». В режиме «Приближение» инфракрасный датчик посылает свой инфракрасный сигнал и может определить отражение этого сигнала от объекта, находящегося перед датчиком. Сила отраженного сигнала может использоваться для определения расстояния до объекта.

Значение приближения прямо не соответствует определенному расстоянию. Если объект находится на расстоянии менее 1 см, то датчик не в состоянии его зафиксировать. Датчик выдает относительные значения расстояния до объекта (0 значит очень близко, 100 – далеко). На показания датчика влияют цвет и материал объекта, а также другие факторы. Это обусловлено тем, что цветные объекты отражают инфракрасный свет лучше, чем темные объекты.

В режиме «Маяк» инфракрасный датчик от Lego может обнаруживать приблизительное положение

дистанционного инфракрасного маяка (ИК-маяка) перед датчиком. Датчик может дать значение приближения маяка (относительного расстояния от датчика) и его направление (угол от направления, которое указывает датчик). Режим «Маяк» можно использовать, например, для того, чтобы заставить робота искать ИК-маяк и двигаться к нему. В режиме «Дистанционное управление» инфракрасный датчик может определять нажатие кнопок на ИК-маяке.



Рис. 2. Детектор инфракрасного излучения *HiTechnic*.
Инфракрасный мяч *HiTechnic*

Датчик для обнаружения источника инфракрасного излучения *HiTechnic (InfraredSeeker)* посредством пяти инфракрасных сенсоров, направленных в разные стороны через каждые 60° , определяет, откуда исходят ИК-сигналы и их интенсивность. Инфракрасный мяч *HiTechnic* генерирует электромагнитное излучение в инфракрасном спектральном диапазоне. Питание мяча осуществляется от четырех элементов питания типа ААА. Мяч функционирует в четырех режимах работы: режим *A* – режим излучения затухающий, частота импульсов 1200 Гц, период – 833 м/с, время работы в данном режиме не менее 7 часов 30 мин; режим *B* – постоянное ИК-излучение, время работы в данном режиме не менее 1 часов 20 мин; режим *C* – частота импульсов 600 Гц, период – 1667 м/с, время работы в данном режиме не менее 3 часов 30 мин; режим *D* – частота импульсов 1200 Гц, период – 833 м/с, время работы в данном режиме не менее 3 часов 30 мин.

Датчик для обнаружения объектов

Датчик для обнаружения объектов *HiTechnic (EOPDSensor)* даёт возможность построить робота, который сможет обнаруживать объекты при помощи отраженных световых сигналов. Предусмотрено две настройки чувствительности в зависимости от удаленности объекта. Встроенная функция автоматической коррекции окружающего освещения позволяет использовать его как при ярком освещении, так и в затемнённых помещениях.



Рис. 3. Датчик для обнаружения объектов *HiTechnic*

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.
4. Постройте функции преобразования для датчиков звука и света *LegoMindstormsNXT*.
5. Заполните таблицу (для датчиков, используемых в лабораторном практикуме): Характеристики датчиков учебного робота

№	Датчик	Диапазон измерений	Погрешность измерений	Единицы измерения выходной величины

Лабораторная работа №10

Обнаружение и отслеживание движения с помощью цифровой камеры

Цель работы: изучить способы взаимодействия робота и цифровой камеры.

Методические рекомендации:

Теоретическое введение

Arduino – удобная платформа быстрой разработки электронных устройств для новичков и профессионалов. Представляет собой небольшую плату с собственным процессором, памятью. На плате имеется достаточно большое количество контактов, к которым можно подключать всевозможные компоненты: лампочки, датчики, моторы другие электронные компоненты.

Программы для *Arduino* пишутся на языке C++, дополненным простыми и понятными функциями для управления вводом/выводом на контактах. Для удобства работы с *Arduino* существует бесплатная официальная среда программирования

«*Arduino IDE*», работающая под *Windows*, *Mac OS* и *Linux*. Для изучения образовательной робототехники можно использовать образовательный набор «Амперка», в комплект которого входит в том числе и микроконтроллер *Arduino*.



Рис. 1. Платформа *Arduino* «Амперка»



Рис. 2. Образовательный набор «Амперка»

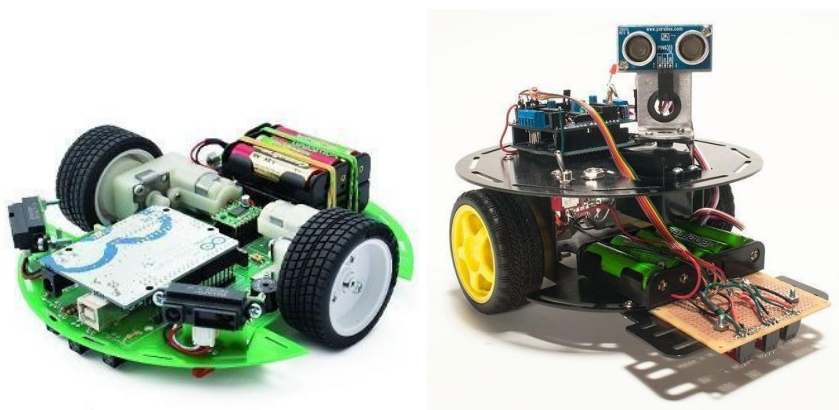


Рис. 3. Модели учебных мобильных роботов на базе *Arduino*

Bioloid

Это семейство образовательных робототехнических конструкторов, предназначенных для обучения в области разработки и отладки сложных робототехнических систем. Название *Bioloid* получено путем сложения слов «bio» + «all» + «oid», что отражает концепцию данного конструктора: любое живое существо может быть представлено в виде робота. Для программирования микроконтроллера CM-5 служит среда *RoboPlus* (см.рис.

22). Движение созданных роботов обеспечивается сервоприводами *Dynamixel AX-12A* или *AX-18A*.

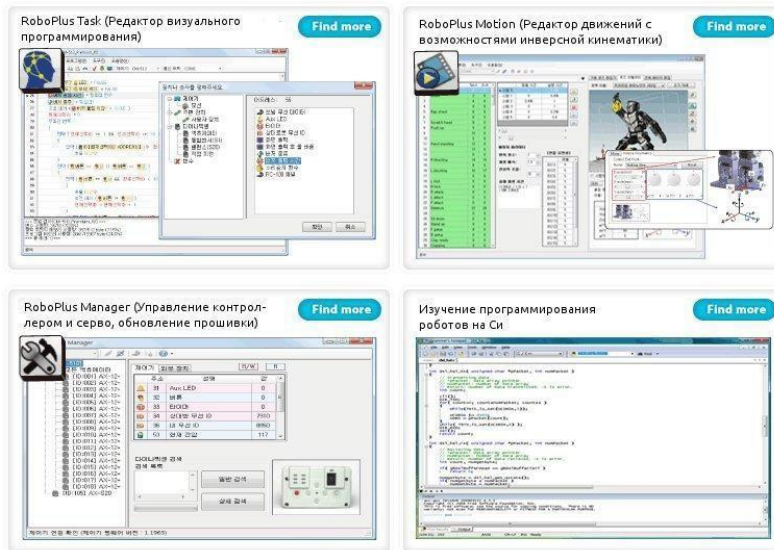


Рис. 4. RoboPlus (среда разработки роботов Bioloid)

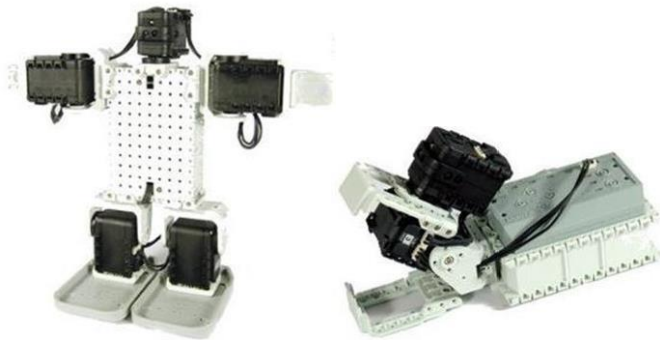


Рис. 5. Модели учебных роботов «Пингвин», «Крокодил» серия «*Robotics Bioloid* новичок (*BeginnerKit*)»



Рис. 6.. Модели учебных роботов «Королевский паук», «Гуманоид» серия «*Robotis Bioloid* комплексный (*Comprehensive Kit*)»

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Подготовьте реферат по одной из тем:
 - 1.1. Использование элементов робототехники на уроках физики;
 - 1.2. Использование элементов робототехники на уроках информатики;
 - 1.3. Использование элементов робототехники на уроках математики;
 - 1.4. Использование элементов робототехники на уроках технологии;
 - 1.5. Использование элементов робототехники на уроках биологии;
 - 1.6. Образовательная робототехника в начальной школе.
2. Используя рекомендованные печатные издания и интернет-источники составьте сводную таблицу по

оборудованию для изучения образовательной робототехники (название конструктора, образовательные задачи, решаемые при использовании данного конструктора, предполагаемый возраст учащихся, возможности использования в учебном процессе, преимущества, недостатки).

3. Используя интернет-источники, самостоятельно соберите информацию о конструкторах (отечественных разработках), которые можно было бы использовать на занятиях по образовательной робототехнике в школе.

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.

Лабораторная работа 11 **Обнаружение и отслеживание объектов с помощью цифровой камеры**

Цель работы: изучить способы взаимодействия роботов и цифровой камеры.

Методические рекомендации:

Серия конструкторов *Roborobo* представлена пятью комплектами, отличающимися по уровню сложности. В каждый из них входят: текстолитовые платы, металлические соединители и уголки, моторы и сервоприводы, колеса и гусеницы, контроллер для моторов и сервоприводов, пульт дистанционного управления, программное обеспечение, кабели для подключения к компьютеру. Для программирования *Roborobo* используется программное обеспечение *ROGIC*, оно позволяет начинающим легко освоить азы программирования.



Рис. 1. Модели учебных роботов Roborobo

Fischertechnik

Робототехнические конструкторы *Fischertechnik* созданы немецким ученым – профессором Артуром Фишером. Эти конструкторы во многом не уступают *LEGO Mindstorms*, они имеют аналогичные составляющие элементы. Робототехнические конструкторы *Fischertechnik* рассчитаны на учащихся старше 10 лет. В комплекты конструкторов входят программируемые контроллеры (*Robo TX*, *Robotics TXT*), двигатели, различные датчики и блоки питания, что позволяет приводить механические конструкции в движение, создавать роботов и программировать их с помощью компьютера. Для разработки управляющих программ для контроллера *Robo TX* используется среда программирования *Robo Pro*. Программы составляются на графическом языке в виде блок-схем. Готовые программы загружаются в контроллер через интерфейсы *USB* или *Bluetooth*.



Рис. 2. Учебная модель
«Автоматический высотный стеллажный склад из серии»
«RoboTX. Автоматические роботы»

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.

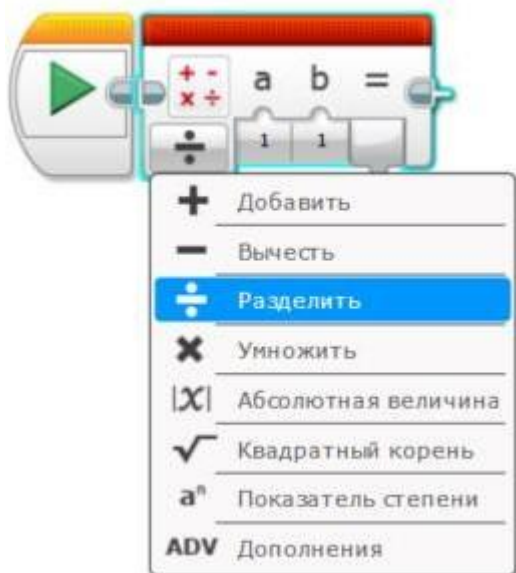
Лабораторная работа 12.

Программирование регулируемого движения робота

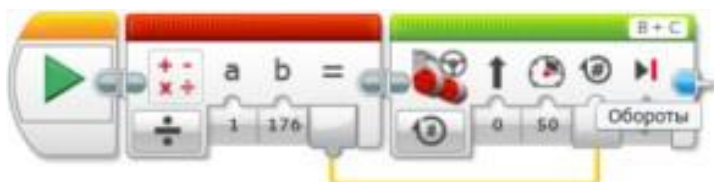
Цель работы: изучить методы программирования робота.

Методические рекомендации:

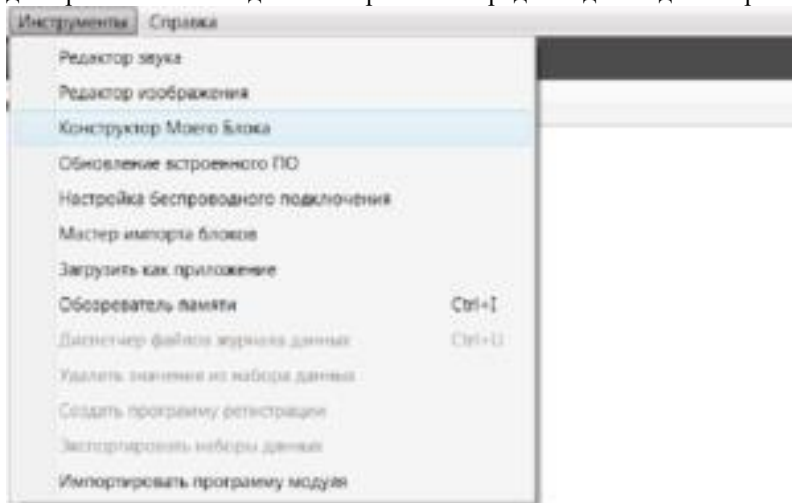
Составление программы начинаем с использования блока **Математика** в режиме **Разделить**:



В параметр b впишем значение длины окружности, а выход блока **Математика** подадим на вход **Обороты** блока **Рулевое управление**.



Выделяем оба блока и с помощью меню Инструменты > Конструктор моего блока создадим собственный блок для прямолинейного движения робота вперед/назад на заданное расстояние с заданной скоростью.



В окне **Конструктор моего блока** в поле **Имя** впишем название нашего блока **Move**. Щелчком по кнопке «+» добавим два параметра:



Затем переключаемся во вкладку **Значки параметров**. Для первого параметра выбираем значок «X», для второго — значок скорости. После этого нажимаем кнопку **Завершить**.



Следующим шагом, в появившемся новом окне программы, необходимо подключить выходы наших значков, обозначающих расстояние X и скорость, ко входам, как показано на рисунке ниже.



Затем переключаемся во вкладку с основной программой Program:



В этом окне появляется новый блок, с помощью которого мы можем программировать движение нашего робота на прямолинейных участках с заданной длиной пути. Первый параметр блока указывает расстояние в миллиметрах, второй — скорость движения. Для движения назад необходимо один из параметров ввести со знаком «-».

Следует учесть, что колеса робота и поверхность, по которой движется робот, не являются идеальными. Поэтому при выполнении эксперимента пройденное роботом расстояние может отличаться от расчетного. Особенно большие расхождения появятся при больших значениях скоростей, так как здесь уже будет играть свою роль инерция робота. Также большое значение на точность движения оказывает распределение веса

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.
4. проверьте самостоятельно: при каких значениях скоростей пройденный путь роботом наиболее близок к заданному в нашем блоке. Составьте таблицу с результатами вашего эксперимента

Лабораторная работа 13. Программирование прохождения лабиринта роботом

Цель работы: изучить

Методические рекомендации:

Одним из самых простых правил для прохождения лабиринта является правило "одной руки": двигаясь по лабиринту, надо все время касаться правой или левой рукой его стены. Этот алгоритм, вероятно, был известен еще древним грекам. Придется пройти долгий путь, заходя во все тупики, но в итоге цель будет достигнута.

Хотя у этого правила и есть один недостаток, но о нем мы поговорим позже.

Попробуем описать робота, действующего в соответствии с правилом "правой руки".

В начале своей работы робот должен найти стену, по которой он будет следовать. Для этого он может просто двигаться вперед, пока не упрется в преграду.

После того как робот наткнулся на препятствие, он начинает передвигаться в соответствии с правилом "правой руки".

Двигаясь вдоль стены, робот следит, есть ли проход справа. Если проход есть, робот должен идти по нему, чтобы не оторваться от стены справа.

Если прохода нет впереди стена робот поворачивает налево. Если прохода снова нет, он еще раз поворачивает налево, таким образом разворачиваясь на 180 градусов, и идет в обратном направлении.

Блок-схема алгоритма для робота, работающего по правилу "правой руки", представлена на следующем рисунке.

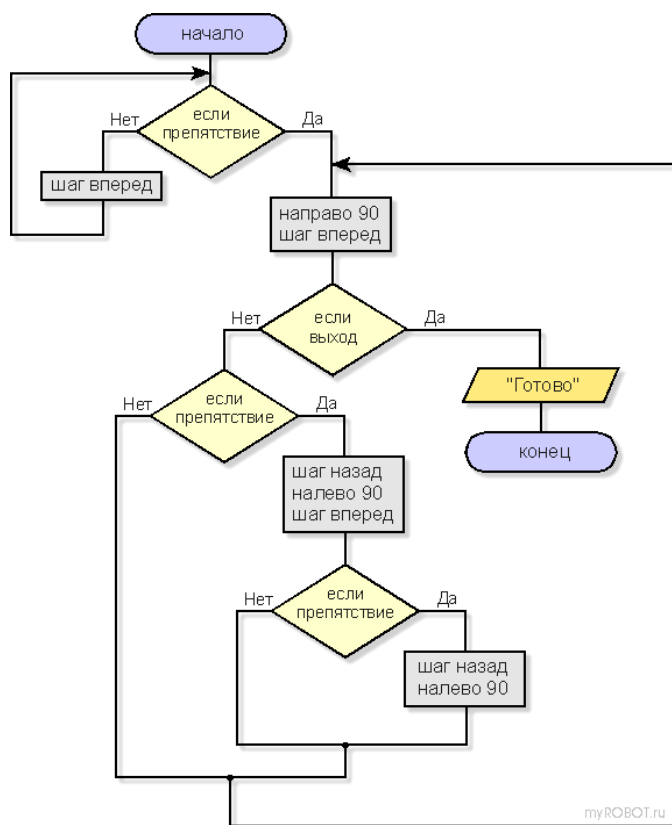


Рис. 1-Блок-схема алгоритма

Попробуем проверить работу данного алгоритма и напомним для него программу. Для этой цели обратимся к среде программирования [GameLogo](#). Эта среда является удобным средством для моделирования различных алгоритмов, связанных с управлением роботами. В ней есть исполнитель черепаха, который по своей сути является не чем иным, как самым настоящим роботом. Черепаха располагает очень удобным набором команд вперед, направо, налево, назад. Кроме того, в центре черепахи есть датчик, принимающий значение от 0 до 100, в зависимости от тона поверхности, на которой она находится.

Диалект языка Лого, который мы будем использовать, очень прост и похож на Basic. Познакомиться с командами языка можно [здесь](#). А бесплатно скачать среду программирования GameLogo [здесь](#). Размер дистрибутива невелик всего 1 Mb.

В самом начале программы дадим команду черепахе, чтобы она подняла перо (по умолчанию черепаха оставляет после себя след).

Размер поля 800 на 600 точек. Исходное положение для черепахи находится в месте с координатами 115, 545 (белый квадрат).

Цвет дорожек лабиринта светлый, на них датчик будет принимать значения больше 50. Цвет стен лабиринта темный, значение датчика будет меньше 50. Выход из лабиринта представлен черным квадратом, значение датчика над которым будет равно 0.

Объявим переменную флаг, с помощью которой будем контролировать, достигнут ли выход из лабиринта.

Напишем программу и запустим ее с помощью большой красной кнопки с надписью "Выполнить".

```

переменная флаг

фон = maze1.gif
    поднять перо
    место 115, 545

' поиск первой стены

повторять пока датчик > 50 {
    вперед 12
}

' правило правой руки

повторять пока флаг = 0 {

    направо 90
    вперед 12

    если датчик = 0 то

        флаг = 1

    иначе

        если датчик < 50 то

            назад 12
            налево 90
            вперед 12

            если датчик < 50 то
                назад 12
                налево 90
            конец условия

        конец условия

    конец условия
}

пиши "цель достигнута"

```

Если известно, что у лабиринта нет отдельно стоящих стенок, то есть нет замкнутых маршрутов, по которым можно возвращаться в исходную точку, то такой лабиринт называют односвязным и его всегда можно обойти полностью, применив правило "одной руки".

Если же лабиринт содержит отдельно стоящие стенки, то, применяя правило "одной руки", не всегда можно пройти все коридоры и тупики. Лабиринты с отдельно стоящими стенками и с замкнутыми маршрутами называются многосвязными. При этом многосвязные лабиринты можно разделить на две группы: без "петли" вокруг цели (замкнутый маршрут не проходит вокруг цели) и с замкнутой "петлей" вокруг цели (цель можно обойти по замкнутому маршруту).

В многосвязных лабиринтах второй группы правило "одной руки" не работает и, применяя его, достичь цели невозможно. Но и эти лабиринты можно пройти, полагаясь на точный алгоритм. Решение задачи о таких лабиринтах принадлежит сравнительно позднему времени, и начало ему положено Леонардом Эйлером. Эйлер не без оснований полагал, что выход из любого лабиринта может быть найден, и притом сравнительно простым путем. Универсальный алгоритм прохождения любых лабиринтов был описан только через столетие в книге французского математика Э. Люка "Recreations matematicques", изданной в 1882 году. Интересно, что Люка при

описании алгоритма указал на первенство другого французского математика М. Тремо. Таким образом, алгоритм стал известен как **алгоритм Люка-Тремо**.

Тремо предлагает следующие правила: выйдя из любой точки лабиринта, надо сделать отметку на его стене (крест) и двигаться в произвольном направлении до тупика или перекрестка; в первом случае вернуться назад, поставить второй крест, свидетельствующий, что путь пройден дважды туда и назад, и идти в направлении, не пройденном ни разу, или пройденном один раз; во втором идти по произвольному направлению, отмечая каждый перекресток на входе и на выходе одним крестом; если на перекресте один крест уже имеется, то следует идти новым путем, если нет то пройденным путем, отметив его вторым крестом.

Зная алгоритм Тремо, можно скорректировать поведение легендарного Тесея. Вдохновленный подарком любимой Ариадны, он уверенно идет по лабиринту. Вдруг перед ним возникает ход, по которому уже протянута нить... Что делать? Ни в коем случае не пересекать ее, а вернуться по уже известному пути, сдвигая нить, пока не найдется еще один непройденный ход.

Применив вариант алгоритма Тремо, отец теории информации Клод Шеннон (Claude Elwood Shannon) построил одного из первых самообучающихся роботов. Шеннон дал ему звучное имя "Тесей", но в истории "Тесей" стал больше известен как "мышь" Шеннона. "Мышь" сначала обследовала весь лабиринт, а затем (во второй раз) проходила весь путь значительно быстрее, избегая участков, пройденных дважды.

В наши дни роботы, проходящие лабиринт, являются участниками одного из самых интересных состязаний думающих машинок, которое проходит в нескольких странах мира. Эти соревнования носят общее название [Micromouse competition](#) и по своим техническим новациям относятся к лидерам робототехнического спорта.

На первой Российской Олимпиаде Роботов проводились соревнования, целью которых было прохождение своеобразного лабиринта: за наиболее короткое время, двигаясь через "открытые двери" в стенках, робот должен был добраться от места старта до места финиша. Контролировать свое движение робот мог по черным линиям, нанесенным на пол лабиринта.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

5. тему лабораторной работы;
6. условие задачи;
7. полный окончательный код примера; результаты ее тестирования.

Лабораторная работа 14. Программирование навигации робота

Цель работы: изучить основы программирования навигации робота.

Методические рекомендации:

Для успешной навигации в пространстве бортовая система робота должна уметь строить маршрут, управлять параметрами движения (задавать угол поворота колес и скорость их вращения), правильно интерпретировать сведения об окружающем мире, получаемые от датчиков, и постоянно отслеживать собственные координаты.

Схемы навигации автономных устройств

- глобальная – определение абсолютных координат
- устройства при движении по длинным маршрутам;
- локальная – определение координат устройства по отношению к некоторой (обычно стартовой) точке. Эта схема востребована разработчиками тактических беспилотных самолетов и наземных роботов, выполняющих миссии в пределах заранее известной области;
- персональная – позиционирование роботом частей своего тела и взаимодействие с близлежащими предметами, что актуально для устройств, снабженных манипуляторами.

Считается, что чем крупнее аппарат, тем выше для него важность глобальной навигации и ниже – персональной. У роботов-малышей все наоборот.

Системы навигации классифицируются еще по одному признаку – они могут быть пассивными и активными. Пассивная система навигации подразумевает прием информации о собственных координатах и других характеристиках своего движения от внешних источников, а активная рассчитана на определение местоположения только своими силами.

Как правило, все глобальные схемы навигации пассивные, локальные бывают и теми и другими, а персональные схемы – всегда активные.

Пассивные навигационные схемы

Глобальная спутниковая система GPS (ошибка в определении собственных координат не может превышать размера автономного аппарата (в противном случае возможны столкновения с устройствами такого же или меньшего размеров и другие конфликты со средой)

Радиомаяки (размещение в зоне действий робота источников радиосигналов, которые обрабатываются бортовым микропроцессором. Но так как радиомаяки располагаются в фиксированных точках некоторого маршрута, аппарат теряет возможность обходить препятствия или выбирать альтернативный путь движения

Активные навигационные схемы

инерционные навигационные системы (ИНС)

Механические и оптические гироскопы (измеряют усилие (момент внешней силы), прикладываемое к телу, на котором они размещены, и на этой основе определять положение тела относительно позиции, с которой началось движение, и его скорость)

Механические акселерометры (определяют собственное ускорение)

Одометр (периодически измеряет скорость вращения колеса и, так как диаметр последнего известен, определяет пройденный путь)

Лазерные дальномеры (LIDAR-

Генератор радио или каких-либо других (чаще всего ультразвуковых и инфракрасных) сигналов
стереоскопические видеосистемы;

Гибридные навигационные схемы

Гибридная бортовая система управления роботом использует навигационные средства всех видов, но занимается прежде всего оценкой окружающей обстановки, анализом выполняемого задания и принятием решений. Аппарат пытается построить собственный образ среды, в которой ему приходится действовать, после чего формирует маршрут и движется по нему, постоянно сопоставляя свою карту пространства с данными, полученными от устройств навигации.

Ведутся исследования по организации целенаправленного поведения коллективов роботов. Аппараты, действующие совместно в некоторой области, могут уточнять друг у друга свои координаты, сравнивать взаимное положение и таким образом быстро передвигаться.

Обнаружение и объезд препятствий. Для решения задач обнаружения и объезда препятствий используется пара камер, образующих стереосистему.

Сопоставляя идентичные элементы изображений от каждой из камер, алгоритм системы технического зрения синтезирует трехмерное изображение пространства перед камерами. Это позволяет построить карту глубины и рассчитать дальность до препятствий, попавших в поле зрения камер.

Дальность, на которой от стереопары можно получить достоверные данные определяется стереобазой и, применительно к роботу, составляет 4-5 метров. Увеличивая стереобазу и разрешение видеокамер системы технического зрения, можно значительно повысить точность и глубину синтезируемой карты диспаратности.

По мере своего продвижения робот выстраивает трехмерную карту окружающего пространства. По ней, собственно, прокладывается путь и планируется движение робота вычислителем автопилота.

Процесс навигации включает в себя следующие этапы:

- а) составление карты среды;
- б) коррекция траектории движения робота;
- в) планирование маршрута (выбор оптимального пути, ведущего к цели);
- г) управление локальными перемещениями;
- д) обход роботом опасных участков трассы.

Составление карты среды

Сначала формируется карта рабочей зоны робота, при этом внешняя среда дискретизируется, и каждому участку, содержащему препятствие, ставится в соответствие информация о типе этого препятствия.

Также предполагается, что для определения проходимости участков среды используется дистантная сенсорная подсистема сканирования на основе лазерного дальномера, а определение пройденного пути осуществляется с помощью одометрической подсистемы с применением метода пассивного колеса. Построение карты происходит одновременно с исследованием внешней среды

Участки с номерами 1-8 считаются потенциально проходимыми.

Для уточнения их проходимости робот осуществляет последовательное сканирование данных участков.

Сканирование начинается с участка номер 1.

Если данный участок свободен, то выполняется перемещение в среде в направлении центра первого участка как показано на рисунке . Причем величина перемещения равна максимальному из габаритных размеров участков, на которые дискретизируется среда.

После этого считается, что робот переместился в центр следующего свободного участка. При этом координаты центра данного участка наносятся на карту.

Также наряду с координатами центра каждому участку ставятся в соответствие индексы смещения по координатным осям X , Y относительно начального участка 0 .

Это делается для того, чтобы упорядочить хранение карты среды в памяти робота в соответствии с индексами участков. После того, как робот нанес информацию о первом участке на карту, он сканирует проходимость участков 2, 3, ..., 8 двигаясь против часовой стрелки по периметру участка 0, как показано на рисунке.

Если какой-то из данных участков оказывается занятым, то робот определяет, находится ли на нем препятствие либо цель и вносит информацию о данном участке на карту. После обследования участков 1-8 робот расширяет зону исследований и переходит к сканированию участков 9-23 и т.д. Данный процесс продолжается до тех пор, пока во внешней среде не останется ни одного неисследованного участка. После этого в памяти робота формируется карта его рабочей зоны.

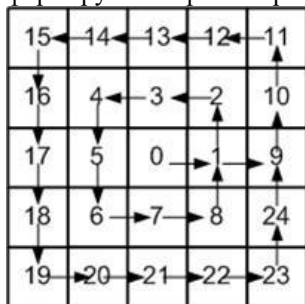


Рис.1 –Карта рабочей зоны

Исходя из этого, строим алгоритм составления карты местности:

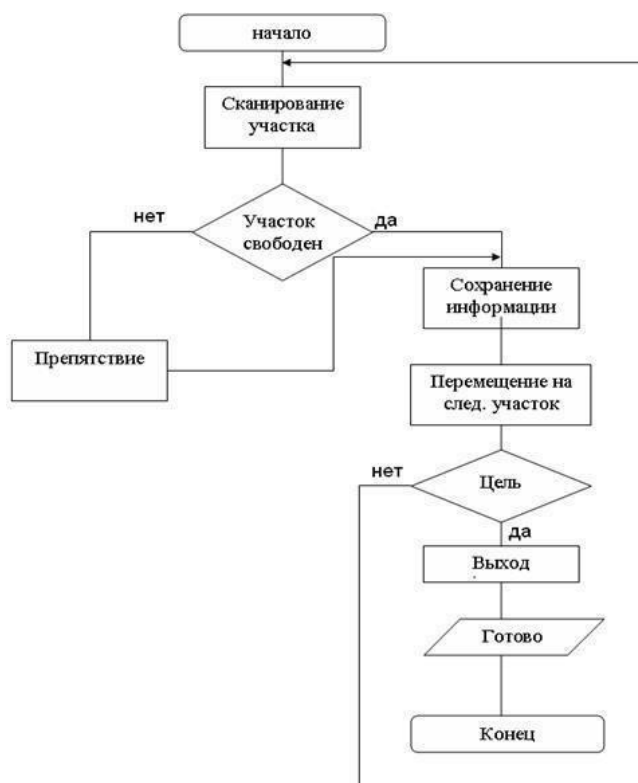


Рис. 2 Алгоритм составления карты местности

Результат:

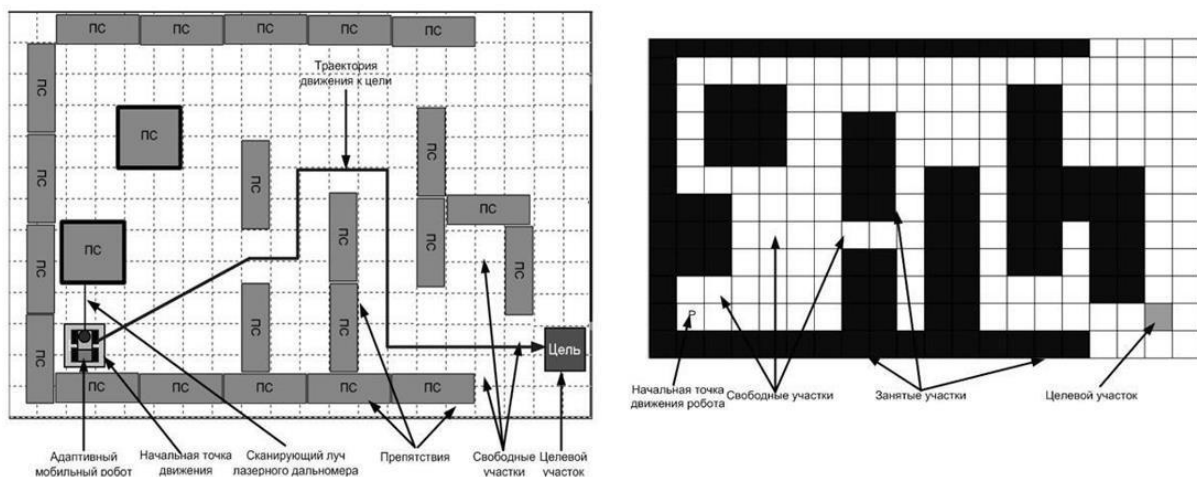


Рис. 3 – Навигационная карта робота

Система управления мобильным роботом должна решать следующие задачи:

- а) обработка сенсорных данных (в т.ч. данных от интерфейса с оператором) с целью сбора информации о роботе и внешней среде вокруг него;
- б) планирование мероприятий по уяснению целевого задания и планирование последовательности подзадач, необходимых для выполнения этого задания;
- в) формирование таких программных траекторий движения МР, которые бы приводили к выполнению роботом локальной подзадачи (например, прибытие к целевой точке в среде с препятствиями);
- г) формирование таких задающих воздействий на исполнительные механизмы робота, которые бы приводили к максимально точному и быстрому выполнению ими программной траектории движения.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Отчет о выполненной самостоятельной работе должен содержать:

1. тему лабораторной работы;
2. условие задачи;
3. полный окончательный код примера; результаты ее тестирования.

Учебно-методическое и информационное обеспечение дисциплины Перечень основной и дополнительной литературы, необходимой для

Перечень основной литературы:

- 1 Джонс, М. Х. Электроника практический курс / М. Джонс; пер. с англ. Е. В. Воронова, А. Л. Ларина. 2-е изд., испр. М.: Техносфера, 2013. 512 с. (Мир электроники). Прилож.: с. 478-497. Библиогр.: с. 498-499. Предм. указ.: с. 500-510. ISBN 978-5-94836-341-7
- 2 Майк, Предко. PIC-микроконтроллеры. Архитектура и программирование / Предко Майк: практическое пособие; пер. Ю.В. Мищенко Электронный ресурс. PIC-микроконтроллеры. Архитектура и программирование, 2019-04-19: Профобразование; Саратов, 2017. 512 с. Книга находится в базовой версии ЭБС IPRbooks. ISBN 978-5-4488-0062-7

Перечень дополнительной литературы:

- 1 Афонин, В. Л. Интеллектуальные робототехнические системы: учебное пособие / Афонин В. Л. Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. 222 с. Книга находится в базовой версии ЭБС IPRbooks. ISBN 5-9556-00024-8.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:

- 1 Программирование роботов. Среда программирования роботов Microsoft Robotics Studio:

<http://www.robotlive.ru/program/>.

2 Роботы и робототехника: <http://www.prorobot.ru/>.

3 Электронная библиотека СКФУ: <http://www.ncfu.ru/nzuchnaya-biblioteka-skfu.html>.

4 Электронно-библиотечная система "Университетская библиотека-онлайн": www.biblioclub.ru.

Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень программного обеспечения и информационных справочных систем:

1. Электронная библиотека СКФУ: <http://www.ncfu.ru/nzuchnaya-biblioteka-skfu.html>.

2. Электронно-библиотечная система "Университетская библиотека-онлайн": www.biblioclub.ru.

3. Роботы и робототехника: <http://www.prorobot.ru/>.

4. Программирование роботов. Среда программирования роботов Microsoft Robotics Studio:

<http://www.robotlive.ru/program/>.

Программное обеспечение:

1. Программная инструментальная среда NXC.

2. Программная инструментальная среда QReal.

3. Среда разработки LabView.

4. Среда разработки Robolab.

Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине:

1. Антропоморфный робот, модель AR-601E.

2. Лабораторный стенд "Мехатроника".

