И. И. Мацаль

Основы программирования в среде VEXcode V5

Учебно-методическое пособие

ЭКЗАМЕН ТЕХНОЛАБ

Издательство ЭКЗАМЕН®

МОСКВА
2020

VEX V5

УДК 372.8:004
ББК 32.816
М36

Мацаль И. И.

М36 Основы программирования в среде VEXcode V5: учебно-методическое пособие / И. И. Мацаль. — М. : Издательство «Экзамен», 2020. — 96 с.

ISBN 978-5-377-16442-5

Предлагаемое пособие содержит сведения о конструировании роботов, дает обзор основных алгоритмов управления. Оно поможет организовать проектную деятельность в области робототехники в образовательных организациях. VEXcode – это среда программирования, которая учитывает возрастные особенности учащихся. Интуитивно понятные вид и устройство VEXcode позволяет ученикам быстро и легко приступить к работе. Расширенный робототехнический набор для класса с модулем технического зрения, на базе контроллера VEX V5, позволяет участвовать в соревнованиях по робототехнике

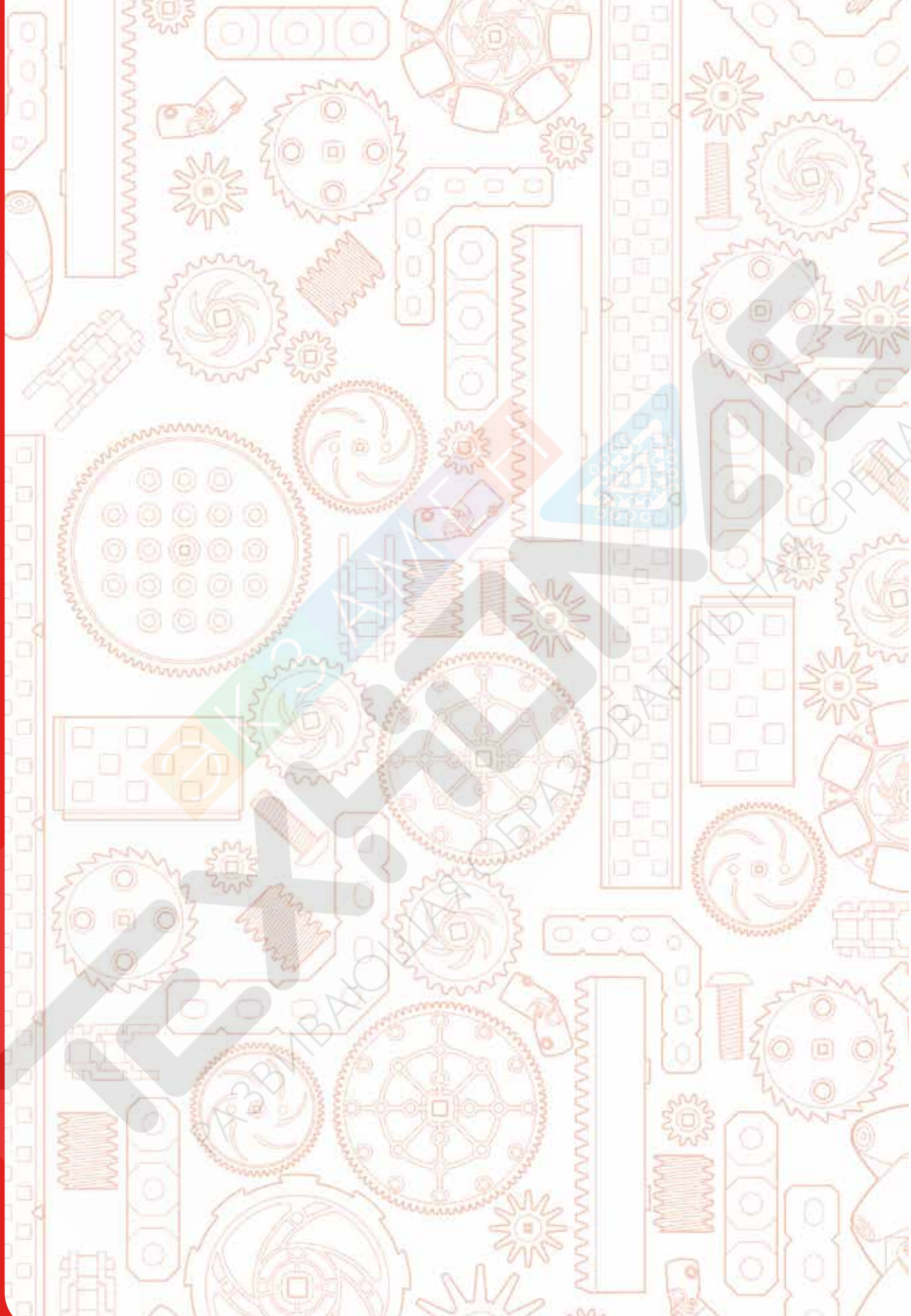
Пособие предназначено педагогам, учащимся основной и средней школы. Также оно будет интересно всем, кто интересуется и занимается робототехникой самостоятельно.

УДК 372.8:004
ББК 32.816

Подписано в печать с диапозитивов 16.09.2020.
Формат 60х90/8. Гарнитура «Calibri». Бумага офсетная.
Усл. печ. л. 12. Тираж 500 экз. Заказ №

ISBN 978-5-377-16442-5

Содержание



Конструктивные элементы и комплектующие конструкторов VEX

Отличительной особенностью наборов VEX является многообразие различных конструктивных элементов. Конструктивные элементы VEX представляют собой металлические детали и крепежные элементы, позволяющие разрабатывать на их базе сложные и прочные механизмы (Рисунок 1).

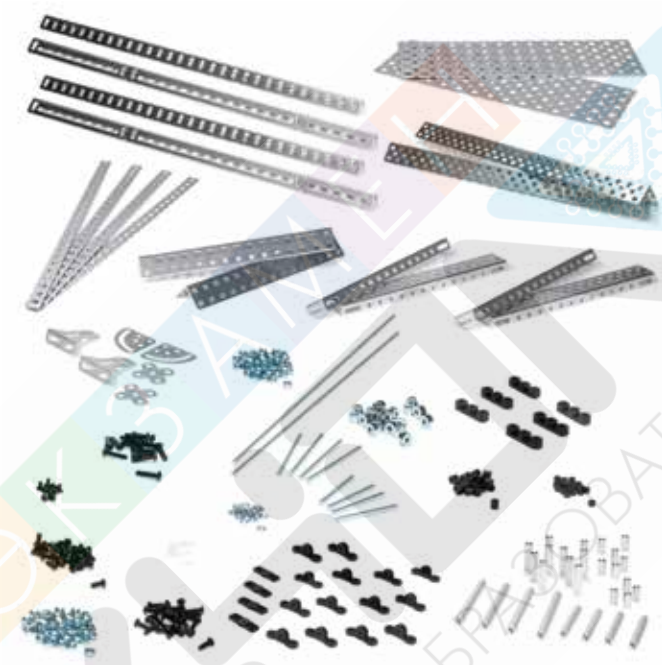


Рисунок 1. Конструктивные элементы VEX

Основными комплектующими VEX являются пластины и уголки из перфорированного алюминия. Благодаря наличию множества отверстий с равным шагом данные детали могут скрепляться друг с другом произвольным образом (Рисунок 2).

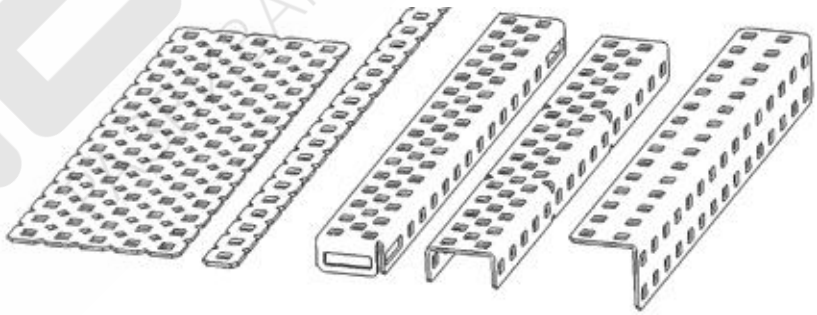


Рисунок 2. Пластины и уголки VEX

Отличительной особенностью деталей VEX является то, что все отверстия в них

квадратной формы. Благодаря этому становится возможным фиксировать положение различных элементов и деталей относительно друг друга (Рисунок 3).

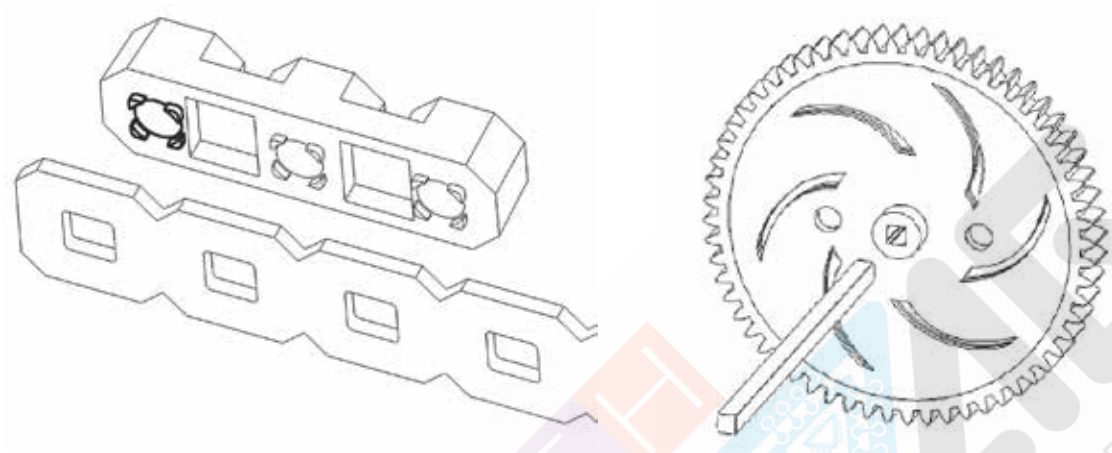


Рисунок 3. Способы соединения VEX

Фиксация элементов осуществляется за счет специальных бортиков, входящих в углы квадратных отверстий, тем самым задавая ориентацию деталей. В случае закрепления валов или осей квадратные детали прочно и надежно устанавливаются в соответствующие отверстия (Рисунок 4).

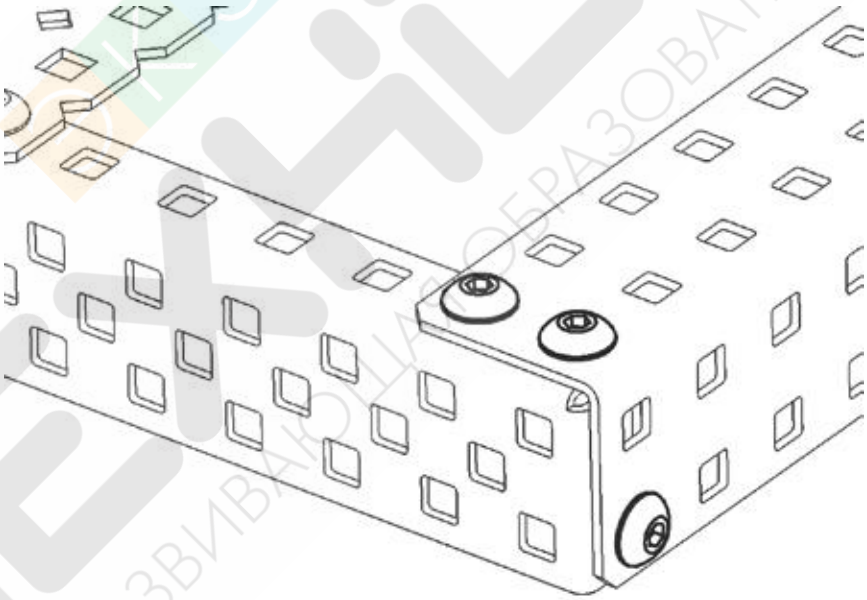


Рисунок 4. Фиксация элементов с помощью винтов и гаек

Соединение деталей между собой осуществляется с помощью резьбовых соединений на основе гаек и винтов различной длины. Все винты имеют шляпку с отверстием под шестигранный инструмент, входящий в робототехнический набор. Фиксация винтов осуществляется с помощью гаек различного типа, как обычных, так и стопорящих (Рисунок 5).

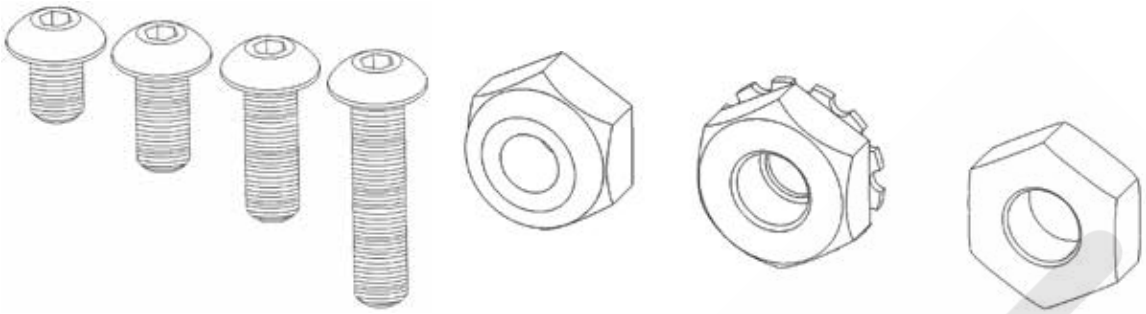


Рисунок 5. Элементы крепления (винты и гайки)

Также соединение некоторых комплектующих может осуществляться с помощью пластиковых заклепок (Рисунок 6).

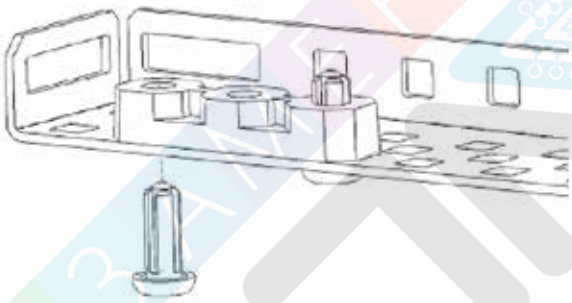


Рисунок 6. Использование пластиковых заклепок

Для закрепления различных конструкций или устройств могут применяться как стандартные детали и крепежные элементы, так и специализированные разделители. В робототехнический набор VEX входит комплект разделителей (Рисунок 7).

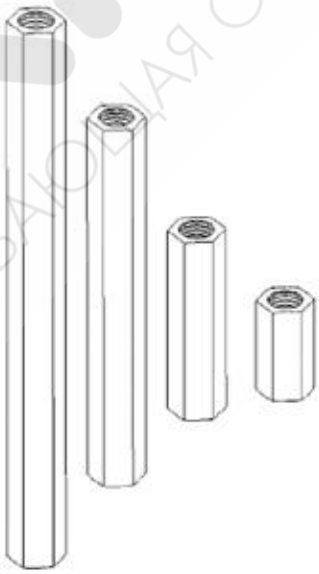


Рисунок 7. Комплект разделителей

Одной из ключевых особенностей деталей и комплектующих VEX является возможность согнуть или разрезать любую из плоских пластин вдоль специальных направляющих пазов. Благодаря этому можно создавать детали и конструктивные элементы специальной формы, удовлетворяющей проекту конструкции (Рисунок 8).

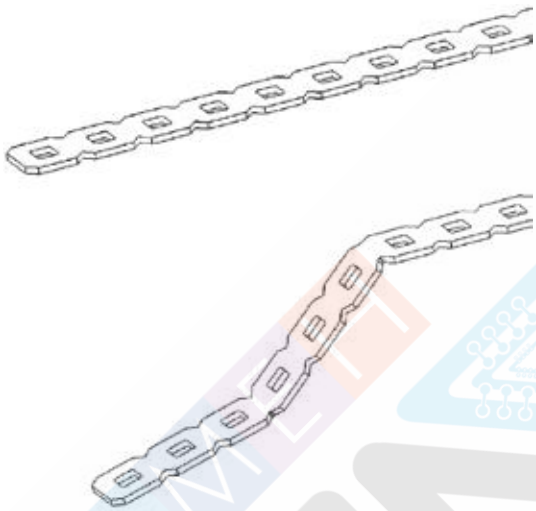


Рисунок 8. Изгиб пластины по направляющим пазам

С помощью различных компонентов и деталей можно сконструировать модели роботов различных габаритов и назначения. Соединяя детали между собой, можно создавать как статичные конструкции, так и подвижные механизмы (Рисунок 9).

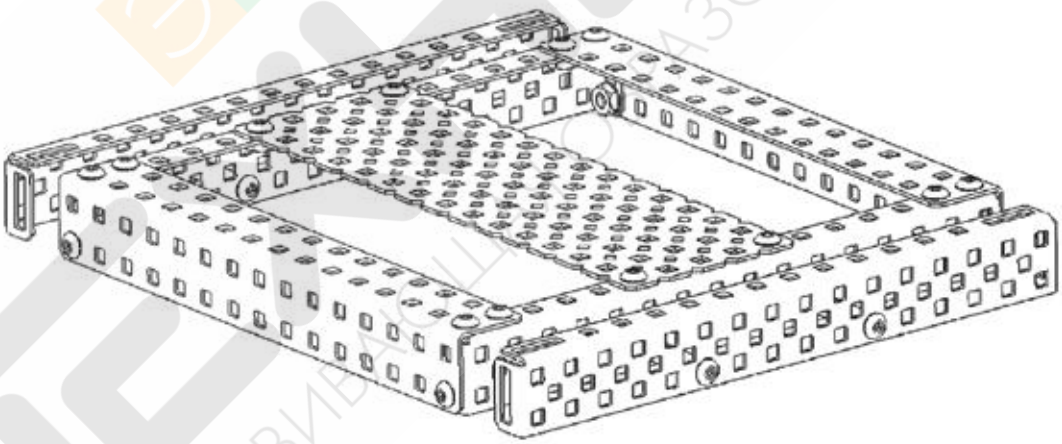


Рисунок 9. Статичная конструкция

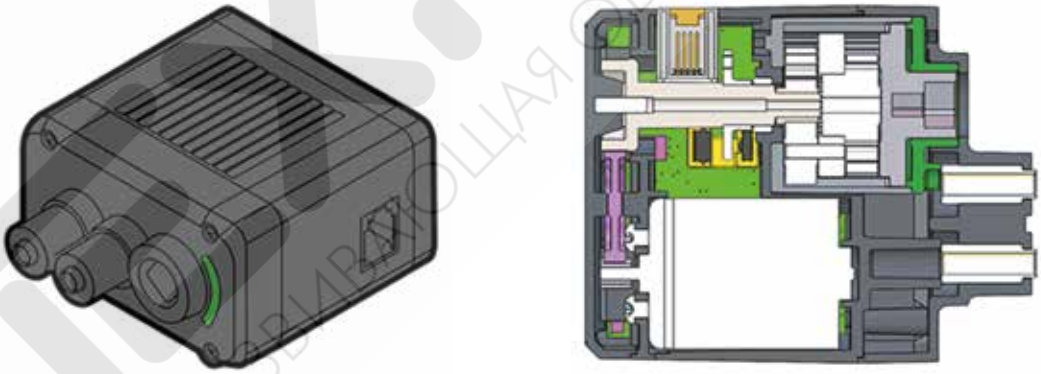
Примечание: при разработке конструкций роботов и прочих механизмов не следует забывать о жесткости конструкции в целом. Для упрочнения конструкции необходимо использовать как стандартные уголки и ребра жесткости, так и специализированные элементы.

Исполнительные механизмы конструкторов VEX

Разработка сложных робототехнических устройств и подвижных механизмов, помимо надежных конструктивных элементов, требует наличия специализированных исполнительных механизмов, таких как приводы, системы линейного перемещения и элементы зубчатых передач (Рисунок 10).

Рисунок 10. Зубчатые колеса

В базовый робототехнический набор VEX входят приводы и сервоприводы на базе двигателей постоянного тока. С помощью данных устройств можно разрабатывать подвижные механизмы и конструкции различного назначения. Каждый привод представляет собой электромеханическое устройство, состоящее из двигателя постоянного тока, редуктора и системы управления (Рисунок 11).



a *b*

Рисунок 11. Привод VEX V5 (a – внешний вид привода, b – привод в разрезе)

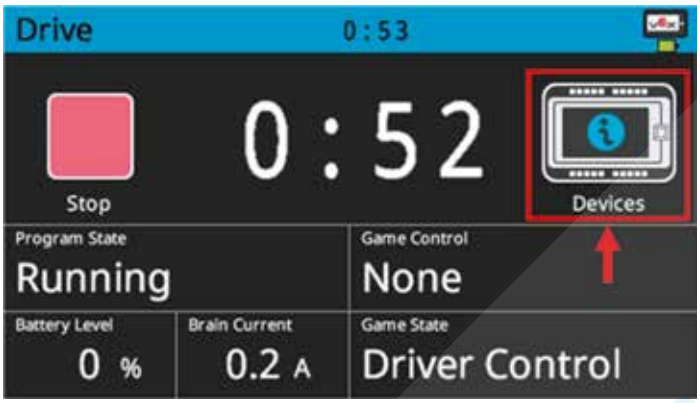
Конструкция каждого из приводов разборная, и пользователь в любой момент может внести изменения в его конструкцию, например изменить передаточное отношение, то есть заменить картридж редуктора (Приложение 1).

В наборах VEX V5 приводы обладают встроенными энкодерами и возможностью сообщать контроллеру V5 о ряде своих параметров, таких как температура, скорость и ряд других параметров. В таблице 1 приведены характеристики привода V5.

Таблица 1. Характеристики привода V5

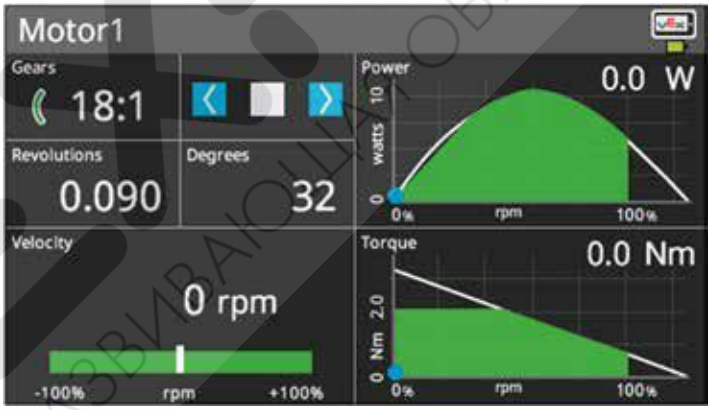
Параметр	Значение
Скорость (в зависимости от картриджа)	Примерно 100, 200 или 600 об/мин
Максимальная мощность	11 Вт
Постоянная мощность	11 Вт
Момент привода (с картриджем на 100 об/мин)	2,1 Н*м
Выходная мощность при низком уровне заряда аккумулятора	100% выходной мощности
Обратная связь	Положение Скорость (вычисляется) Ток Напряжение Мощность Момент (вычисляется) КПД (вычисляется) Температура
Разрешение энкодера	1800 тиков/оборот с передачей 36:1 шестерни 900 тиков/оборот с передачей 18:1 шестерни 300 тиков/оборот с передачей 6:1 шестерни
Размеры	57,3 мм W x 71,6 мм L x 33,0 мм H
Вес	155 грамм

Для того чтобы узнать текущие параметры привода нам, необходимо выполнить шаги, показанные на рисунке 12.



a

b



c

Рисунок 12. Получение параметров привода:
a – выбираем пункт Devices;
b – выбираем интересующий нас привод;
c – доступные параметры привода перед вами.

Привод VEX V5 обладает индикатором, по которому можно судить о текущем состоянии привода. В таблице 2 приведены возможные режимы индикации.

Таблица 2. Режимы работы индикатора

Состояние индикатора	Расшифровка состояния индикатора
Индикатор не активен	Нет соединения с включенным контроллером
Горит красный цвет	Создано соединение между контроллером и приводом
Быстро мигает красный цвет	Этот порт выбран на контроллере для отображения данных
Медленно мигает красный цвет	Ошибка соединения с контроллером

Как было сказано ранее, скорость, момент и мощность привода могут быть изменены с помощью замены картриджа редуктора (Рисунок 13).

a *b* *c*

Рисунок 13. Сменные картриджи редуктора

Конструкция приводов позволяет пользователю передавать с них вращение на удаленные механизмы с помощью валов различной длины. Вал может быть установлен напрямую в привод, а также может быть закреплен с помощью специальной муфты, ограничивающей передаваемый момент и препятствующей поломке привода или механизма в случае заклинивания.

Помимо сменных валов, приводы могут передавать вращения на различные механизмы, устанавливаемые на валы с помощью фиксирующих втулок. Таким образом, можно разрабатывать различные конструкции и механизмы, состоящие из валов и зубчатых передач (Рисунок 14).

Рисунок 14. Пример использования привода
(1 – привод VEX V5, 2 – зубчатая передача, состоящая из зубчатых колес 3, 4, 5)

Комбинируя зубчатые колеса различного диаметра, можно конструировать механизмы с различным передаточным отношением, тем самым изменяя скорость вращения и передаваемый ими момент (Рисунок 15).

Рисунок 15. Комбинация двух зубчатых колес

Используя различные зубчатые передачи, можно конструировать все возможные механизмы роботов – гусеничные и колесные шасси, поворотные основания и приводы качения и т.п.

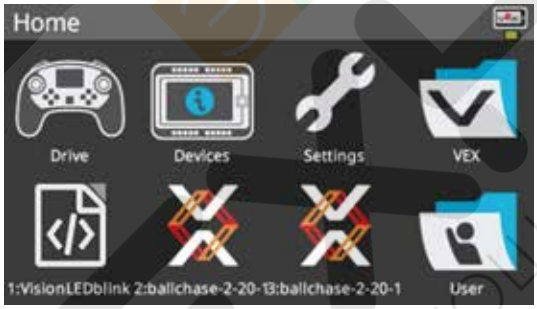
Комплектующие VEX позволяют конструировать различных роботов, решающих широкий спектр задач, начиная от образовательных, исследовательских и соревновательных, вплоть до прикладных задач, решаемых профессиональными роботами.

Электронные компоненты VEX V5

Одной из важных особенностей образовательных наборов VEX является наличие контроллера VEX V5 (Рисунок 16).

Рисунок 16. Контроллер VEX V5

Данный контроллер обладает возможностью подключения более чем 20 устройств с разным интерфейсом. Кроме того, благодаря наличию сенсорного экрана вы можете посмотреть доступные программы, режимы работы и параметры подключенных устройств. Для получения информации о контроллере необходимо выбрать соответствующий пункт меню (Рисунок 17).



a

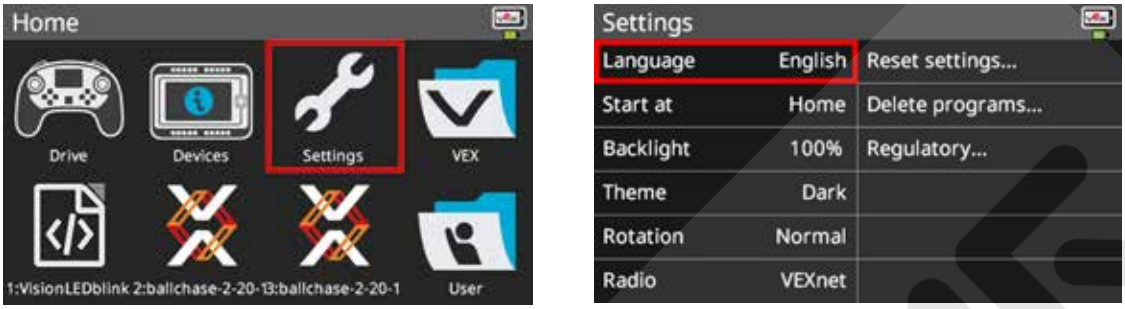
b

c

d

Рисунок 17. Шаги для получения информации об устройстве

Для того чтобы поменять язык интерфейса, вам необходимо зайти на контроллере в пункт «Settings» и в меню «Language» выбрать соответствующий язык (Рисунок 18).



a *b*

Рисунок 18. Изменение языка интерфейса контроллера

Контроллер обладает слотом для встраиваемой памяти (не входит в состав контроллера) для расширения возможностей хранения пользовательских программ и информации. Более подробные характеристики контроллера доступны в таблице 3.

Таблица 3. Характеристики контроллера VEX V5

Параметр	Значение
Количество портов для подключения приводов	Любой из 21-го порта мотора
Количество портов для подключения датчиков и радио модуля	Любой из 21-го порта мотора
Трехпроводные порты	8 штук
Характеристики основного процессора и сопроцессора VEXos	Процессор: один Cortex A9 – 667 МГц Сопроцессор: два Cortex M0 – 32 MHz
ОЗУ	128 Мбайт
Флеш-память	32 Мбайт
USB	2.0 (480 Mbit/s)
Цветной сенсорный экран	4.25”, 480 x 272 пикселей, 65000 цветов
Максимальный объем встраиваемой памяти (не входит в состав)	до 16 Гбайт, FAT32
Доступные беспроводные интерфейсы	VEXnet 3.0 и Bluetooth 4.2
Напряжение питания	12,8 В
Размер и вес	101,6 мм x 139,7 мм x 33,02 мм, 285 грамм

Помимо контроллера VEX V5 наборы могут включать в себя пульт управления, который позволяет реализовывать удаленное управление вашим роботом (Рисунок 19).

Рисунок 19. Пульт управления контроллером V5

Для зарядки пульта управления используется кабель USB-micro USB, входящий в состав набора. Обратите внимание на порядок создания соединения между контроллером и пультом управления, доступный в приложении 2.

Для калибровки пульта управления необходимо выбрать соответствующий пункт меню на экране пульта (Рисунок 20).

a *b*

Рисунок 20. Этапы калибровки пульта управления

Помимо вышеупомянутых электронных компонентов в набор входят элементы автономной навигации: датчики касания и камера. Датчики касания представляют собой две кнопки с двумя возможными состояниями: нажата или отжата.

Камера (Рисунок 21) представляет для нас больший интерес, так как хранит в себе целый ряд функций для успешной автономной работы, таких как: определение цвета и формы объектов, определение расположения объектов в относительной системе координат, определение количества объектов и других.

Рисунок 21. Камера VEX V5

Характеристики камеры приведены в таблице 4.

Таблица 4. Характеристики камеры VEX V5

Параметр	Значение
Частота кадров	50 кадров в секунду
Восприятие цветов	7 независимых объектов
Размер изображения	640 x 400 пикселей
Тип микроконтроллера	Dual ARM Cortex M4 и M0
Беспроводная связь	2.4 GHz 802.11 Wi-Fi Direct Hotspot с встроенным веб-сервером
Совместимость	Любое устройство с WiFi и браузером
Размер и вес	63,4 мм x 54 мм x 22,6 мм, 350 грамм

Пневматика

Линейка VEX включает в себя возможность использования пневматической линии (Рисунок 22), которая позволяет расширить функционал робота.

Рисунок 22. Элементы пневматической линии VEX

В приложении 3 находится инструкция по сборке пневматической линии. В нашем случае система будет состоять из резервуара с воздухом, регулятора воздуха и пневматического цилиндра. Пневматический цилиндр имеет всего два крайних положения, переход в которые осуществляется с помощью подачи воздуха в соответствующую камеру цилиндра (Рисунок 23).

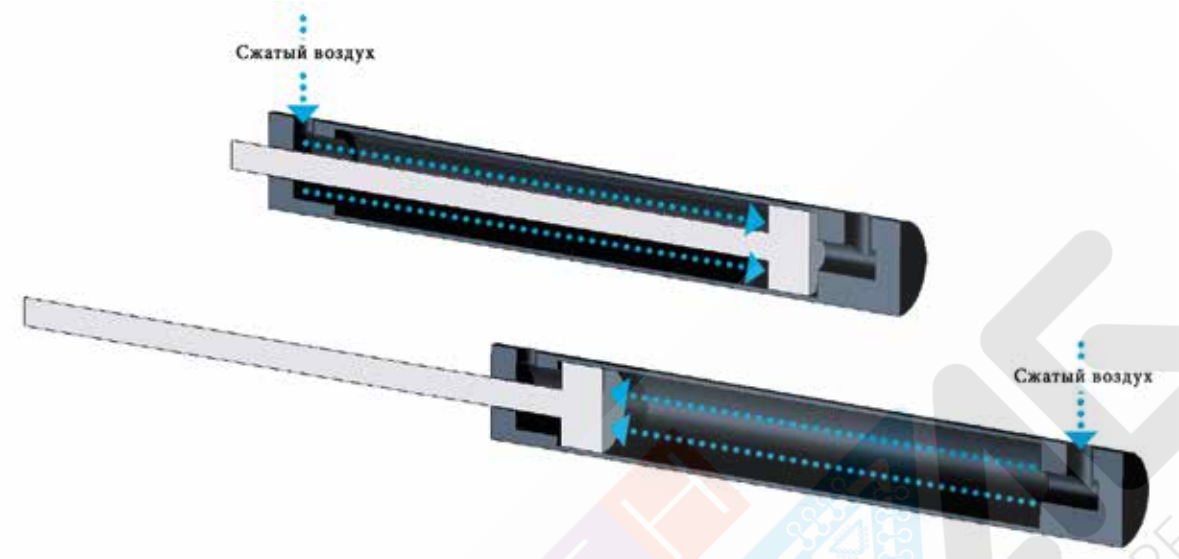


Рисунок 23. Принцип работы пневматического цилиндра

Используя пневматическую линию, вы можете создавать различные механизмы, схваты роботов, конвейерные линии и многое другое (Рисунок 24).

Рисунок 24. Пример использования пневматического цилиндра

Среда программирования VEXcode V5 Text

Для программирования контроллера V5 используется текстовая среда «VEXcode V5 Text». Скачивание доступно на российском сайте VEX (http://vex.examen-technolab.ru/vexedr/vexcode_v5). Интерфейс программы интуитивно понятен и обладает стандартными пунктами меню (Рисунок 25).

Рисунок 25. Интерфейс программы VEXcode V5 Text

Для предварительной настройки среды программирования вам необходимо зайти в меню «File» -> «Preferences». Перед вами откроется меню настройки (Рисунок 26).

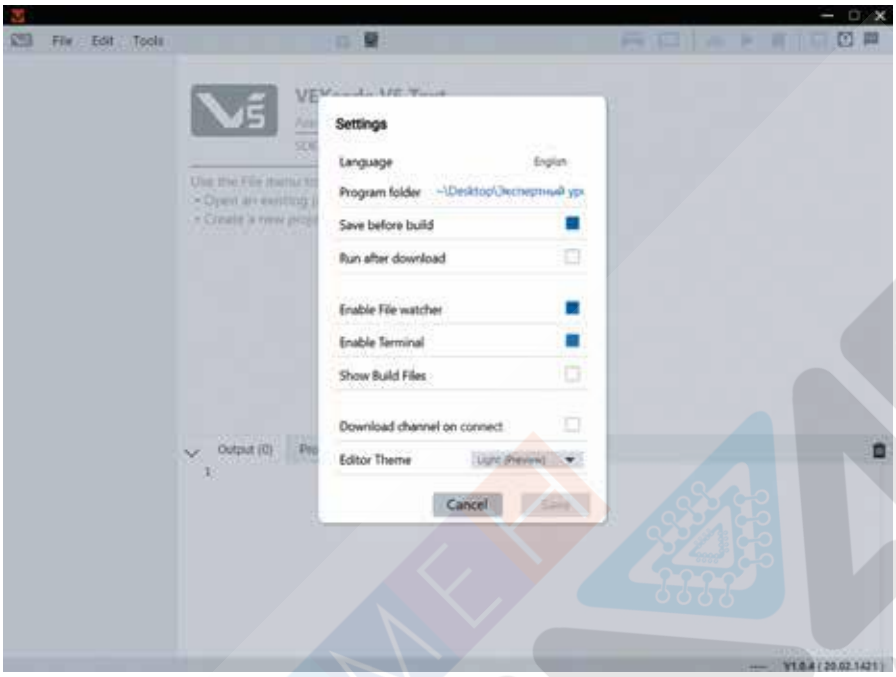


Рисунок 26. Меню настройки программы VEXcode V5 Text

Для создания новой программы вам необходимо зайти в меню «File» и далее выбрать пункт «New». Программа предложит вам дать имя будущему проекту, после чего перед вами откроется окно программы (Рисунок 27).

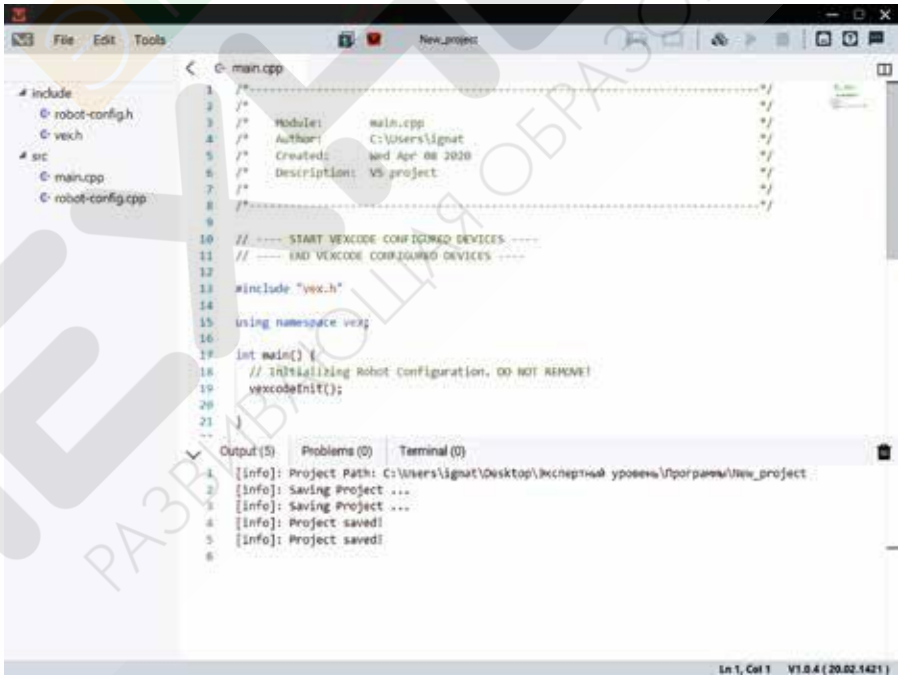


Рисунок 27. Стандартные поля после создания новой программы

Слева от рабочей области программы показаны все сопутствующие файлы вашего проекта: библиотеки с расширением <.h> и файлы с кодом программы <.cpp> (Рисунок 28).

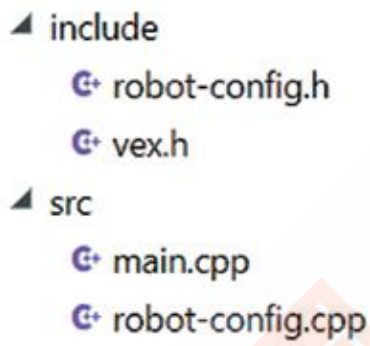


Рисунок 28. Файлы проекта

Настройка подключенных к контроллеру устройств проводится с помощью меню «Robot Configuration», доступного в правом верхнем углу окна программы (Рисунок 29).

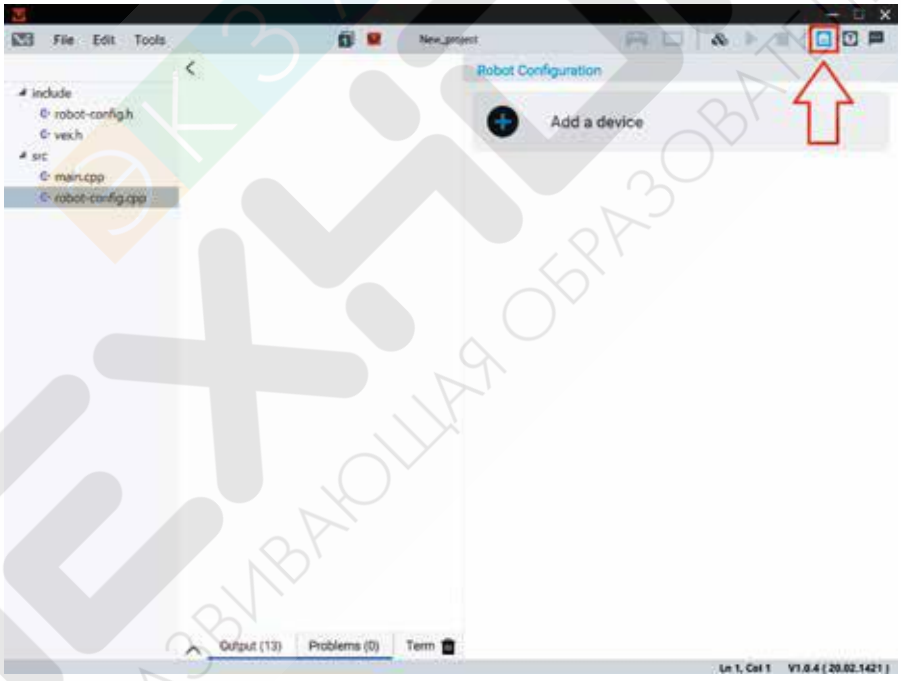


Рисунок 29. Меню настройки устройств робота

Кроме того, вы можете посмотреть информацию о подключенном контроллере, обновить контроллер и подключенные к нему устройства (Рисунок 30).

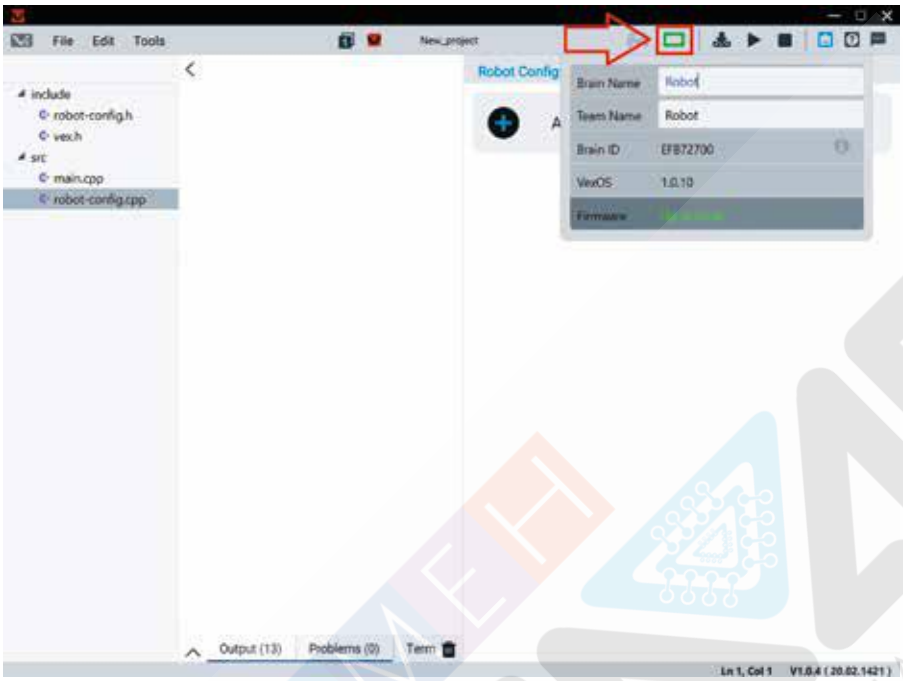


Рисунок 30. Просмотр сведений о контроллере

Для корректной работы и отображения данных вашей программы также рекомендуется настроить стандартный язык отображения (английский) для программ, не поддерживающих русский язык (Рисунок 31).

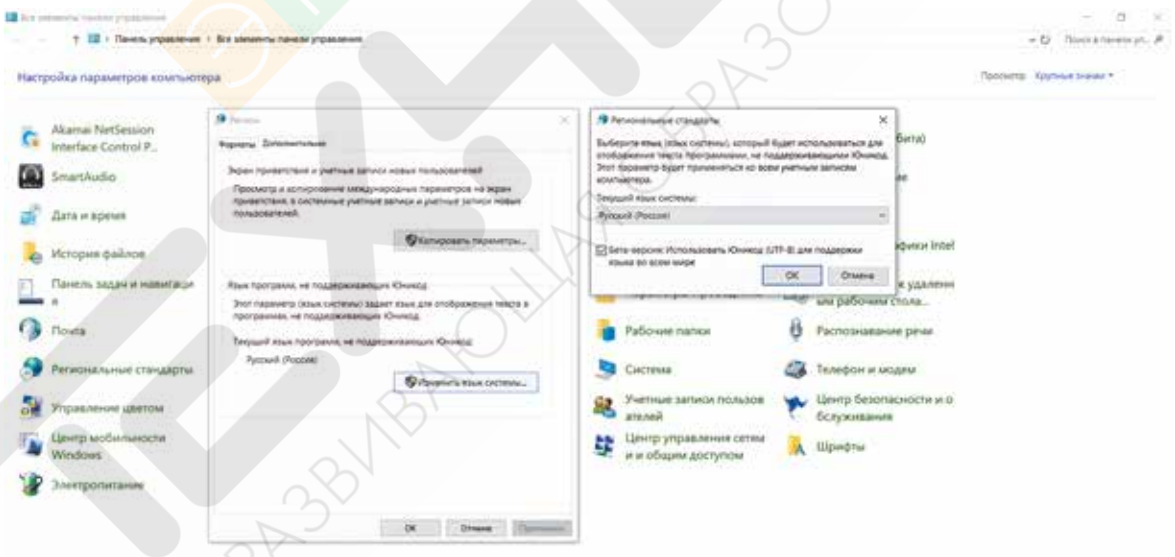


Рисунок 31. Настройка языка системы

Итак, мы познакомились с основными элементами из наборов VEX. Самое время перейти к программированию роботов.

В качестве базовой модели робота мы будем использовать колесную платформу с датчиками (SpeedBot). Инструкцию по сборке вы можете найти в приложении 4. Создайте новый проект нажатием кнопки «New» в меню файл. Перед вами откроется шаблон кода программы. Кроме того, вы можете заметить, что помимо файла «main.cpp» доступны еще три файла: robot-config.cpp, vex.h и robot-config.h. Файлы robot-config.cpp и robot-config.h отвечают за настройку всех устройств робота и по умолчанию недоступны для записи. Для того чтобы изменять данные файлы, вам необходимо активировать режим «Эксперт». Активация режима происходит установкой галочки рядом с названием режима (Рисунок 32).

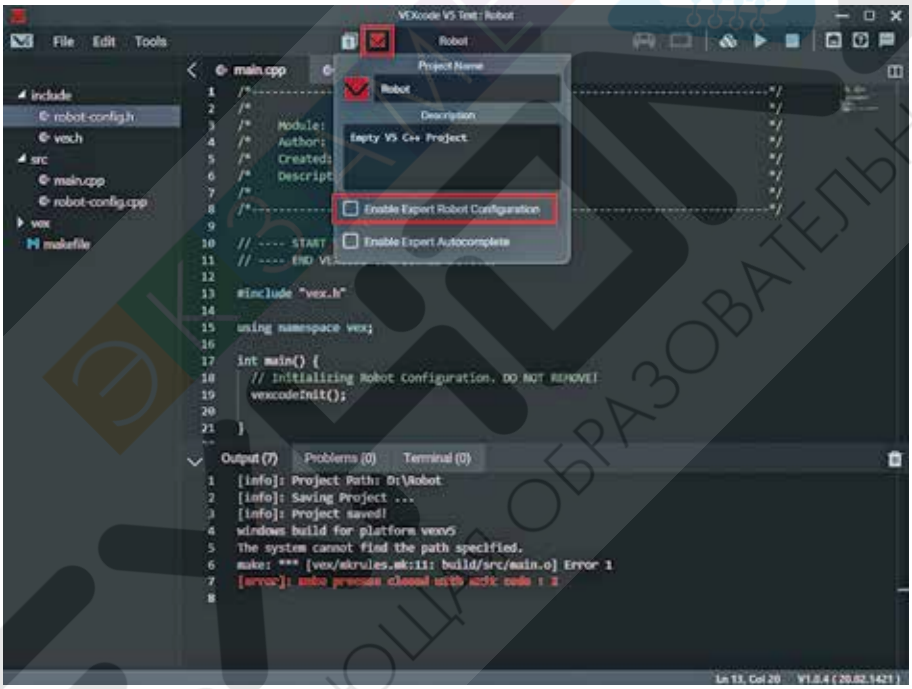


Рисунок 32. Активация режима эксперта

Разберем структуру файла main.cpp.

```
#include «vex.h»
```

Данная строка подключает файл библиотеки, содержащей функции для управления устройствами робота.

```
using namespace vex;
```

Среда VEXcode использует объектно-ориентированное программирование (ООП) в лице языка C++. Данная строка подключает пространство имен VEX, которое содержит классы, связанные с устройствами VEX V5.

int main() – функция, отвечающая за управление всей программой. Без функции main будет невозможно использование остальных функций. Ее объявление обязательно!

vexcodeInit() – инициализирует конфигурацию вашего робота (определяет для контроллера все подключенные устройства). Присутствие данной функции в main.cpp обязательно!

Мы разобрались с файлом main.cpp. Теперь настроим конфигурацию подключенных к нашему роботу устройств, а именно – приводов. Для этого нам необходимо перейти в визуальное меню настройки устройств – Robot Configuration. Данное меню доступно при отключенном режиме эксперта. Правый привод подключен к порту номер десять (10), а левый – к порту номер (1). Кроме того, к порту шесть (6) подключен радиомодуль, но он инициализируется автоматически и не требует специальной настройки. Рассмотрим настройку левого привода (Рисунок 33).

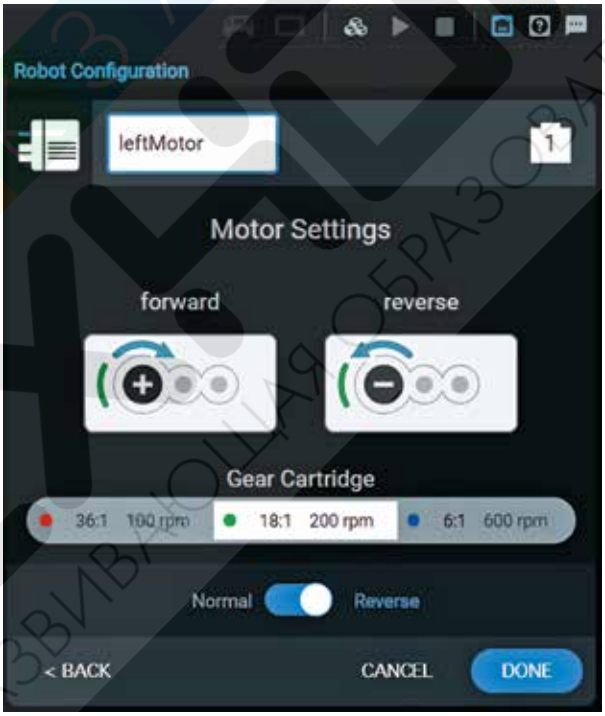


Рисунок 33. Настройка левого привода

Прежде всего необходимо указать имя привода, по которому вы будете к нему обращаться. Обратите внимание на разрешенные символы в имени: буквы, цифры и знаки подчеркивания, то есть логика построения имени точно такая же, как и в случае имен переменных. Не забудьте реверсировать привод: это необходимо для того, что-

бы изменить направление по умолчанию. Если вы сравните положение привода на роботе с изображением без установки флажка «Reverse» (Рисунок 34), то увидите, что для движения вперед необходимо добавлять к скорости левого привода знак минус.

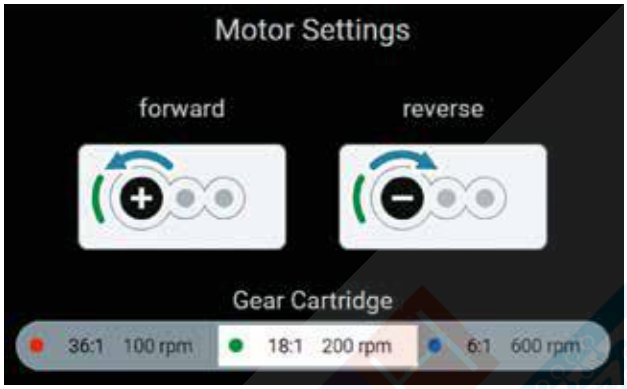


Рисунок 34. Направления положительного и отрицательного вращений привода без установки флажка Reverse

По умолчанию установлен картридж зеленого цвета. Подтвердите настройки привода с помощью нажатия кнопки Done. Настройте аналогичным образом правый привод робота (Рисунок 35).

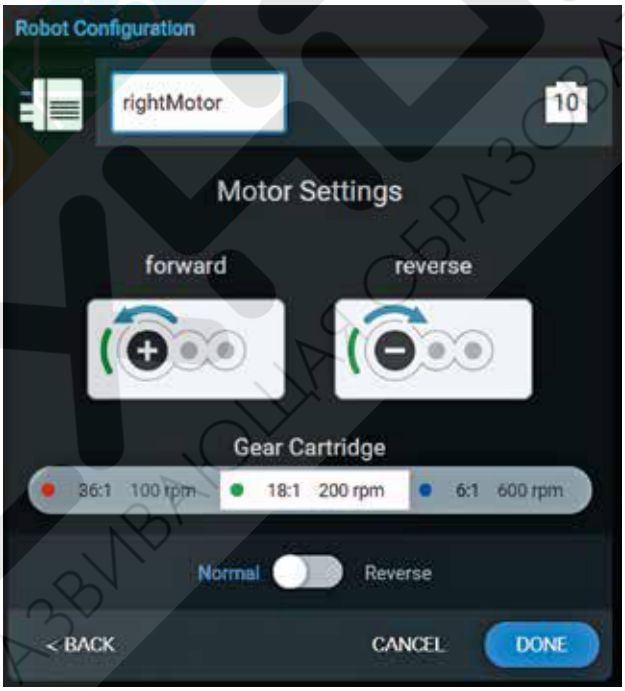


Рисунок 35. Настройка правого привода

Перейдем к файлу robot-config.cpp.

```
#include «vex.h»

using namespace vex;
using signature = vision::signature;
using code = vision::code;
```

Данные строки частично повторяются в файле main.cpp. Строки, содержащие слово vision, инициализируют настройки камеры.

```
brain  Brain;
```

Данная строка инициализирует функции вывода на экран (объект Brain).

```
motor leftMotor = motor(PORT1, ratio18_1, true);
motor rightMotor = motor(PORT10, ratio18_1, false);
```

В режиме эксперта вы можете самостоятельно инициализировать устройства. Давайте разберем функцию инициализации привода. В общем виде она будет иметь следующую структуру:

motor(<номер порта подключения>, <тип картриджа>, <true – реверс привода>)

```
void vexcodeInit( void ) {
}
```

В данном разделе инициализируются все необходимые компоненты для работы контроллера.

Перейдем к написанию программы движения нашего робота.

Запуск приводов

Для начала напишем простейшую программу для движения вперед в течение двух секунд и рассмотрим входящие в нее функции. Написание программы происходит в файле main.cpp

```
#include «vex.h»

using namespace vex;

int main() {

  vexcodeInit();

  leftMotor.setVelocity(20, percent);
  rightMotor.setVelocity(20, percent);

  leftMotor.spin(forward);
  rightMotor.spin(forward);

  wait(2, seconds);

  leftMotor.stop();
  rightMotor.stop();

}
```

Вызов функций по работе с устройствами осуществляется с помощью обращения к объектам, отвечающим за то или иное устройство.

<объект>.<функция для работы с объектом>

```
leftMotor.setVelocity(20, percent);
rightMotor.setVelocity(20, percent);
```

Данные функции устанавливают процент скорости привода. В общем виде функция будет выглядеть следующим образом:

<имя привода>. setVelocity(<значение скорости>, <единица измерения: percent – проценты, rpm – обороты в минуту>);

```
leftMotor.spin(forward);
rightMotor.spin(forward);
```

Данные функции определяют направление вращения привода. В общем виде функция будет выглядеть следующим образом:

<имя привода>. spin (<направление вращения: forward – положительное направление вращения, reverse – отрицательное направление вращения>);

```
wait(2, seconds);
```

Функция «удерживает» выполнение предыдущих функций и структур. После истечения времени, указанного в скобках, управление передается следующим структурам. В общем виде функция будет выглядеть следующим образом:

wait(<значение>, <единица измерения: sec – секунды, msec – миллисекунды>);

```
leftMotor.stop();
rightMotor.stop();
```

Данная функция отвечает за остановку указанного привода. В общем виде функция будет выглядеть следующим образом:

<имя привода>.stop()

Задачи для самостоятельного решения:

1. Напишите программу для поворота направо в течение двух секунд.
2. Напишите программу для движения робота вперед в течение двух секунд со скоростью 100 об./мин.

Ограничение максимального момента привода

Иногда бывает необходимо ограничить момент на выходном валу привода для защиты привода от выхода из строя или для защиты объектов от чрезмерного приложения силы роботом. В VEXcode V5 существует функция для установки максимального момента на нужном приводе. В общем виде функция будет выглядеть следующим образом:

<имя привода>.setMaxTorque(<значение>, <единица измерения: percent – проценты>)

Давайте проведем небольшой эксперимент. Для этого нам необходим схват робо-

та с установленным приводом (Рисунок 36).

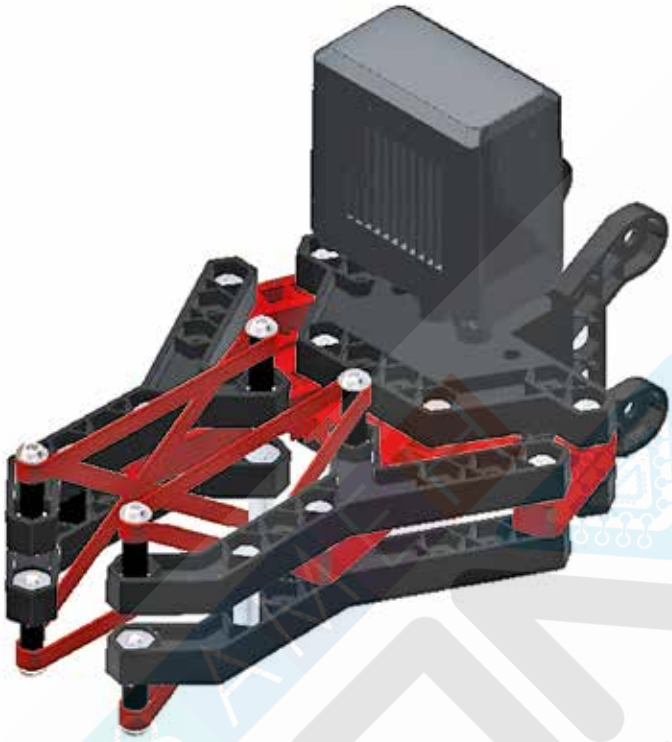


Рисунок 6. Схват робота с установленным приводом

Подсоедините привод к порту подключения под номером 2 (Рисунок 37).

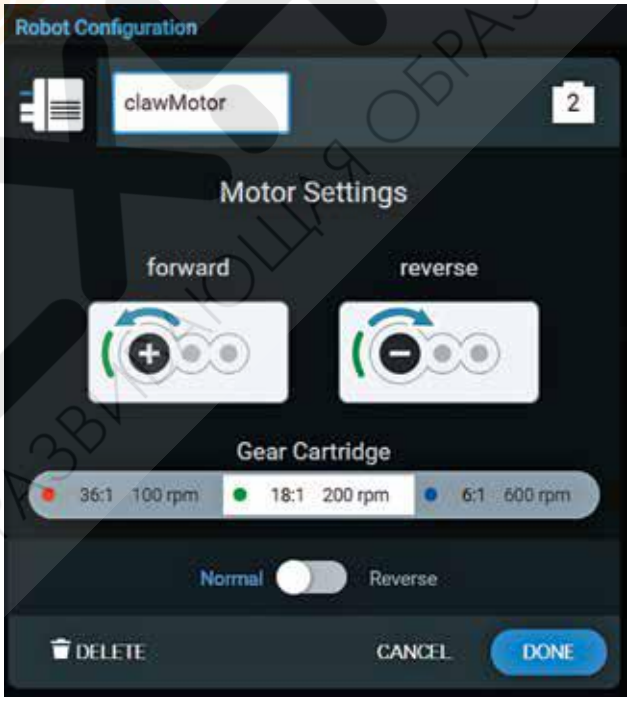


Рисунок 37. Настройки привода схвата

Раскройте схват вручную и поставьте по центру любой нежесткий объект (например, резиновую втулку или пластиковую бутылку с водой). Разберем следующий код программы.

```
#include «vex.h»

using namespace vex;

int main()
{
vexcodeInit();
    clawMotor.setVelocity(20, percent);

    for(int torque = 5; torque < 100; torque += 5)
    {
            clawMotor.setMaxTorque(torque, percent);
            clawMotor.spin(forward);
            wait(1, sec);
    }

clawMotor.stop();

}
```

Для нас представляет интерес конструкция **for**, благодаря которой мы будем постепенно увеличивать момент привода: создается переменная шага **torque**, далее указывается условие выхода из цикла (в нашем случае пока значение момента не достигнет максимального значения) и в конце указывается шаг, с которым цикл будет работать. Функция **setMaxTorque** ограничивает максимальный момент привода. В общем виде функция будет выглядеть следующим образом:

<имя привода>. setMaxTorque(<значение>, <единицы измерения: percent – проценты>);

Запустите программу и обратите внимание на степень деформации вашего объекта в зависимости от момента. Как вы можете заметить, деформация увеличивается с увеличением момента. Данную функцию можно использовать при разработке автономных систем погрузки/разгрузки объектов, имеющих повышенную хрупкость.

Использование таймера

Иногда необходимо досрочно прерывать работу тех или иных устройств. Это может

быть связано как с необходимостью уложиться в определенное время по регламентам соревнований, так и с конструктивными особенностями вашего робота. В VEXcode V5 существует специальная функция, позволяющая работать со встроенным в контроллер таймером. В общем виде функция будет выглядеть следующим образом:

<имя привода>.setTimeout(<значение>, <единица измерения: sec – секунды, msec – миллисекунды>);

Время отсчитывается с момента вызова функции. В контексте VEXcode данная конструкция используется для прерывания функций, связанных с установкой привода в определенное положение. Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;

int main()
{
  vexcodeInit();

  leftMotor.setVelocity(10, percent);
  leftMotor.setTimeout(1, sec);
  leftMotor.setPosition(360, degrees);
}
```

Функция **setPosition** поворачивает привод относительно начального положения на 360 градусов. В общем виде функция будет выглядеть следующим образом:

<имя привода>.setPosition(<значение>, <единица измерения: degrees – градусы, turns – обороты>);

Программа работает следующим образом: если привод за одну секунду не успеет достичь значения в 360 градусов, то он останавливается принудительно функцией **setTimeout**. Пришло время познакомиться со встроенными в приводы энкодерами.

Программирование энкодеров

Энкодеры позволяют роботу совершать точные движения его составных частей: поднимать/опускать груз на необходимую высоту, проезжать точное расстояние и т. д. Именно поэтому большинство робототехнических систем используют этот датчик. В предыдущем разделе мы уже использовали функцию **setPosition**, которая использует энкодеры для реализации поворота на определенный угол. Разберем вопрос, связан-

ный с движением на определенное расстояние.

Для того чтобы передвигаться на определенное расстояние, нам необходимо знать следующие входные параметры: диаметр колеса *d* (или длину окружности колеса), необходимое расстояние *S*. Логика нахождения расстояния без рассмотрения противодействующих сил (например, трения) довольно проста: нам необходимо узнать, сколько длин колес помещается в отрезке величиной *S* (Рисунок 38).

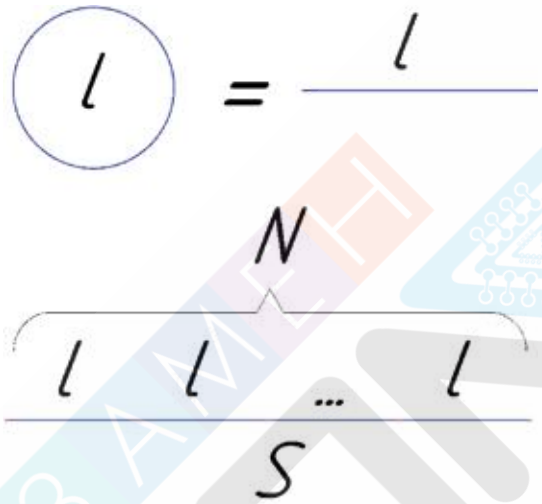


Рисунок 38. Постановка задачи

Для того чтобы робот проехал требуемое расстояние, нам необходимо узнать количество оборотов, которое совершает колесо. Для нахождения числа оборотов воспользуемся формулой:

$$N = \frac{S}{\pi * d}$$

Знаменатель представляет собой длину окружности колеса. Код программы будет выглядеть следующим образом:

```
#include «vex.h»

using namespace vex;

int main()
{
  vexcodeInit();
  const float l = 3.14 * 10;
  float S = 50;
  float N = S / l;

  leftMotor.setVelocity(10, percent);
  rightMotor.setVelocity(10, percent);

  leftMotor.setPosition(0, degrees);
  rightMotor.setPosition(0, degrees);

  while(leftMotor.position(turns) < N)
  {
    leftMotor.spin(forward);
    rightMotor.spin(forward);
  }

  leftMotor.stop();
  rightMotor.stop();
}
```

В данной программе мы будем использовать переменные для хранения входных параметров. Обратите внимание, что все переменные относятся к вещественному типу данных. Перед использованием энкодера его необходимо обнулить, то есть установить счетчик оборотов на нуль. Для установки энкодера на нуль используется функция setPosition. В общем виде функция будет выглядеть следующим образом:

<имя привода>.setPosition(<значение>, <единицы измерения: turns – обороты, degrees - градусы>);

Помимо этого, в программе используется функция **position**, которая получает число оборотов, сделанных роботом, с начала работы. В общем виде функция будет выглядеть следующим образом:

<имя привода>.position(<единицы измерения: turns – обороты, degrees - градусы>);

Цикл **while** будет работать до тех пор, пока текущее значение энкодера не сравня-

ется с расчетным. После цикла приводы необходимо остановить.

Задача для самостоятельного решения:
Напишите программу для объезда роботом квадрата со стороной 20 см.

Режимы торможения приводов VEX V5

Прежде чем перейти к рассмотрению программирования других устройств и систем, входящих в набор, нам необходимо обратить внимание на режимы работы наших приводов. Для задания режима работы привода существует функция **setStopping**. В общем виде функция будет выглядеть следующим образом:

<имя привода>. setStopping (<режим работы: brake, coast, hold>);

Режим торможения brake

Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;
int main()
{
  vexcodeInit();

  leftMotor.setVelocity(60, percent);
  rightMotor.setVelocity(60, percent);

  leftMotor.setStopping(brake);
  rightMotor.setStopping(brake);

  leftMotor.spin(forward);
  rightMotor.spin(forward);

  wait(2, seconds);

  leftMotor.stop();
  rightMotor.stop();

}
```

Что происходит с показателями привода (Рисунок 39)?

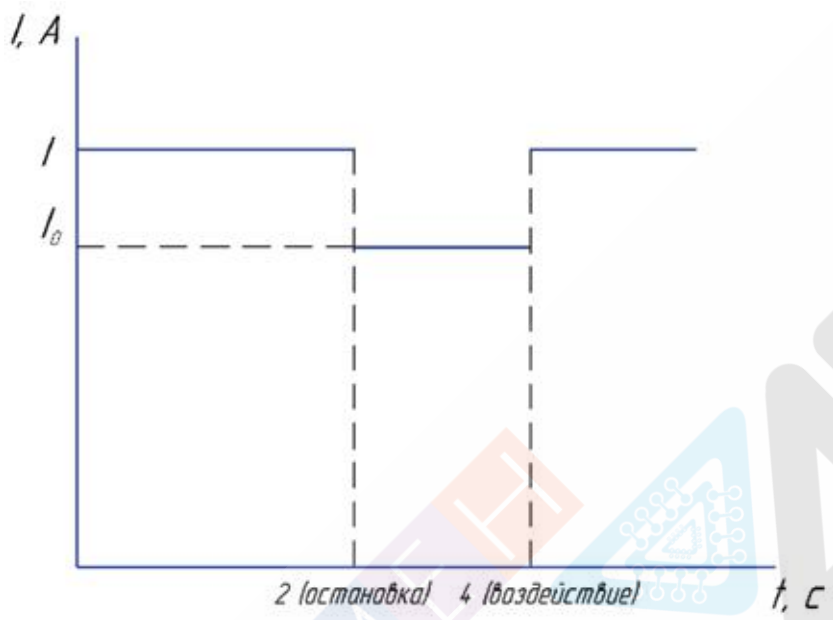


Рисунок 39. График зависимости I(t) в режиме brake

При запуске программы ток системы будет равен определенному значению (I_0), которое необходимо для поддержания работоспособности всей системы. При запуске привода ток возрастает до значения *I*, необходимого для работы привода. После остановки ток возвращается к изначальному значению I_0. В случае внешнего воздействия на любое из колес ток снова увеличится до значения *I*.

Изменение положения привода описывается следующим графиком (Рисунок 40).

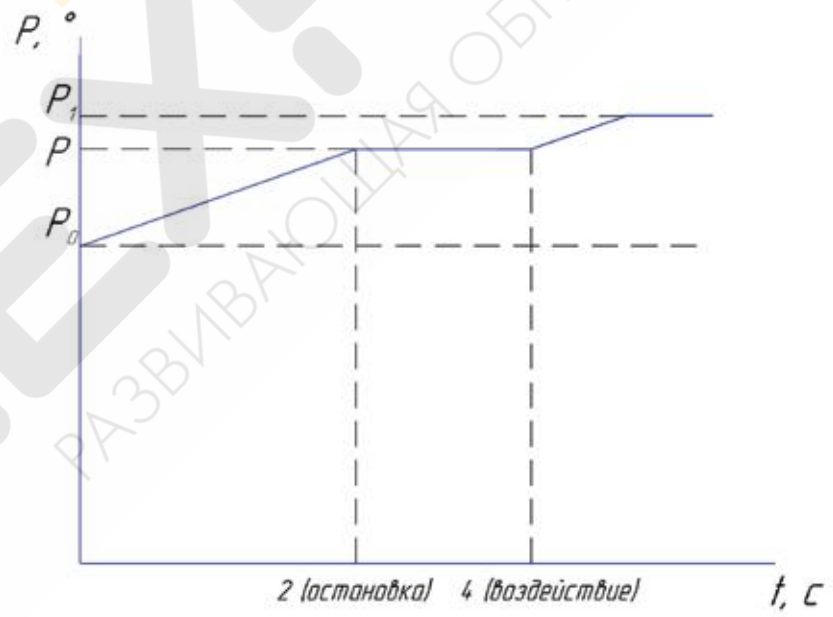


Рисунок 40. График зависимости P(t) в режиме brake

Привод будет «сопротивляться» изменению текущего положения выходного вала, но в случае изменения положения запишет его как текущее.

Режим торможения coast

Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;

int main()
{
  vexcodeInit();

  leftMotor.setVelocity(60, percent);
  rightMotor.setVelocity(60, percent);

  leftMotor.setStopping(coast);
  rightMotor.setStopping(coast);

  leftMotor.spin(forward);
  rightMotor.spin(forward);

  wait(2, seconds);

  leftMotor.stop();
  rightMotor.stop();

}
```

Отличие от режима **brake** заключается в изменении силы тока после остановки (Рисунок 41).

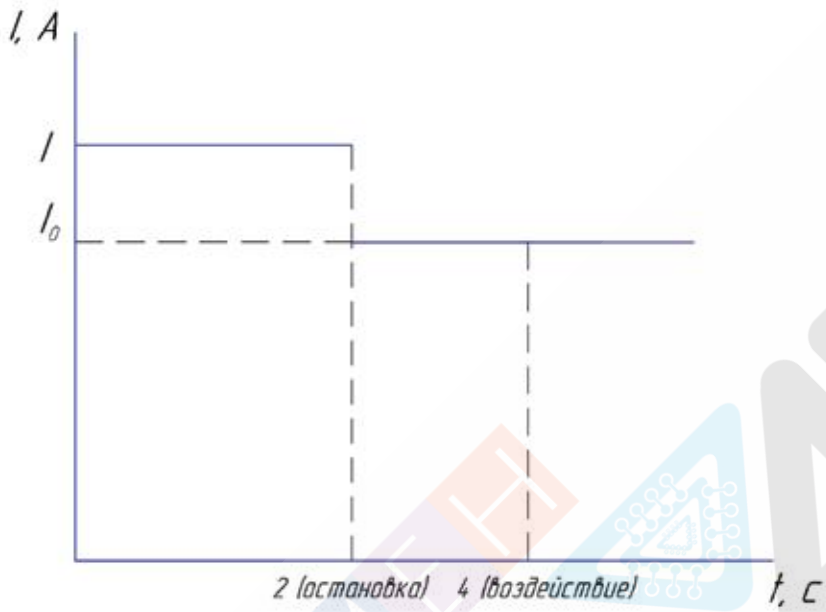


Рисунок 41. График зависимости I(t) в режиме coast

В режиме **coast** сила тока после остановки снижается до значения тока системы и вне зависимости от внешнего воздействия на привод остается равной значению силы тока системы. График для положения привода будет совпадать с графиком режима **brake** (Рисунок 42).

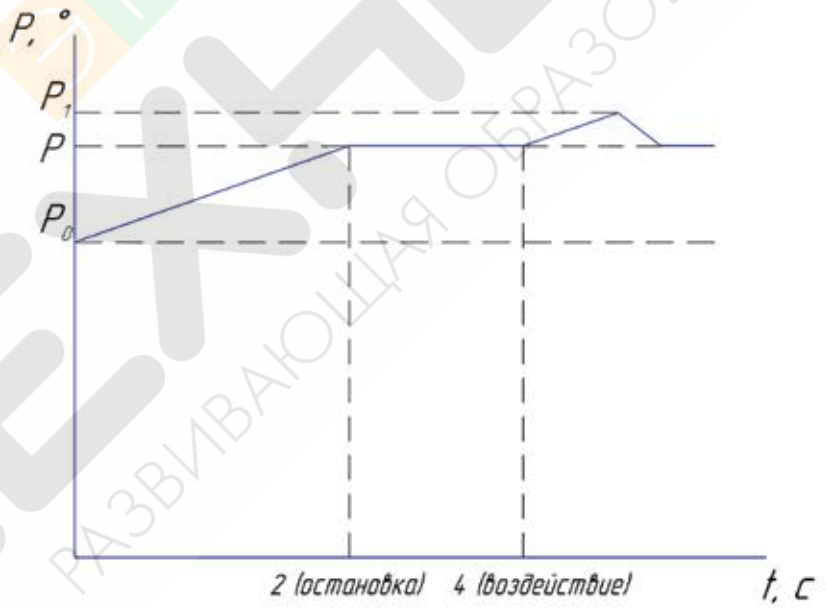


Рисунок 42. График зависимости P(t) в режиме coast

Режим торможения hold

Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;

int main() {

  vexcodeInit();

  leftMotor.setVelocity(60, percent);
  rightMotor.setVelocity(60, percent);

  leftMotor.setStopping(hold);
  rightMotor.setStopping(hold);

  leftMotor.spin(forward);
  rightMotor.spin(forward);

  wait(2, seconds);

  leftMotor.stop();
  rightMotor.stop();

}
```

В режиме **hold** сила тока после остановки вернется в значение I_1 (Рисунок 43), так как встроенному энкодеру необходимо постоянно отслеживать свое положение. Ток, необходимый для работы энкодера, будет равен: (I_1 - I_0).

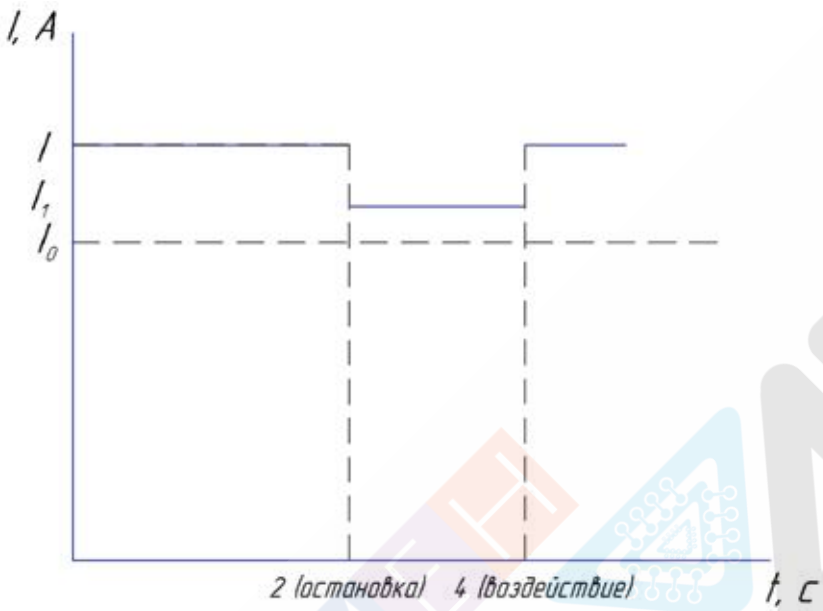


Рисунок 43. График зависимости I(t) в режиме hold

После остановки привода энкодер запоминает положение выходного вала после остановки (*P*) и, в случае внешнего воздействия на вал, возвращает вал в положение *P* (Рисунок 44).

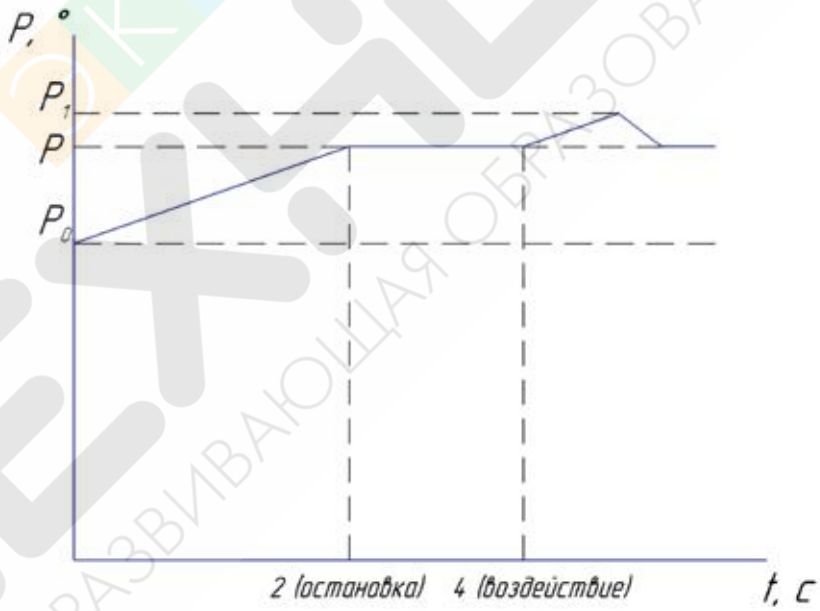


Рисунок 44. График зависимости P(t) в режиме hold

Работа с сенсорным экраном контроллера VEX V5

Контроллер VEX V5 обладает сенсорным дисплеем (Рисунок 45), который сильно расширяет функционал набора, позволяя отображать данные, получаемые с приводов и датчиков.

Рисунок 45. Сенсорный экран контроллера VEX V5

Помимо этого, мы можем использовать его в качестве координатной плоскости или устройства вывода текстовой или графической информации. Точка с координатами (*0, 0*) находится в левом верхнем углу контроллера (Рисунок 46).

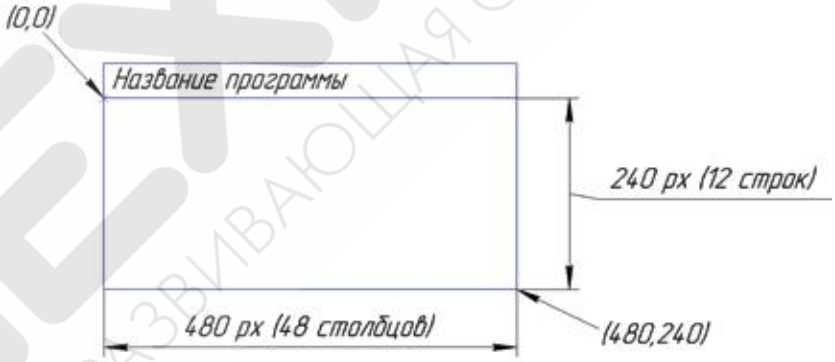


Рисунок 46. Характеристики сенсорного экрана

Обратите внимание, что ось *X* направлена вправо, а ось *Y* – вниз.

Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;

int main() {

 vexcodeInit();

 Brain.Screen.setFillColor(red);
 Brain.Screen.setPenColor(yellow);
 Brain.Screen.setPenWidth(3);
 Brain.Screen.setFont(monoM);

 while (1) {
  Brain.Screen.setCursor(1, 24);
  Brain.Screen.print(«time»);
  Brain.Screen.setCursor(6, 1);
  Brain.Screen.print(«velocity»);
  Brain.Timer.reset();
  int x[] = {0, 0};
  int y[] = {0, 0};

  for (int i = 0; i < 51; i++) {

   leftMotor.setVelocity(i, percent);
   rightMotor.setVelocity(i, percent);

   leftMotor.spin(forward);
   rightMotor.spin(forward);

   x[1] = 10 * Brain.Timer.value();
   y[1] = 2 * i;

   Brain.Screen.drawLine(x[0], y[0], x[1], y[1]);

   x[0] = x[1];
   y[0] = y[1];
   wait(200, msec);

   if (i % 10 == 0) {
    Brain.Screen.drawCircle(x[1], y[1], 5);
   }

  }
```

```
    leftMotor.stop();
    rightMotor.stop();
    Brain.Screen.setCursor(6, 18);
    Brain.Screen.print(«Press to restart»);

    while (!Brain.Screen.pressing()) {
    }
    Brain.Screen.clearScreen();
  }
}
```

В чем заключается данная программа? У робота постепенно увеличивается скорость колес и параллельно с этим на экране контроллера появляется график зависимости скорости от времени.

Будем разбираться по порядку.

```
Brain.Screen.setFillColor(red);
Brain.Screen.setPenColor(yellow);
Brain.Screen.setPenWidth(3);
Brain.Screen.setFont(monoM);
```

Данный отрывок кода содержит сразу четыре новые функции. Функция **setFillColor** отвечает за установку цвета заливки геометрических и текстовых объектов. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.setFillColor(<цвет заливки>);

Функция **setPenColor** отвечает за установку цвета текста, обводки и линий. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.setPenColor(<цвет заливки>);

Функция **setPenWidth** устанавливает толщину линий и обводки. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.setPenColor(<толщина линий в пикселях>);

Функция **setFont** устанавливает шрифт. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.setFont(<название шрифта>);

Итак, мы подготовили контроллер к дальнейшей работе. После запуска бесконеч-

ного цикла нам необходимо подписать координатные оси. Для этого первым деломнеобходимо установить курсор в нужную точку экрана. Установка курсора производится с помощью функции **setCursor**. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.setCursor(<номер строки>, <номер столбца>);

После перевода курсора в нужную позицию нам необходимо ввести название осей. Ввод текста производится с помощью функции **print**. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.print(«<текст>»);

Перед тем как перейти к непосредственному построению графика, нам необходимо обнулить таймер контроллера для того, чтобы построение графика начиналось строго с нуля. Для этого используем функцию reset для объекта Timer. В общем виде функция будет выглядеть следующим образом:

Brain.Timer.reset();

Для хранения координат точек нашего графика мы будем использовать два массива: *x* и *y*. Всегда помните о том, что нумерация элементов массива начинается с нуля! В общем виде объявление массива будет выглядеть следующим образом:

<тип данных: int, float, bool и т.д.> <название массива>[<размер массива>] = {<элементы массива>}

В нашем случае для построения графиков мы будем применять линейную интерполяцию, которая заключается в представлении графика в качестве набора отрезков. Так как для задания отрезка необходимо всего лишь две координаты: начало и конец, то наши массивы будут содержать всего по два элемента.

Запустим цикл **for**, в скобках которого объявим переменную *i*. Переменная будет отвечать за текущую скорость привода. В качестве условия возьмем ограничение скорости привода (то есть переменной *i*) до 50% включительно. Первым делом запустим наши приводы с помощью уже изученной функции **spin**, в скобках которой укажем в качестве параметра скорости переменную *i*.

```
x[1] = 10 * Brain.Timer.value();
y[1] = 2 * i;
```

В этом отрывке кода мы меняем второй элемент массива, который будет соответствовать конечной точке отрезка. Напоминаем, что в данном случае координата *x* будет соответствовать времени, а координата *y* – скорости привода. Коэффициенты перед значениями таймера и переменной *i* нужны для увеличения масштаба графика.

Функция **value** получает текущее значение таймера. В общем виде функция будет выглядеть следующим образом:

```
Brain.Timer.value();
```

После того как мы определили координаты отрезка, необходимо построить первый отрезок нашего графика. Построение линий осуществляется с помощью функции **drawLine**. В общем виде функция будет выглядеть следующим образом:

```
Brain.Screen.drawLine(<координата по x первой точки>, (<координата по y первой точки>, <координата по x второй точки>, <координата по y второй точки>);
```

Проведя линию, нам необходимо переназначить первые элементы массивов, так как именно с последней координаты текущего отрезка начнется построение следующего отрезка. Кроме того, добавим на график каждые 10% скорости промежуточные точки, представленные в виде окружностей. Для построения окружностей используется функция **drawCircle**. В общем виде функция будет выглядеть следующим образом:

```
Brain.Screen.drawCircle(<координата центра по x>, <координата центра по y>, <радиус окружности в пикселях>);
```

В данной программе также используется небольшая задержка в виде функции **wait** для более наглядного построения графика параллельно с работой привода.

После выполнения цикла **for** программа будет ожидать касания экрана. Обработчик события касания вызывается с помощью функции **pressing**. В общем виде функция будет выглядеть следующим образом:

```
Brain.Screen.pressing()
```

Функция возвращает значение **true** в случае касания экрана и **false** в случае некасания экрана. После того как пользователь коснется экрана, произойдет очистка экрана с помощью функции **clearScreen**. В общем виде функция будет выглядеть следующим образом:

```
Brain.Screen.clearScreen();
```

Произойдет очистка экрана и бесконечный цикл будет запущен еще раз.

Обработчик касаний экрана

Помимо рисования на экране вы также можете использовать его для запуска различных подразделов вашей программы. Рассмотрим следующий код:

```
#include «vex.h»

using namespace vex;

int main() {

 vexcodeInit();

 Brain.Screen.setPenColor(red);
 Brain.Screen.setPenWidth(4);
 Brain.Screen.drawLine(240, 1, 240, 240);
 Brain.Screen.setFillColor(blue);
 Brain.Screen.setPenColor(white);
 Brain.Screen.drawRectangle(1, 1, 238, 240);
 Brain.Screen.setFillColor(green);
 Brain.Screen.drawRectangle(242, 1, 238, 240);
 Brain.Screen.drawCircle(1,242,10);

 while(1)
 {
  while(!Brain.Screen.pressing())
  {

  }
  wait(1, sec);

  if(Brain.Screen.xPosition() < 238)
  {
   leftMotor.spinToPosition(360,degrees);
  }
  else if(Brain.Screen.xPosition() > 242)
  {
   rightMotor.spinToPosition(360,degrees);
  }
  else
  {

  }
  leftMotor.setPosition(0, degrees);
  rightMotor.setPosition(0, degrees);
 }

}
```

Программа заключается в том, что в случае нажатия на синий прямоугольник робот поворачивается направо, а на зеленый – налево. В начале нашей программы нам необходимо нарисовать оба прямоугольника. Для рисования прямоугольников используется функция **drawRectangle**. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.drawRectangle(<координата X левой верхней вершины>, <координата Y левой верхней вершины>, <ширина по X пикселях>, <высота по Y пикселях>);

После подготовки экрана мы запускаем бесконечный цикл, в котором первым делом ожидаем нажатия экрана. Нажав на экран, контроллеру необходимо определить, на какой именно прямоугольник вы нажали. Для этого используется функция **xPosition**, которая определяет координату по *X* последнего касания экрана. В общем виде функция будет выглядеть следующим образом:

Brain.Screen.xPosition()

В зависимости от места касания мы запускаем правое или левое колеса.

Задача для самостоятельного решения:
Экран состоит из четырех частей:
- при нажатии на левую верхнюю часть робот едет вперед пять секунд;
- при нажатии на левую нижнюю часть робот едет назад шесть секунд и на экране показывается график зависимости скорости от времени;
- при нажатии на правую верхнюю часть робот начинает поворачивать направо до тех пор, пока не нажата права нижняя часть.

Программирование датчика расстояния

Наборы VEX V5 могут содержать ряд так называемых «наследуемых» датчиков, одним из которых является датчик расстояния (Рисунок 47).

Рисунок 47. Датчик расстояния

Установите датчик произвольным образом на заднем бампере вашего робота, используя крепежные элементы из набора. Для добавления датчика к вашей программе вам необходимо выбрать в пункте меню «Robot Configuration» раздел «3-wire» (Рисунок 48).

Рисунок 48. Меню выбора трехпроводных датчиков

После того как вы выбрали датчик расстояния из раскрывающегося списка, вам будет предложено выбрать один из восьми портов подключения устройства (Рисунок 49).

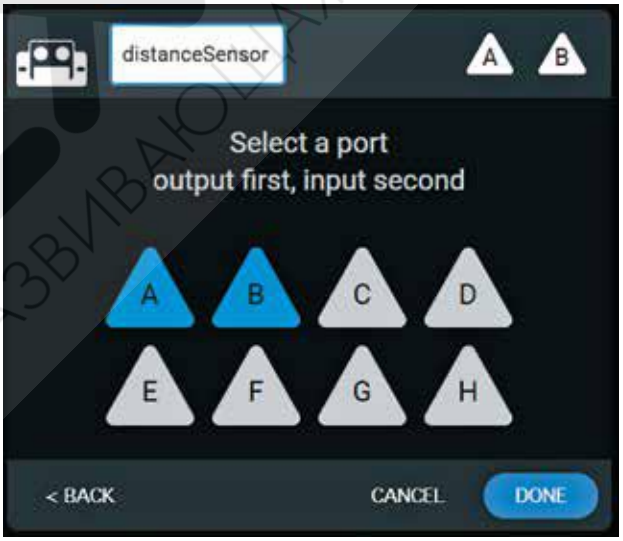


Рисунок 49. Настройка датчика расстояния

Обратите внимание, что в данном случае к порту «A» подключается кабель «output», а к порту «B» – «input».

Рассмотрим код программы:

```
#include «vex.h»

using namespace vex;

int main() {

  vexcodeInit();

  leftMotor.setVelocity(50, percent);
  rightMotor.setVelocity(50, percent);

  while(distanceSensor.distance(mm) > 100)
  {
    leftMotor.spin(forward);
    rightMotor.spin(forward);
  }

  leftMotor.stop();
  rightMotor.stop();

}
```

Робот будет ехать вперед до тех пор, пока датчик не зафиксирует препятствие на расстоянии 100 мм. Для получения показаний с датчика используем функцию **distance**. В общем виде функция будет выглядеть следующим образом:

<имя датчика>.distance(<единицы измерения: mm – миллиметры, inch - дюймы>);

Задача для самостоятельного решения:
Робот едет вперед. В случае наличия препятствия на расстоянии 200 мм он поворачивается направо в течение двух секунд и едет дальше. В случае обнаружения нового препятствия алгоритм повторяется.

Программирование датчика касания

Датчик касания также относится к числу наследуемых датчиков VEX (Рисунок 50).

Рисунок 50. Датчик касания типа бампер

Подключение происходит также к одному из восьми трехпроводных портов. Рассмотрим код программы:

```
#include «vex.h»
using namespace vex;
int main() {
 vexcodeInit();
 leftMotor.setVelocity(50, percent);
 rightMotor.setVelocity(50, percent);
 while(!Brain.Screen.pressing())
 {
  while(touchSensor.pressing())
  {
   leftMotor.spin(forward);
   rightMotor.spin(forward);
  }
  leftMotor.spin(reverse);
  rightMotor.spin(reverse);
 }
  leftMotor.stop();
  rightMotor.stop();
}
```

В данной программе будем использовать датчик касания для переключения направления движения тележки. Если датчик не нажат, то тележка едет назад, если нажат, то вперед. Для считывания показаний с датчика используется функция **pressing**. В общем виде функция будет выглядеть следующим образом:

<имя датчика>.pressing()

Внешний цикл программы **while** будет работать до тех пор, пока оператор не нажмет на экран.

Программирование потенциометра

В состав набора входит датчик, называемый потенциометром (Рисунок 51).

Рисунок 51. Потенциометр

Потенциометр является аналоговым датчиком, который определяет текущий угол поворота выходного вала привода или механизма относительно нулевого положения. Особенностью датчика является ограниченный рабочий угол. Для потенциометра в наборах VEX он лежит в промежутке от нуля (0) до двухсот сорока (240) градусов. Потенциометр может быть использован в тех степенях свободы робота, угол поворота в которых ограничен механически или же его необходимо ограничить во избежание поломок робота. Подключение потенциометра происходит аналогично другим наследуемым датчикам через трехпроводные порты на торце контроллера.

Для изучения работы потенциометра необходимо собрать конструкцию, изображенную на рисунке 52 (конструкция носит рекомендательный характер; параметры конечных элементов могут отличаться).

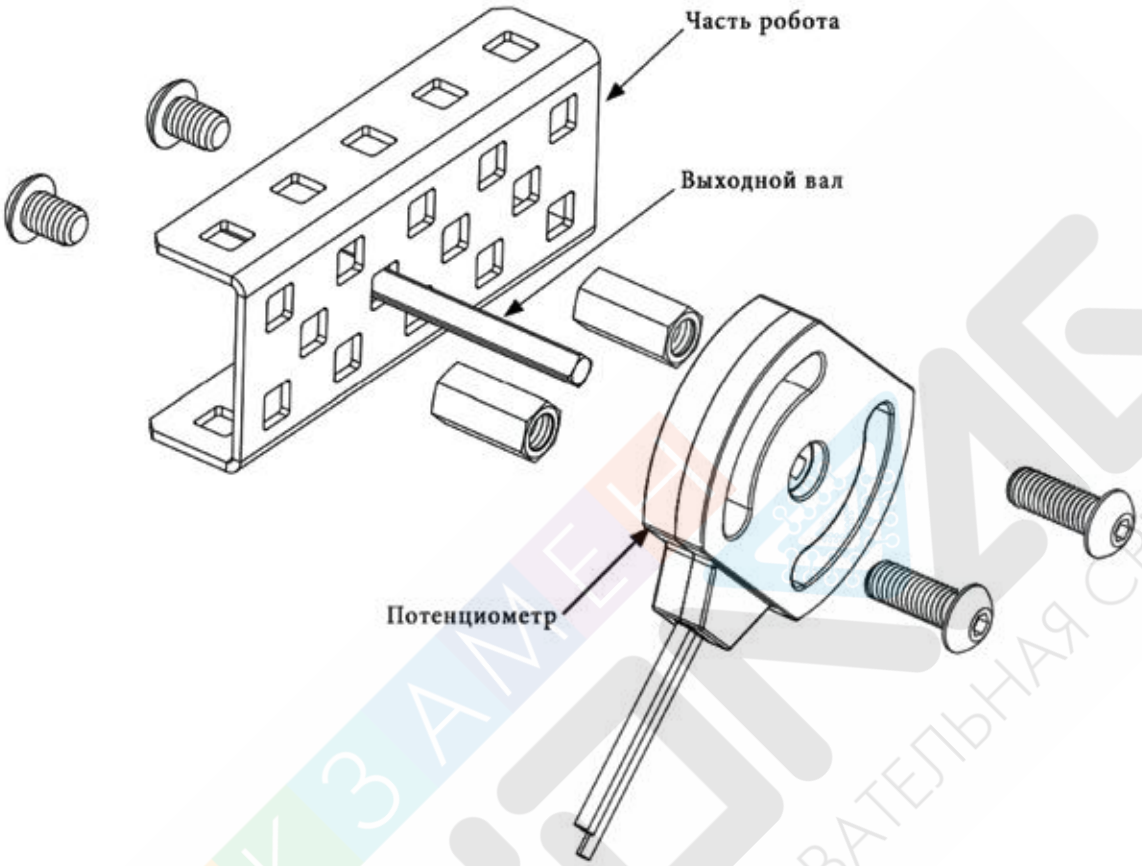


Рисунок 52. Сборка конструкции для программирования потенциометра

Также рекомендуем надеть на вал большую шестерню для удобства вращения вала (Рисунок 53).

Рисунок 53. Установка шестерни на вал

Подключите потенциометр к трехпроводному порту A.

Рассмотрим код программы:

```
#include «vex.h»
using namespace vex;
int main() {
  vexcodeInit();
  while(1)
  {
    if(potentiometer.angle(degrees) == 0 || potentiometer.angle(degrees) == 250)
    {
   Brain.Screen.print(«Достигнуто одно из предельных положений потенциометра!»);
      wait(2, sec);
      Brain.Screen.clearScreen();
    }
    else
    {
      Brain.Screen.print(«Вращайте потенциометр!»);
    }
  }
}
```

В ходе выполнения программы на экране будет появляться фраза о достижении потенциометром своих предельных положений или фраза о продолжении вращения потенциометра. Для получения показаний с потенциометра используется функция **angle**. В общем виде функция будет выглядеть следующим образом:

<имя датчика>.angle(<единицы измерения: degrees - градусы>)

Задача для самостоятельного решения:
Установите потенциометр на одно из колес без привода. Напишите программу, во время которой робот «буксует» на месте в силу достижения потенциометром своих предельных значений.

Программирование камеры

В настоящее время большинство автономных робототехнических систем используют техническое зрение для взаимодействия с окружающим миром и обнаружения различных объектов. Наборы VEX V5 также могут содержать камеру (Рисунок 54), одной из особенностей которой является запоминание до семи (7) объектов с целью дальнейшего их распознавания и использования.

Рисунок 54. Камера VEX V5

Для добавления камеры воспользуйтесь меню настройки «Robot Configuration» (Рисунок 55).

Рисунок 55. Добавление камеры

Для настройки камеры используется отдельное окно (Рисунок 56), которое можно открыть нажатием кнопки «Configure».

Рисунок 56. Меню настройки камеры

Для добавления объекта в поле видимости камеры необходимо поставить галочку в пункте «Set» напротив интересующей ячейки хранения. Вы можете самостоятельно назначить имя объекта, нажав на поле слева от «Set». Корректировка яркости камеры происходит с помощью стрелок рядом с «Brightness», находящейся справа от кнопки очистки всего хранилища «Clear All». Рассмотрим пример захвата красного кубика в поле видимости камеры (Рисунок 57).

Рисунок 57. Захват кубика

Для захвата объекта необходимо выделить объект прямоугольной рамкой и далее нажать на «Set». Захват будет более точным при установке объекта на однотонном фоне и с использованием функции «заморозки» экрана «Freeze». После добавления объекта в хранилище и отключения «заморозки» экрана в левом верхнем углу вашего объекта появятся координаты его левой верхней вершины относительно абсолютной системы координат камеры, находящейся в левом верхнем углу рабочего окна, и размеры объекта (Рисунок 58).

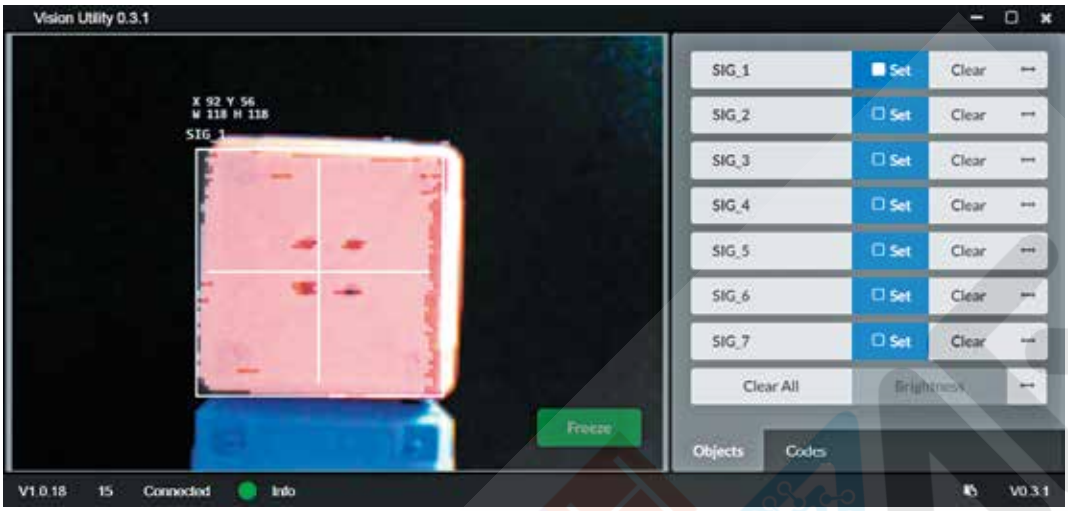


Рисунок 58. Параметры объекта

Во вкладке «Codes» (Рисунок 59) вы можете создавать коды объектов, которые будут представлять собой комбинации кодов уже захваченных объектов.

Рисунок 59. Комбинирование объектов в меню Code

Кроме того, камера обладает встроенным WiFi модулем, который позволяет удаленно получать с камеры изображение на экране любого устройства, обладающего встроенным WiFi модулем. Для подключения изображения с камеры по WiFi вам необходимо:

1. Включить WiFi модуль камеры в меню «Setting» на контроллере V5;
2. Подключиться к WiFi точке на устройстве, которое вы будете использовать для просмотра изображения с камеры;
3. Ввести в строке браузера 192.168.1.1 для доступа к камере.

Программа следования за объектом

Довольно часто роботу необходимо следовать за некоторым объектом на определенном расстоянии, при этом соблюдая положение объекта относительно некоторой системы координат (например, связанной с камерой). Примером использования такой задачи может служить футбол роботов, во время которого роботу необходимо обнаруживать перед собой мяч и вести его к воротам соперника. Также камеру можно использовать в матчах VRC Tower Takeover для обнаружения кубов и установки их друг на друга.

Рассмотрим программу:

```
#include «vex.h»
using namespace vex;
int main() {
  vexcodeInit();
  leftMotor.setMaxTorque(100, percent);
  rightMotor.setMaxTorque(100, percent);
  int speed = 0;
  speed = 10;
  leftMotor.setVelocity(speed, percent);
  rightMotor.setVelocity(speed, percent);

  while (1) {
    Vision20.takeSnapshot(Vision20__SIG_1);
    if (Vision20.objects[0].width > 45) {
      if (Vision20.objects[0].centerX > 164) {
        rightMotor.spin(reverse);
        leftMotor.spin(forward);
      } else if (Vision20.objects[0].centerX < 156) {
        leftMotor.spin(reverse);
        rightMotor.spin(forward);
      }
    }
    else
    {
      leftMotor.stop();
      rightMotor.stop();
    }
  }
}
```

Перед написанием кода нам необходимо настроить камеру на некоторый объект, за которым роботу необходимо следить. Напоминаем, что настройка камеры ведется

с помощью меню настройки камеры при выборе порта подключения.

Важной особенностью работы камеры VEX является поиск объекта на последнем сделанном снимке окружающей среды, называемого «снэпшотом». То есть для того чтобы робот постоянное обрабатывал информацию об окружающей среде на наличие требуемых объектов, необходимо получать снимки экрана огромное число раз. Поэтому мы запускаем бесконечный цикл. Для получения снимка с камеры используется функция **takeSnapshot**. В общем виде функция будет выглядеть следующим образом:

<имя датчика>.takeSnapshot(<имя искомого объекта>)

В силу особенностей освещения некоторые объекты окружающей среды могут быть восприняты в качестве искомого объекта, поэтому нам необходимо отсеять такого рода объекты (такие объекты называются шумом). В данной программе мы будем отсеивать такого рода объекты с помощью установки нижнего предела для ширины объекта. Для получения ширины найденного объекта (по предложенной сигнатуре в функции **takeSnapshot**) используется класс **width**. Для определения высоты объекта используется класс **height**. В общем виде поле класса **objects** будет выглядеть следующим образом:

<имя камеры>.objects[<номер объекта из массива захваченных объектов, где 0 – самый крупный объект в массиве>].width

Для поворота робота в нужную сторону мы будем отслеживать координату по *X* его центра. Если координата будет правее центра плоскости абсолютной системы координат, то поворачиваем направо, если левее – налево. Координаты центра объекта можно получить с помощью функций **centerX** и **centerY**. В общем виде поле класса **objects** будет выглядеть следующим образом:

<имя камеры>.objects[<номер объекта из массива захваченных объектов, где 0 – самый крупный объект в массиве>].centerX

Задача для самостоятельного решения:
Робот отъезжает назад, если объект находится слишком близко (с использованием камеры робота).

Программные средства управления камерой

В данном разделе мы не будем рассматривать готовый код программы, а обратим внимание на функции и элементы классов, которые могут понадобиться для улучшения поиска объектов. Для доступа к этим элементам управления и включения автозаполнения необходимо поставить галочку напротив пункта «Enable Expert Autocomplete».

Камера обладает встроенным светодиодом, цвет которого можно изменить с помощью функции **setLedColor** по шкале rgb. В общем виде функция будет выглядеть следующим образом:

<имя камеры>.setLedColor(<красный канал>, <зеленый канал>, <синий канал>)

Важно! Перед установкой цвета с помощью функции **setLedColor** необходимо установить режим работы светодиода в ручной с помощью следующего выражения:

<имя камеры>.setLedMode(vex::vision::ledMode::manual);

Кроме того, для работы с объектом при различном окружении часто применяют такие параметры, как яркость и баланс белого. Регулировка яркости камеры происходит с помощью функции **setBrightness**. В общем виде функция будет выглядеть следующим образом:

<имя камеры>.setBrightness(<значение яркости в процентах>)

Баланс белого – это процесс цветокоррекции, в результате которой объекты, которые глаз видит как белые, будут показаны белыми на вашем снимке. В нашем случае баланс белого имеет смысл применять в ситуациях, при которых искажается цвет объекта (например, при желтом освещении). Для управления параметрами баланса белого используется функция **setWhiteBalanceValues**. В общем виде функция будет выглядеть следующим образом:

<имя камеры>.setWhiteBalanceValues(<красный канал>, <зеленый канал>, <синий канал>)

Важно! Перед установкой параметров баланса белого с помощью функции **setWhiteBalanceValues** необходимо установить режим работы фильтра белого баланса в ручной с помощью следующего выражения:

<имя камеры>.setWhiteBalanceMode(vex::vision::whiteBalanceMode::manual)

Функции расширяют функционал управления параметрами изображения, получаемого с камеры, для дальнейшей сегментации и фильтрации.

Программирование пульта управления

В наборах VEX V5 есть пульт управления (Рисунок 60), который позволяет удаленно управлять движениями робота.

Рисунок 60. Пульт управления V5

Пульт управления оснащен экраном, который позволяет отображать показания с контроллера (Рисунок 61).

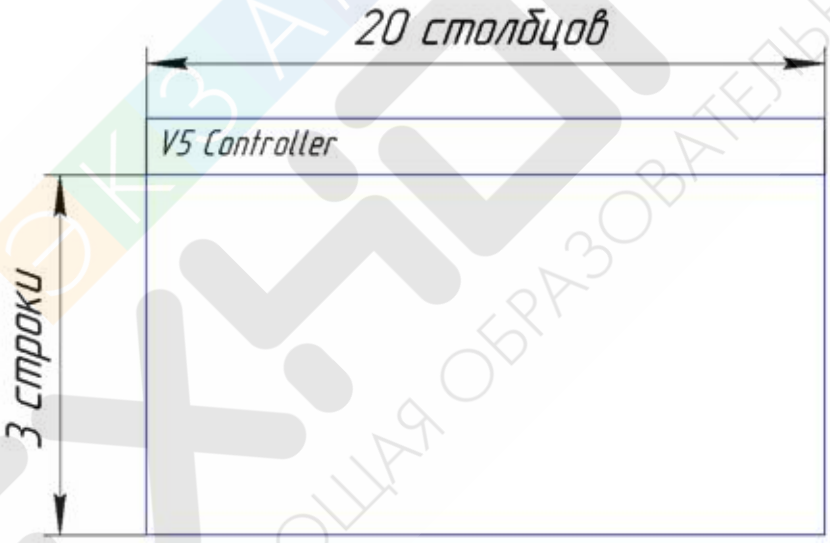


Рисунок 61. Параметры пульта управления

Перед использованием пульта управления его необходимо добавить в меню «Robot Configuration». Для данной программы необходимо подключение камеры и пульта управления к контроллеру.

Рассмотрим программу:

```
#include «vex.h»
using namespace vex;
int main() {
  vexcodeInit();
  int speedRight = 0;
  int speedLeft = 0;;
  visionSens.setLedMode(vex::vision::ledMode::manual);
  while(1)
  {
    speedLeft = Controller1.Axis3.value() + Controller1.Axis4.value();
    speedRight = Controller1.Axis3.value() - Controller1.Axis4.value();
    leftMotor.setVelocity(speedLeft, percent);
    rightMotor.setVelocity(speedRight, percent);
    leftMotor.spin(forward);
    rightMotor.spin(forward);
    if(Controller1.ButtonA.pressing())
    {
      visionSens.setLedColor(255, 0, 0);
    }
    else
    {
      visionSens.setLedColor(0, 0, 255);
    }
  }
}
```

В ходе данной программы мы будем управлять колесами робота с помощью левого джойстика и переключать цвет светодиода на камере с помощью кнопки A.

Для понимания процесса написания программы для управления колесами нам необходимо разобраться со знаками скоростей колес. Посмотрим на таблицу 5.

Таблица 5. Знаки скоростей при движении в различных направлениях

	поворот направо	поворот налево	вперед	назад
leftSpeed	+	-	+	-
rightSpeed	-	+	+	-

Джойстик в нейтральном положении будем рассматривать как начало координатной плоскости 3 – 4, где 3 – ось *Y*, а 4 – ось *X* (Рисунок 62).

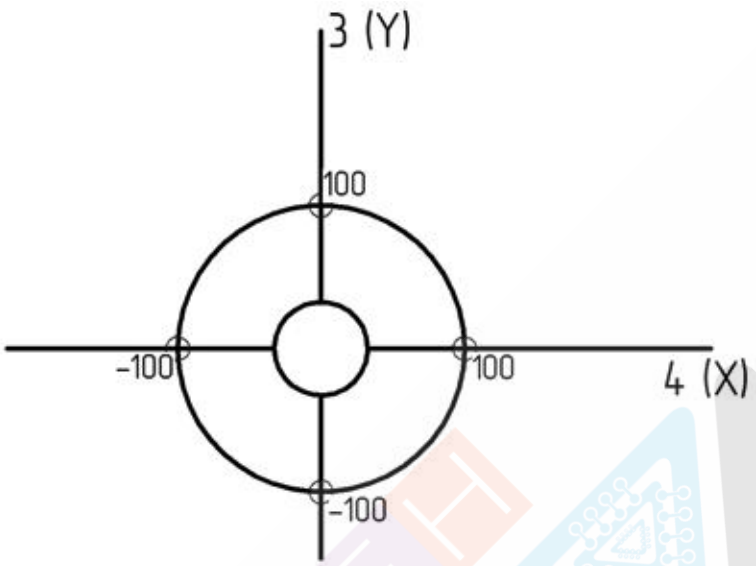


Рисунок 62. Представление джойстика в системе координат 3–4

На рисунке видно, что максимальное значение джойстика равно 100. Рассмотрим случай поворота направо для написания конечного выражения, которое будет определять текущее значение скорости на приводе.

Пусть значение джойстика по оси 4_*X* = 100, а по оси 3_*Y* = 0. Скорость привода будет складываться из двух этих значений. Для поворота направо необходимо, чтобы значение скорости на левом колесе было больше, чем на правом. Запишем выражения для скоростей, учитывая все сказанное выше:

```
leftSpeed = 3_Y + 4_X = 100
rightSpeed = 3_Y - 4_X = -100
```

Для снятия показаний о текущем положении джойстика используется функция **value**. В общем виде функция будет выглядеть следующим образом:

<имя пульта управления>.<имя кнопки или оси>.value()

Для получения информации о состоянии кнопки пульта управления используется функция **pressing**, которая возвращает **true**, если кнопка нажата, и **false**, если не нажата. В общем виде функция будет выглядеть следующим образом:

<имя пульта управления>.<имя кнопки или оси>. pressing()

Задача для самостоятельного решения:

Робот управляется с пульта правым джойстиком. При нажатии на кнопку «B» он поворачивается на месте до тех пор, пока не будет нажата кнопка «X».

Программирование пневматических цилиндров

В набор входит пневматическая линия с возможностью управления с контроллера через трехпиновые порты подключения.

Рассмотрим код программы:

```
#include «vex.h»
using namespace vex;
digital_out dig1 = digital_out(Brain.ThreeWirePort.A);
int main() {
  int count = 0;
  while(1) {
    if( Controller1.ButtonL1.pressing() ) {
      dig1.set( true );
    }
    else {
      dig1.set( false );
    }
    wait(100, msec);
  }
}
```

Первым делом нам нужно определить порт подключения, к которому мы подключим нашу управляющую плату цилиндра. В данной программе используем для этого порт подключения A.

digital_out <имя порта> = digital_out(Brain.ThreeWirePort.<название порта>);

Управление цилиндром будет осуществляться с помощью нажатия кнопки L1 на пульте управления. Для активации цилиндра необходимо подать напряжение на цифровой вход A. Подача напряжения осуществляется с помощью функции **set**. В общем виде функция будет выглядеть следующим образом:

<имя порта подключения>.set(<true или false>);

Получение информации об аккумуляторе

Среда программирования VEXcode позволяет получать информацию о состоянии аккумулятора для того, чтобы корректировать работу робота.

Для получения информации об уровне заряда батареи используется функция

capacity. В общем виде функция будет выглядеть следующим образом:

```
Brain.Battery.capacity()
```

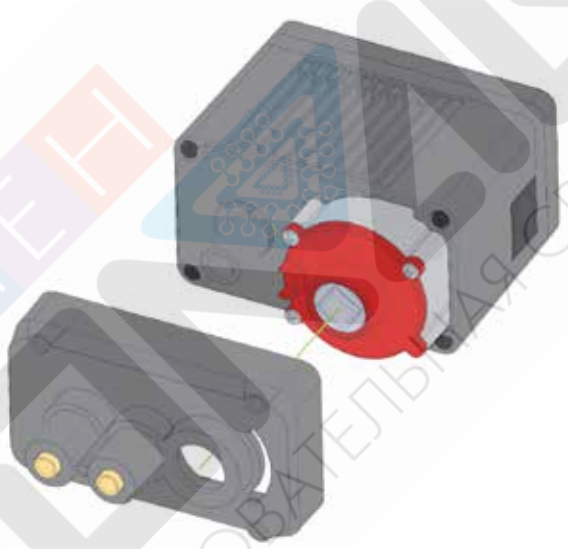
Для получения информации о температуре батареи используется функция **temperature**. В общем виде функция будет выглядеть следующим образом:

```
Brain.Battery.temperature()
```

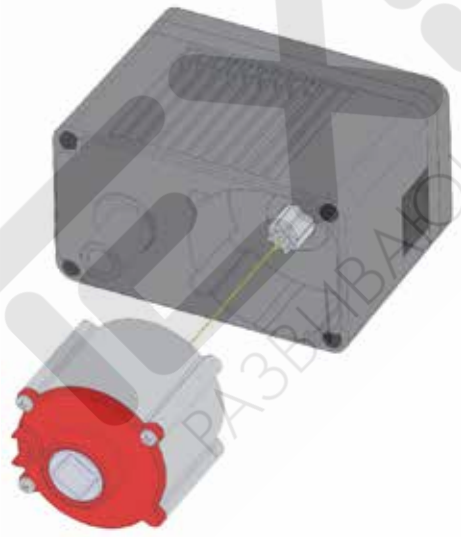
Приложение 1.
Замена картриджа редуктора в приводе V5

Шаг 1. Извлеките 4 винта из крышки привода.

Шаг 2. Удалите крышку привода.



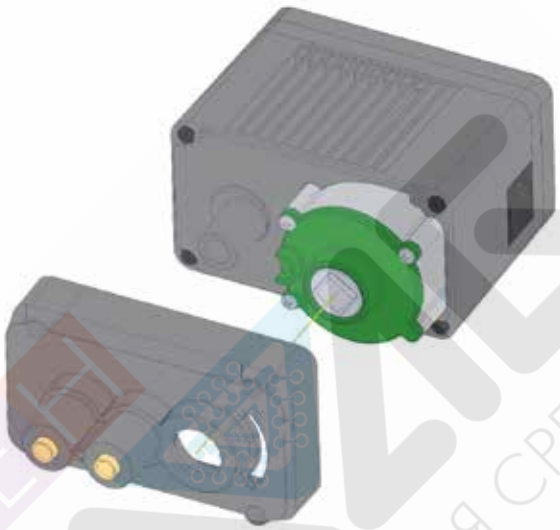
Шаг 3. Извлеките картридж редуктора.



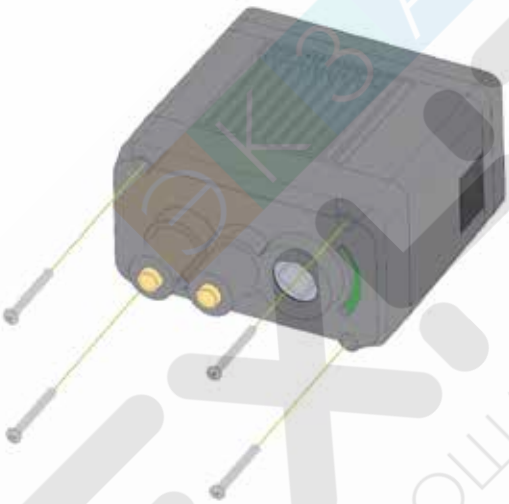
Шаг 4. Вставьте новый картридж редуктора.

Шаг 5. Убедитесь в том, что картридж установлен плотно.

Шаг 6. Установите крышку привода.



Шаг 7. Вставьте 4 винта в пазы.



Шаг 8. Затяните винты. Готово!

Приложение 2.
Порядок создания соединения между контроллером и пультом

Шаг 1. Подготовьте требуемые компоненты.

Вам понадобятся:
- Контроллер с подсоединенным аккумулятором;
- Радиомодуль;
- Пульт управления;
- Два кабеля для соединения с контроллером и пультом.

Шаг 2. Соедините между собой контроллер и пульт управления.

- Вставьте кабель в разъем пульта управления на торцевой панели и в любой порт контроллера;
- Включите контроллер;
- Пульт управления автоматически включится.

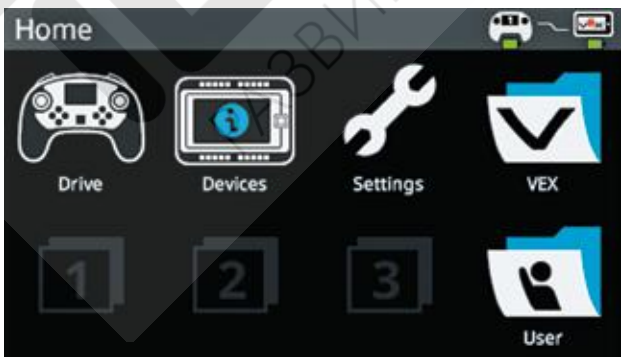
Шаг 3. Проверьте, что произошла синхронизация между контроллером и пультом.

• В случае успешной синхронизации на пульте появится иконка, показанная на изображении выше.

Шаг 4. Подсоедините радиомодуль к контроллеру.

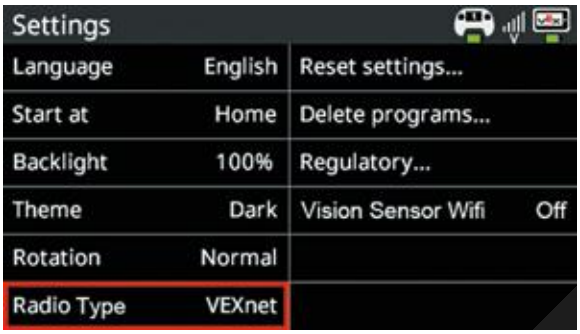
• Радиомодуль можно подключить к любому свободному порту.

Шаг 5. Убедитесь в создании соединения между устройствами.



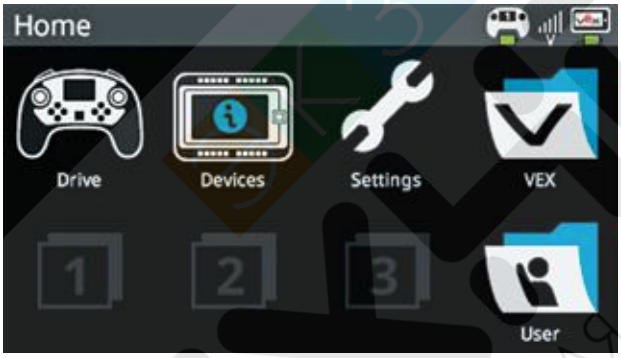
• В случае успешного соединения в правых верхних углах контроллера и пульта управления будет иконка, показанная на рисунках выше.

Шаг 6. Установите режим радио.



- Зайдите в меню «Settings»;
- Нажмите на пункт «Radio Type» для выбора режима «VEXnet»;
- Подтвердите смену режима работы.

Шаг 7. Установите режим радио.



- Проверьте работу пульта, отсоединив его от контроллера;
- В случае успешного подключения радиомодуль будет мигать красным цветом.

Приложение 3.
Инструкция по сборке пневматической линии

Часть 1. Сборка компонентов пневматической линии

Шаг 1. Удалите запирающие пломбы из отверстий резервуара. Установите фитинг под шланг и соединительный фитинг под насос в соответствующие пазы (ключ 7/16 дюймов не входит в состав набора). Затягивайте элементы до тех пор, пока резьба полностью не исчезнет.

Шаг 2. Установите элемент контроля потока воздуха и фитинг под шланг в оба цилиндра с помощью небольших плоскогубцев (не входят в состав набора). После исчезновения резьбы затяните вручную на ¼ оборота.

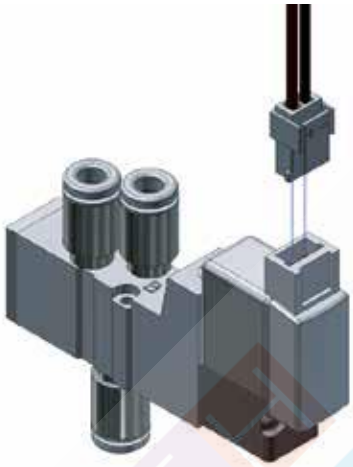
Шаг 3. Установите на цилиндр элементы крепления, состоящие из двух гаек и п-образной планки (ключ 5/16 дюймов не входит в состав набора).

Шаг 4. Присоедините монтажную пластину с отверстиями для крепления к роботу с помощью винта дюймового винта 8-32 и нейлоновой гайки 8-32.

Шаг 5. Установите штуцер с зажимом под шланг в центральное отверстие управляющей платы. После того как резьба полностью исчезнет, затяните пальцами штуцер примерно на ¼ оборота.

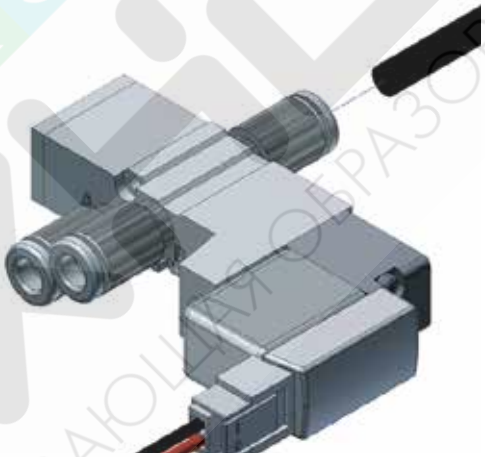
Шаг 6. Установите еще два штуцера в отверстия управляющей платы с противоположной стороны. После того как резьба полностью исчезнет, затяните пальцами штуцер примерно на ¼ оборота.

Шаг 7. Подсоедините кабель к порту подключения управляющей платы. Обратите внимание, что кабель устанавливается в разъем единственно возможным способом.



Часть 2. Сборка пневматической линии

Шаг 1. Обрежьте шланг в соответствии с требуемыми техническими характеристиками. Обратите внимание, что при срезе необходимо направлять режущий инструмент (не входит в набор) перпендикулярно сечению шланга.

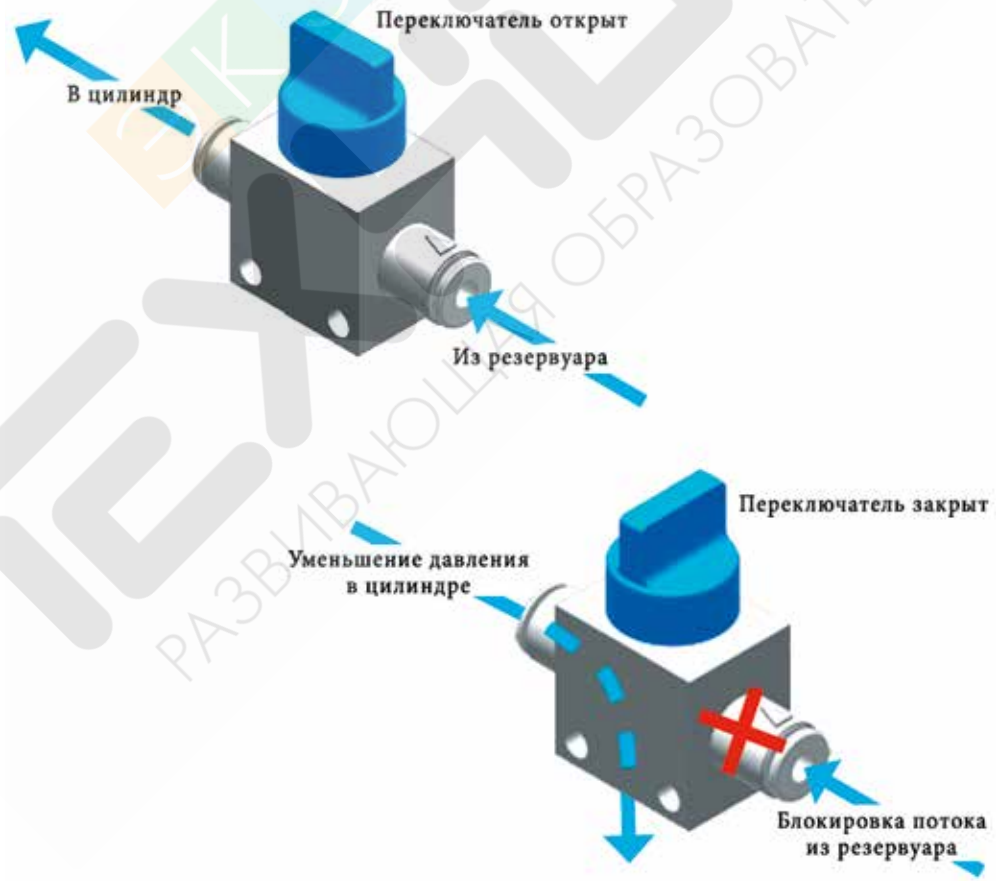


Шаг 2. Заполните резервуар воздухом с помощью насоса (не входит в состав набора) через соединительный фитинг под насос.

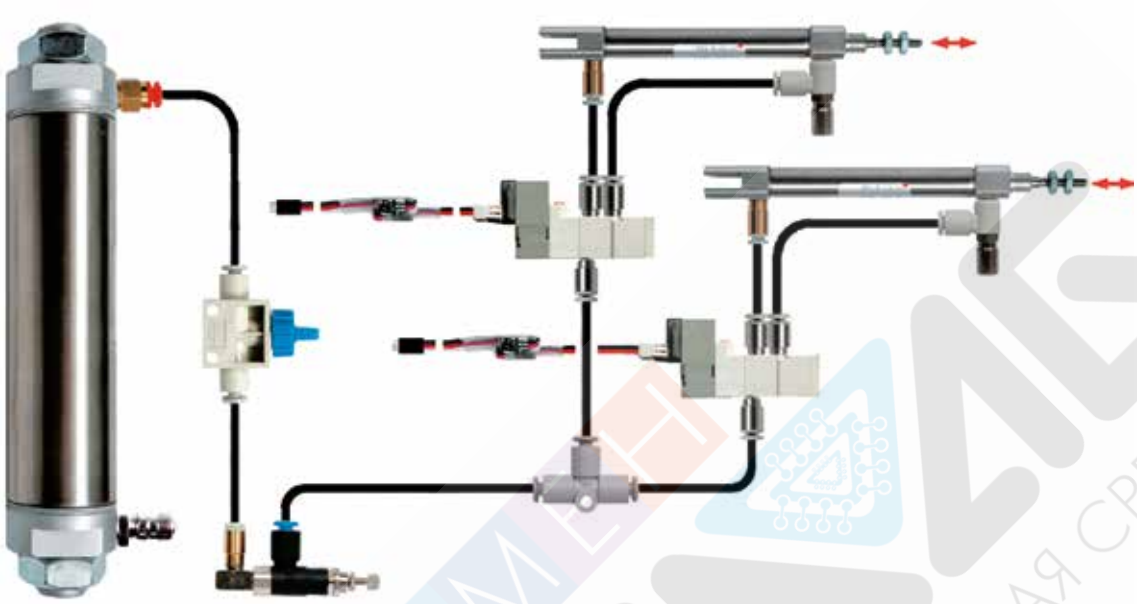
Внимание! Максимальное входное давление должно быть не больше 0,7 Мпа. Использование более высоких значений давления может повредить оборудование.

Шаг 3. Используйте регулятор давления для установки необходимого вам давления в цилиндре.

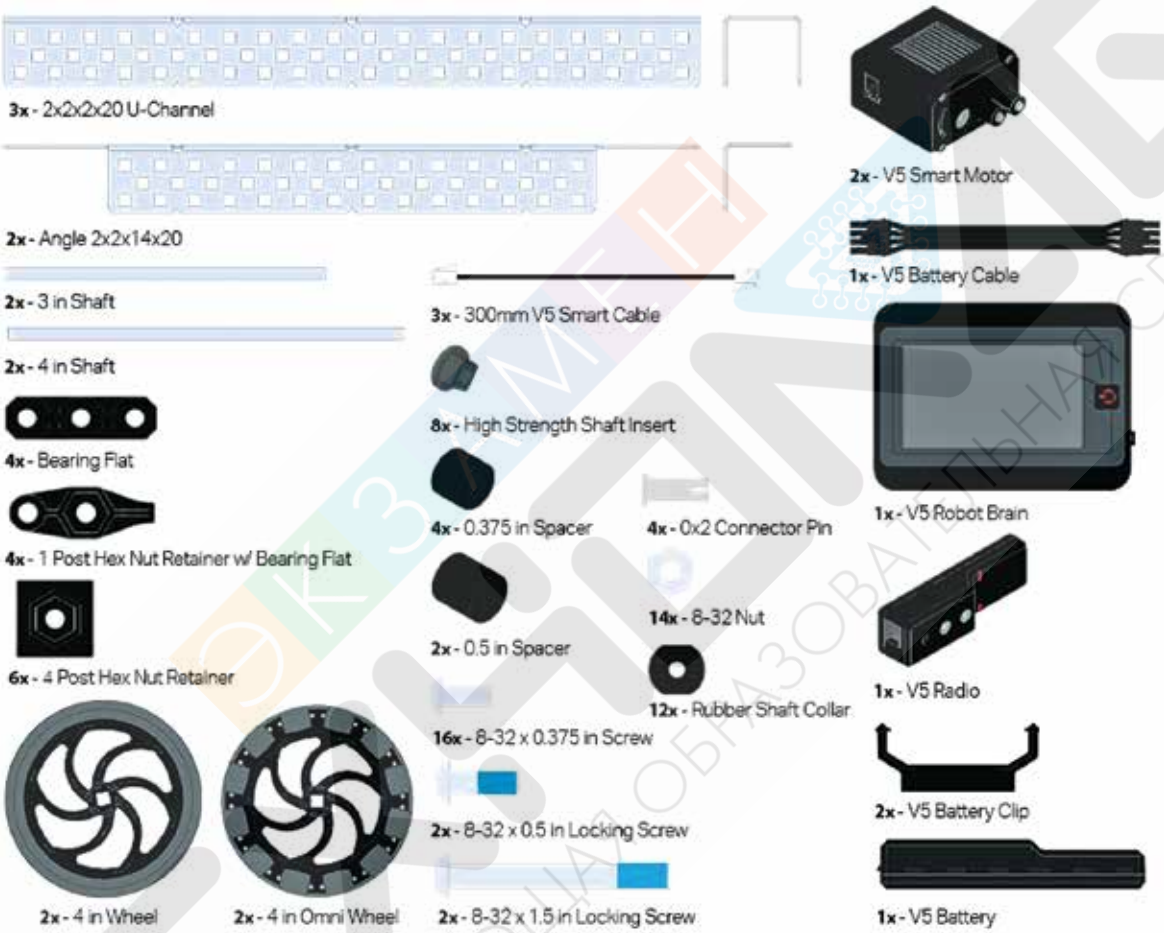
Шаг 4. Включите в пневматическую линию переключатель подачи воздуха из резервуара.

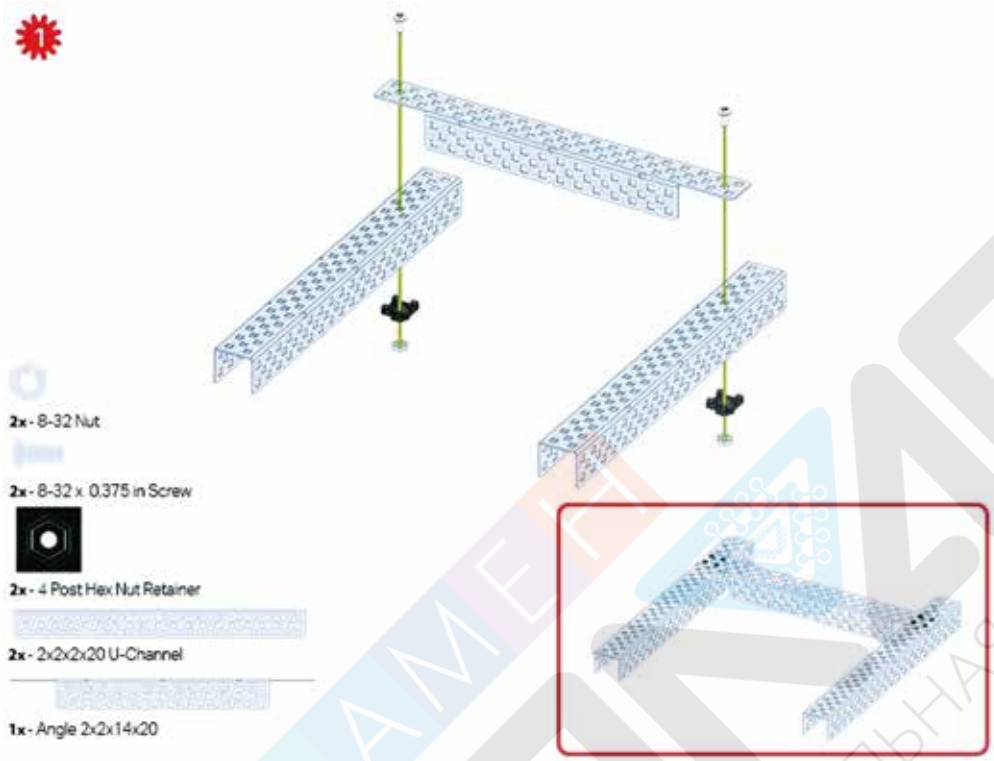


Шаг 5. Пневматическая линия готова!



Приложение 4.
Инструкция по сборке SpeedBot с датчиками





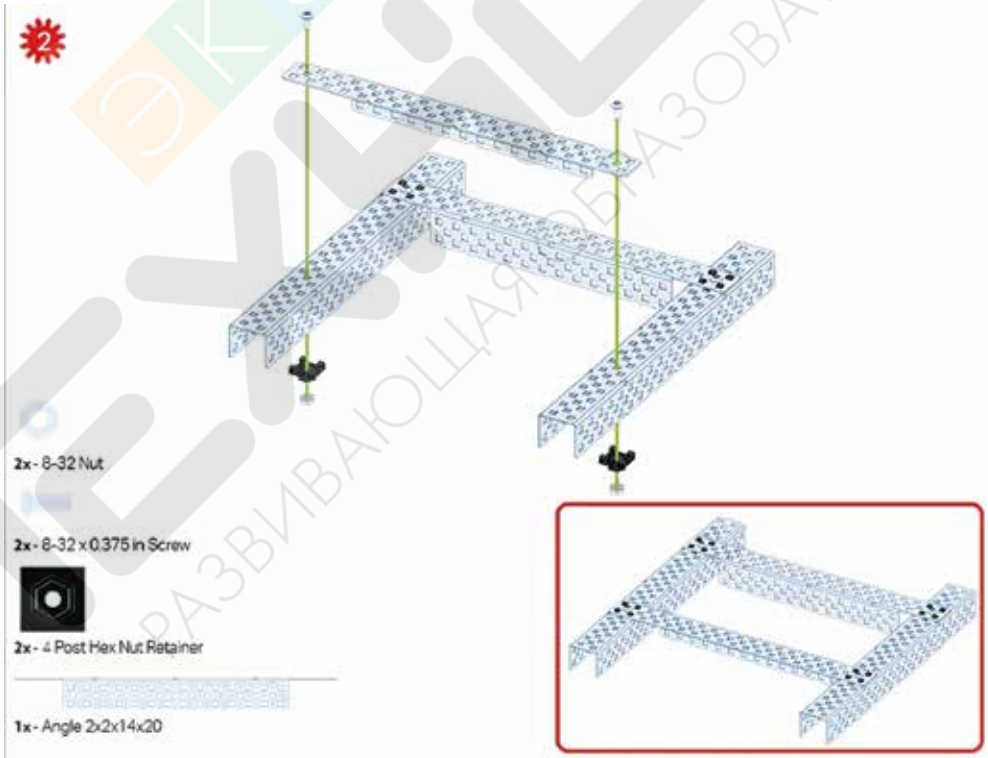
2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 4 Post Hex Nut Retainer

2x - 2x2x2x20 U-Channel

1x - Angle 2x2x14x20



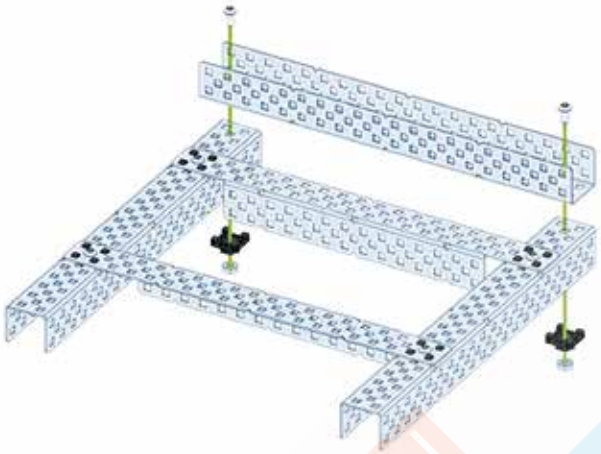
2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 4 Post Hex Nut Retainer

1x - Angle 2x2x14x20

3

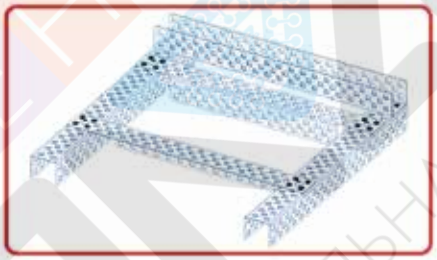


2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 4 Post Hex Nut Retainer

1x - 2x2x2x20 U-Channel



4

2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 1 Post Hex Nut Retainer w/ Bearing Flat

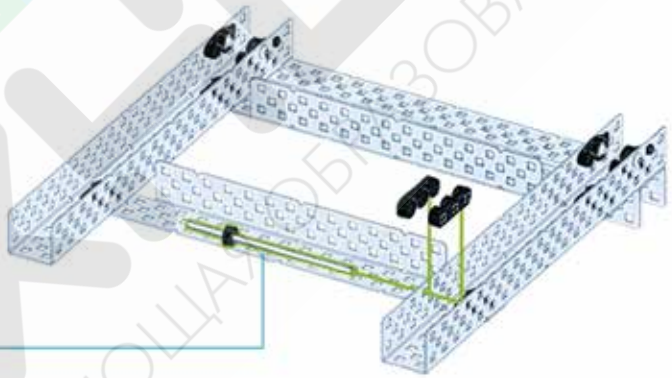
5

2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 1 Post Hex Nut Retainer w/ Bearing Flat

6

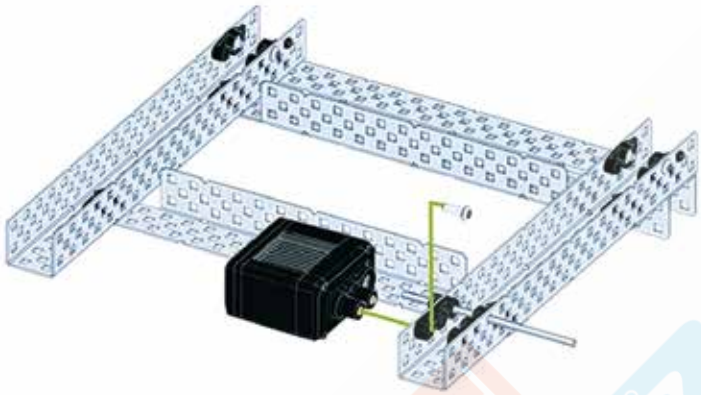


2x - Rubber Shaft Collar

2x - Bearing Flat

2x - 4 in Shaft

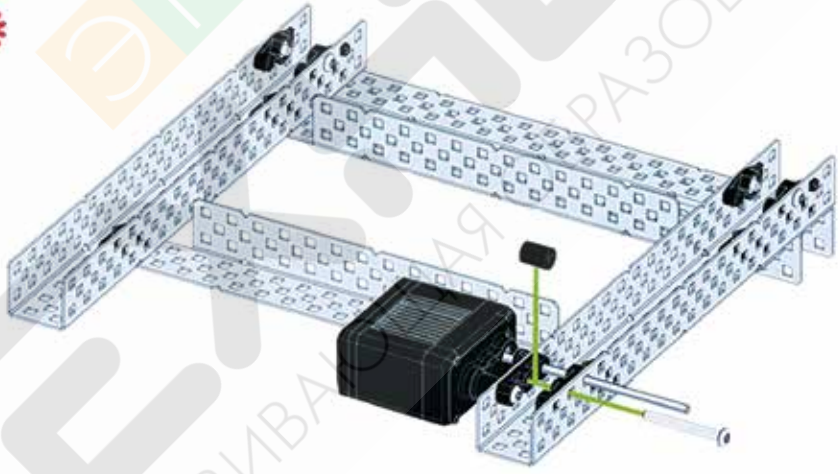
7



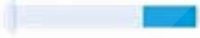
1x - 8-32 x 0.5 in Screw

1x - V5 Smart Motor

8

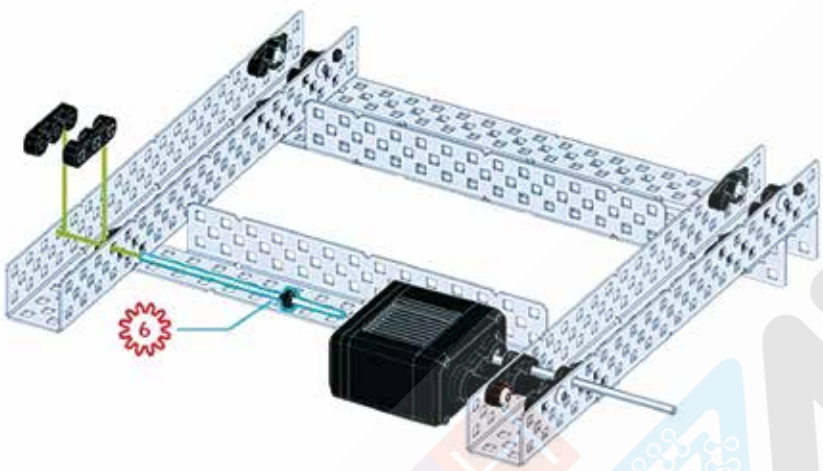


1x - 0.5 in Spacer



1x - 8-32 x 1.5 in Screw

9



2x - Bearing Flat

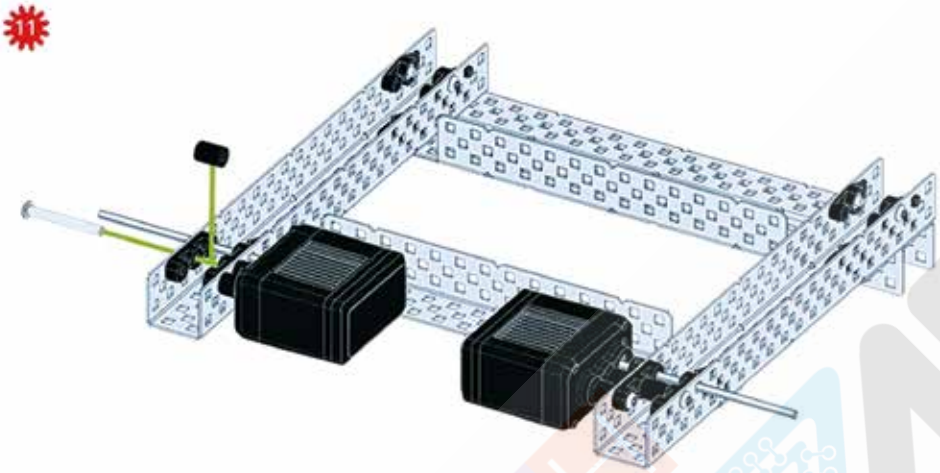
1x - Step 6 Sub-Assembly

10

1x - 8-32 x 0.5 in Screw

1x - V5 Smart Motor

11



1x - 0.5 in Spacer

1x - 8-32 x 1.5 in Screw

12

1x - 0.375 in Spacer

2x - Rubber Shaft Collar

1x - 4 in Wheel

2x - High Strength Shaft Insert

13

1x - 0.375 in Spacer

2x - Rubber Shaft Collar

1x - 4 in Wheel

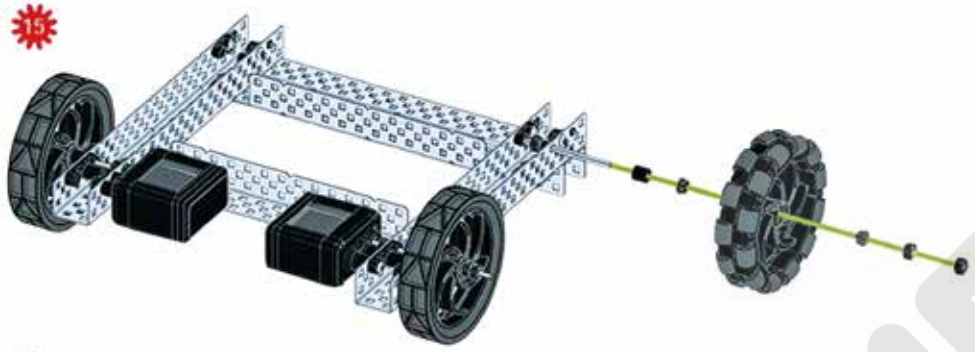
2x - High Strength Shaft Insert

14

1x - Rubber Shaft Collar

1x - 3 in Shaft

15

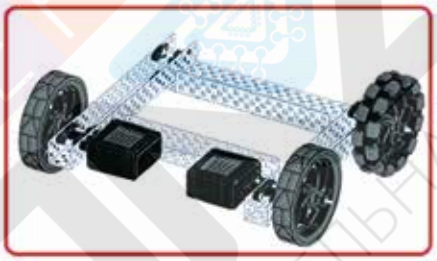


1x - 0.375 in Spacer

2x - Rubber Shaft Collar

1x - 4 in Omni Wheel

2x - High Strength Shaft Insert

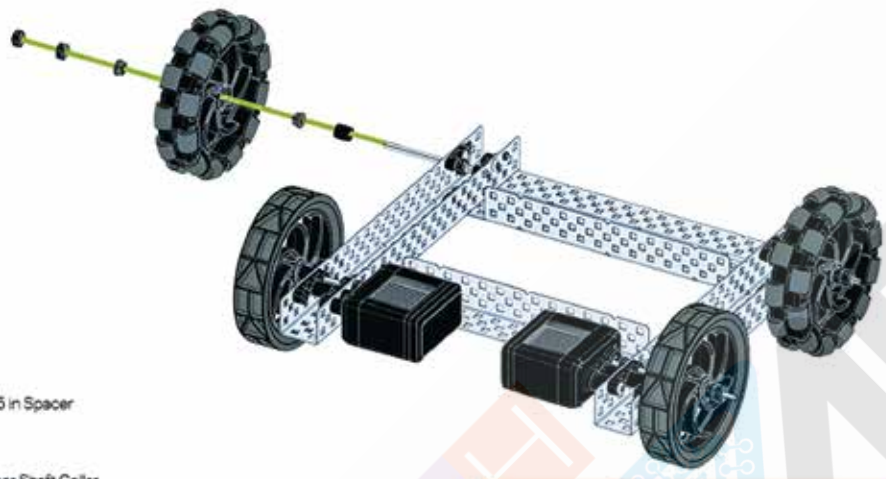


16

1x - Rubber Shaft Collar

1x - 3 in Shaft

17

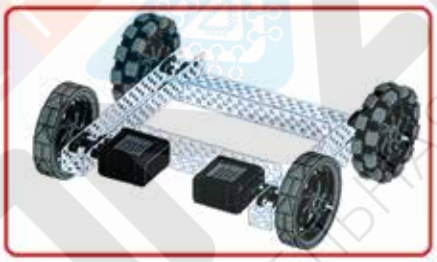


1x - 0.375 in Spacer

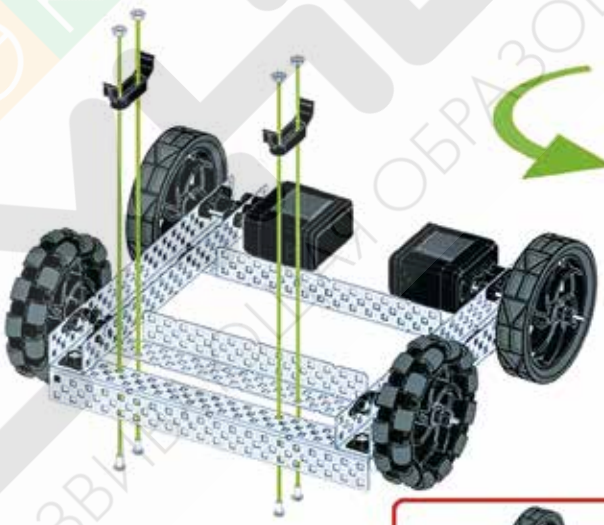
2x - Rubber Shaft Collar

1x - 4 in Omni Wheel

2x - High Strength Shaft Insert



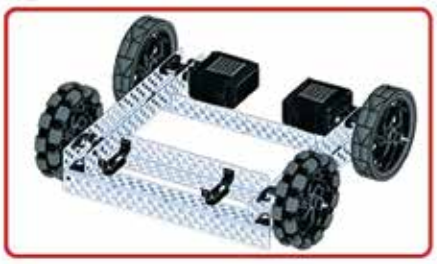
18



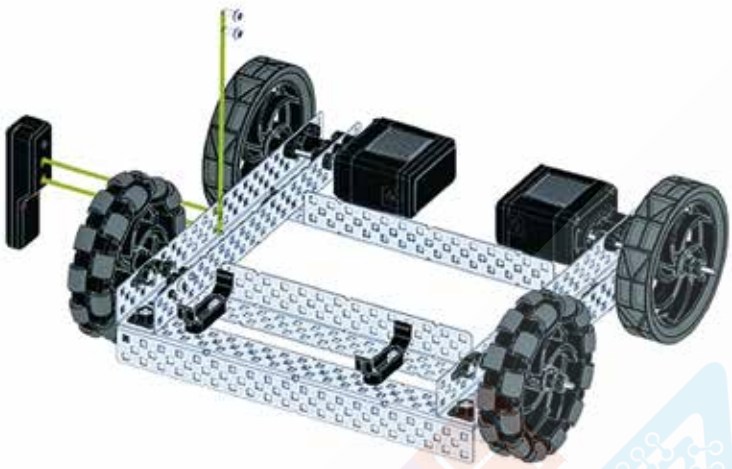
4x - 8-32 x 0.375 in Screw

4x - 8-32 Nut

2x - V5 Battery Clip

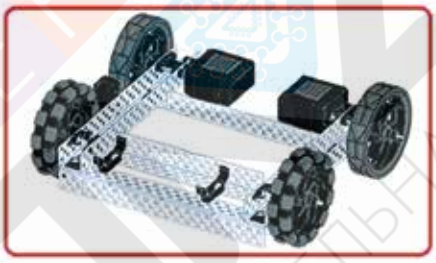


19



1x - V5 Radio

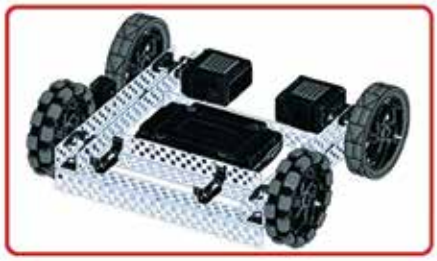
2x - 8-32 x 0.375 in Screw

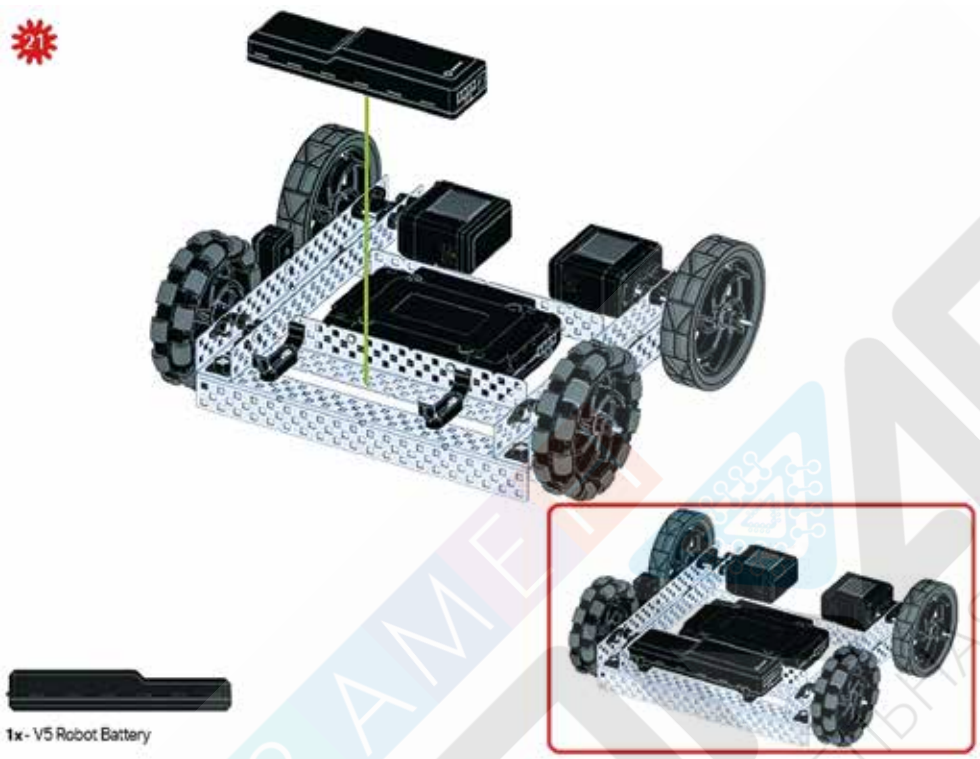


20

4x - 0x2 Connector Pin

1x - V5 Robot Brain





1x - V5 Robot Battery

1x - Battery Cable

3x - 300mm V5 Smart Cable

Пример реализации проекта по Робофутболу

Робофутбол

Инженеры создавали и программировали роботов для игры в футбол на протяжении более двух десятилетий. Несколько организаций, таких как Федерация RoboCup и FIRA, проводят турниры, в которых команды со всего мира соревнуются с роботами, играющими в футбол (Рисунок 1). Как вы думаете, почему робофутбол так популярен?

Рисунок 1. Пример футбола роботов

Две основные причины популярности робофутбола:

• Развлечение – футбол является одним из самых популярных видов спорта в мире. Многие люди любят смотреть и играть в него. Замена человеческих игроков на интеллектуальные машины добавляет новый глоток свежести в игру.

• Исследования и разработки. Робофутбол – это вид деятельности, требующий, чтобы несколько роботов действовали вместе для выполнения задач (забивания голов и защиты своих ворот), а также реагирования на действия другой команды роботов. Робофутбол – это невероятно креативная и аутентичная среда для разработки алгоритмов, которые позволяют роботам быстро отслеживать траекторию движения объекта, управлять и манипулировать им, а также передавать информацию о своем положении и другую для своих товарищей по команде. Все эти алгоритмы могут быть повторно использованы роботами в других средах для выполнения аналогичных задач.

Подготовка рабочего места

Таблица 1. Необходимое оборудование

Количество	Программное обеспечение
1	VEX V5 Classroom Starter Kit (с актуальной прошивкой контроллера)
3	Конусы или чашки
1	V5 Controller 276-4820
1	Секундомер
1	Футбольный мяч или мяч аналогичного размера
1	Рулон ленты
1	Инженерная тетрадь

Создание трассы для ведения мяча

Установите все три конуса (или чашки) по прямой линии. Убедитесь, что между каждым конусом достаточно места, чтобы робот прошел между ними. Чем больше расстояние между конусами, тем уменьшается сложность. Выберите место на внешней стороне трассы (около одного из крайних конусов) и используйте ленту, чтобы изобразить «X» на земле в этом месте. Поместите футбольный мяч (или мяч аналогичного размера) на «X» (Рисунок 2).

Рисунок 2. Пример трассы

Подготовка робота

Включите контроллер V5, убедитесь, что он подключен к пульту V5 и запустите программу **Drive** на контроллере V5, чтобы вы могли управлять роботом по беспроводной связи с пульта (Рисунок 3).

Рисунок 3. Запуск робота

Ведение мяча

Расположите своего робота рядом с мячом. Никакая часть робота не должна касаться мяча до запуска. Ваша цель – подтолкнуть (толкать) мяч между первым и вторым конусами, а затем между вторым и третьим конусами (Рисунок 4).

Рисунок 4. Ведение конуса

После этого вам нужно обвести мяч вокруг третьего конуса и сделать еще один проход по трассе, еще раз обводя мяч через оба конуса, только наоборот. Используйте секундомер, чтобы узнать, как быстро вы сможете пройти курс. Секундомер должен запускаться, когда вы впервые касаетесь мяча своим роботом. Секундомер должен быть остановлен, как только мяч полностью пройдет через последний набор конусов на втором проходе по трассе. Пройдите трассу несколько раз, чтобы увидеть, как ваше время улучшается с практикой. Запишите свои результаты в вашу инженерную тетрадь.

Ответьте на вопросы в своей инженерной тетради:

- Какое у вас было лучшее время?
- Какие стратегии вы использовали, чтобы улучшить свое время?

Ведение инженерной книги

Внутри вашей инженерной книги отметьте несколько идей для дополнительного оборудования, которое можно добавить в вашу VEX EDR Speed Build, что улучшит его способность вести мяч. Поделитесь своими конструкторскими идеями с другими, когда будете презентовать свои проекты (Рисунок 5).

Рисунок 5. Пример инженерной книги

Ответьте на вопросы ниже в своей инженерной тетради:
- Какие положительные отзывы вы получили о ваших проектах?
- Какие негативные отзывы от других вы получили по своему проекту?
- Какую обратную связь вы учли в своем проекте? Объясните, как вы это сделали.
- Помог ли ваш дизайн другим людям улучшить их роботов? Как именно?

Ваша инженерная тетрадь должна иметь следующее:
- Запись для каждого дня или сеанса, когда вы работали над решением инженерной задачи.
- Записи, которые являются хронологическими и датированными.
- Четкое, аккуратное и лаконичное написание и организация деятельности.
- Метки, чтобы читатель понимал все ваши заметки и то, как они вписываются в ваш процесс проектирования.

Запись может включать в себя:
- Идеи мозгового штурма;
- Эскизы или рисунки прототипов;
- Псевдокод и блок-схемы для планирования;
- Любые отработанные вычисления или используемые алгоритмы;
- Ответы на руководящие вопросы;
- Примечания о наблюдениях и / или проведенных испытаниях;
- Примечания и размышления о ваших различных действиях.

Создание собственной конструкции

Просмотрите свои эскизы и выберите тот, который, по вашему мнению, лучше всего улучшит способность вашего робота вести мяч. Проработайте выбранный дизайн и примените его к своему роботу (Рисунок 6).

Рисунок 6. Пример эскизирования

В качестве примера для управления мячом может послужить конструкция робота, изображенная на рисунке 7.

Рисунок 7. Пример конструкции робота для игры в робофутбол

Шаровая пусковая установка VEX V5 включает датчик зрения, установленный над роботом под углом вниз. Этот робот может быть запрограммирован на вращение до тех пор, пока датчик зрения не обнаружит цветной шар, а затем направится к нему. Когда датчик зрения обнаруживает, что робот находится достаточно близко к мячу, робот может втянуть его, используя впуск, а затем выстрелить в цель или мишень.

Поместите своего робота с новой конструкцией на X. Ни одна часть робота не должна касаться мяча до запуска. Ведите мяч по курсу, запустив секундомер, следуя тому же пути, что и в предыдущих испытаниях без навесного оборудования. Секундомер должен запускаться, когда вы впервые касаетесь мяча своим роботом. Секундомер должен быть остановлен, как только мяч полностью пройдет через последний набор конусов на втором проходе по трассе.

Ответьте на вопросы в своей инженерной тетради:

- Помог ли ваш дизайн улучшить время, необходимое для прохождения трассы? Почему «да» или почему «нет»?
- Что вы можете изменить в своем дизайне, чтобы еще улучшить время? Объясните, как и почему это улучшит его.
- Сталкивались ли вы с чем-то неожиданным при использовании модернизированного робота? Если да, то с чем?

Подготовка и проведение соревнований по Робофутболу

В этом соревновании вы будете использовать вашу скоростную сборку VEX EDR и все вложения, которые вы разработали для игры в футбол против соперника (Рисунок 8).

Рисунок 8. Разметка поля для игры в Робофутбол

Для игры в Робофутбол вам понадобится:

- Один футбольный мяч (или мяч аналогичного размера);
- Три человека (два игрока, один судья);
- Два или более VEX EDR Speed Builds;

• Две одинаковые пустые коробки (достаточно большие, чтобы поместился футбольный мяч);
• Секундомер;
• Открытая площадка со стенками размером не менее 3 x 3 м (рекомендуемый размер) с обеими коробками, расположенными на противоположных концах (подумайте о футбольном поле со стенами!).

В этом соревновании вы будете играть в футбол с вашим VEX EDR Speed Build вместе со всеми модификациями, которые вы, возможно, добавили к нему. Вы будете управлять своим пультом робота с помощью программы **Drive** на вашем контроллере V5.

Правила игры

• Один человек будет судить игру и наблюдать, чтобы каждая команда придерживалась правил.
• Мяч должен быть размещен в центре поля в начале каждой игры и после каждого гола.
• Робот должен стартовать перед мячом со стороны своих ворот.
• Игроки должны стоять за своими воротами на протяжении всей игры.
• Каждая игра длится 5 минут.
• На меньших полях (3 x 3 м или меньше) в каждой команде должен быть только один игрок. Команды могут увеличиваться в размере (2 x 2 или 3 x 3) для больших полей.
• Мяч должен быть полностью в воротах для получения очка.
• Каждый гол стоит 1 балл.
• После того как забит гол, робот или роботы каждой команды должны быть возвращены на место старта и обе команды должны поднять вверх большой палец, чтобы показать рефери, что они готовы начать матч.
• Рефери запустит игру и перезапустит секундомер, сказав «Старт!» в начале игры и после каждого гола.
• Команда с наибольшим количеством очков в конце побеждает!

Учебно-методическое издание

Мацаль Игнатий Игнатьевич

Основы программирования в среде VEXcode V5

Издательство ***«ЭКЗАМЕН»***
«ЭКЗАМЕН-ТЕХНОЛАБ»

Гигиенический сертификат
№ РОСС RU C-RU.АК01.Н.04670/19 с 23.07.2019 г.

Главный редактор *Л. Д. Лаппо*
Корректор *В. В. Кожуткина*
Дизайн обложки
и компьютерная верстка *А. А. Винокуров*

107045, Россия, Москва, Луков пер., д. 8.
E-mail: по общим вопросам: robo@examen-technolab.ru;
www.examen-technolab.ru
по вопросам реализации: sale@examen-technolab.ru
тел. +7 (495) 641-00-23

Общероссийский классификатор продукции
ОК 034-2014; 58.11.1 – книги печатные

Отпечатано в соответствии с предоставленными материалами
в ООО «ИПК Парето-Принт», 170546, Россия, г. Тверь, www.pareto-print.ru