ТЕХНОЛАБ

ОБРАЗОВАТЕЛЬНЫЙ РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)

ОСНОВЫ РОБОТОТЕХНИКИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
К ОБРАЗОВАТЕЛЬНОМУ НАБОРУ
ПО РОБОТОТЕХНИКЕ
«ТЕХНОЛАБ»

(БАЗОВЫЙ УРОВЕНЬ)
ОСНОВЫ РОБОТОТЕХНИКИ

Константин Ермишин
Алексей Панфилов
Сергей Косаченко

ОСНОВЫ РОБОТОТЕХНИКИ

Учебно-методическое пособие
к образовательному набору
по робототехнике
«Технолаб»

ОБРАЗОВАТЕЛЬНЫЙ
РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)
12 – 15 ЛЕТ

ЭКЗАМЕН ТЕХНОЛАБ
Издательство ЭКЗАМЕН®

МОСКВА
2017

УДК 372.8:004
ББК 32.816
Е72

Ермишин К. В., Панфилов А. О., Косаченко С. В.

Е72 Основы робототехники: образовательный робототехнический модуль (базовый уровень): 12 – 15 лет/ Ермишин Константин Владимирович, Панфилов Алексей Олегович, Косаченко Сергей Викторович. — М. : Издательство «Экзамен», 2017. — 160 с.

ISBN 978-5-377-10297-7

Данное пособие предназначено для применения совместно с образовательным робототехническим модулем «Базовый уровень», созданным на базе робототехнического конструктора VEX EDR. В пособии описываются состав и функциональные возможности робототехнического модуля и примеры его применения. Пособие содержит информацию о назначении модуля и элементов, входящих в его состав, а также о возможностях применения данного модуля в образовательном процессе в средних и старших классах.

Образовательная робототехническая платформа VEX EDR представляет собой открытую платформу для создания робототехнических комплексов для образовательной, соревновательной и исследовательской деятельности. В дополнение к этому образовательный робототехнический модуль «Базовый уровень» оснащён программируемым контроллером, представляющим собой открытую программно-аппаратную платформу, преемственную с программируемыми контроллерами типа Arduino. Благодаря этому робототехнический модуль «Базовый уровень» может применяться на стыке двух направлений образовательной деятельности учащихся – реализации творческих инженерных проектов на базе программно-аппаратных платформ открытого типа, а также создания робототехнических комплексов для задач образовательного и соревновательного характера.

Данное пособие носит рекомендательный характер и содержит основную информацию, необходимую для работы с образовательным робототехническим модулем «Базовый уровень», а также типовые примеры по созданию и программированию учебных моделей роботов.

УДК 372.8:004
ББК 32.816

Подписано в печать с диапозитивов 25.01.2017.
Формат 60x90/8. Гарнитура «Calibri». Бумага офсетная.
Усл. печ. л. 20. Тираж 600 экз. Заказ №

ISBN 978-5-377-10297-7

Содержание

Введение

Мировые тенденции развития инженерного образования свидетельствуют о глобальном внедрении информационных технологий в образовательный процесс. Робототехника является весьма перспективной областью для применения образовательных методик в процессе обучения за счёт объединения в себе различных инженерных и естественнонаучных дисциплин. В результате такого подхода наблюдается рост эффективности восприятия информации учащимися за счёт подкрепления изучаемых теоретических материалов экспериментом в междисциплинарной области.

Образовательный робототехнический модуль «Базовый уровень» предназначен для изучения основ робототехники, элементов электроники и микропроцессорной техники, теоретических основ механики и деталей машин, а также программирования микропроцессорных устройств и разработки систем управления роботами.

Помимо применения в образовательных целях, данный модуль в первую очередь ориентирован для применения в робототехнических соревнованиях. Поэтому данный модуль не нацелен на проведение отдельных лабораторных работ по каким-либо направлениям, а предназначен для применения произвольным образом в рамках решения робототехнических задач различной сложности.

В состав модуля входят различные металлические детали, крепёжные элементы, зубчатые передачи и многое другое. Благодаря конструктивным возможностям модуля можно разрабатывать сложные механизмы, состоящие из различных передач и металлических конструкций. С использованием данного модуля также возможно разрабатывать роботов и робототехнические устройства, выполняющие вполне реальные задачи различной сложности, например исследование местности, манипулирование объектами, погрузка и разгрузка грузов, транспортирование объектов, патрулирование территорий и многое другое.

Таким образом, применение данного образовательного робототехнического модуля даёт возможность осуществить плавный переход применения образовательных технологий в области робототехники к полноценной инженерной и проектной деятельности.

§1 СОСТАВ ОБРАЗОВАТЕЛЬНОГО РОБОТОТЕХНИЧЕСКОГО МОДУЛЯ

ЭКЗАМЕН ТЕХНОЛАБ

СОСТАВ ОБРАЗОВАТЕЛЬНОГО РОБОТОТЕХНИЧЕСКОГО МОДУЛЯ

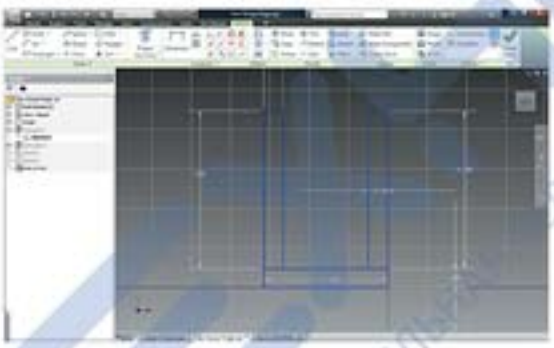
1.1 Состав образовательного робототехнического модуля

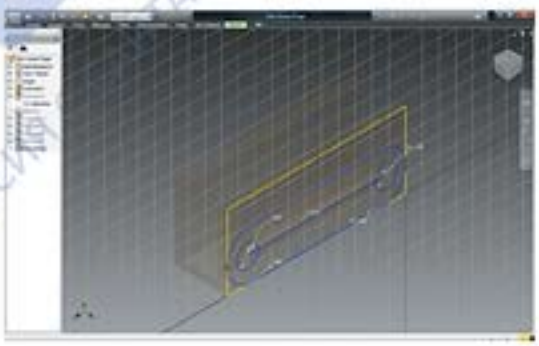
Образовательный робототехнический модуль «Базовый уровень» создан на базе робототехнического конструктора VEX EDR Clawbot и программируемого контроллера, а также комплекта инкрементных энкодеров, набора УЗ-сонаров и ИК-датчиков.

В состав робототехнического модуля входят металлические детали, представляющие собой основные конструктивные элементы робота, предназначенные для сборки различных конструкций и механизмов.

Соединение комплектующих модуля осуществляется с помощью различных крепёжных элементов, а также зубчатых передач. С помощью комплекта зубчатых передач становится возможным создавать подвижные механизмы, выполняющих различные движения робота.

Стоит отметить, что все комплектующие представлены в трёхмерном формате для моделирования и разработки конструкций в специализированных конструкторских программах. Непосредственно со стороны производителя предлагается к применению программное обеспечение компании Autodesk, которое распространяется свободно для применения в учебных и образовательных целях.





С помощью профессионального программного обеспечения можно проектировать различные конструкции роботов, проверять возможность сборки различных элементов, моделировать конструкции роботов на возможность выполнения движений, производить прочностные расчёты конструкций и выполнять множество других инженерных задач. В том числе одной из важных возможностей подобных систем автоматизированного проектирования является автоматизация процесса подготовки конструкторской документации. На основе разработанных моделей роботов можно создать полный комплект чертежей как для отдельных деталей, так и для сборочных единиц.

Применение систем автоматизированного проектирования при решении инженерных задач является одним из важнейших аспектов подготовки специалистов. С их помощью помимо развития профессиональных навыков в области робототехники данные модули становится возможным применять в рамках образовательных программ по технологиям, черчению и т.п.

Для того чтобы разрабатываемые конструкции можно было привести в движение, в состав образовательного модуля входят привода на базе двигателей постоянного тока, которые могут оснащаться инкрементными энкодерами в качестве датчиков положения.

Помимо осуществления различных передвижений зачастую перед роботами ставятся задачи манипулирования различными объектами, для этого применяется захватное устройство, которое можно использовать при конструировании манипуляторов или любых других роботов.

Поскольку роботы и робототехнические устройства большинство своих задач должны выполнять максимально автономно, в состав набора входит комплект сенсорных устройств, предназначенных для оценки состояния окружающей среды.

В состав модуля входят следующие датчики:

комплект ИК-датчиков для обнаружения линии, вдоль которой робот должен осуществлять движения;

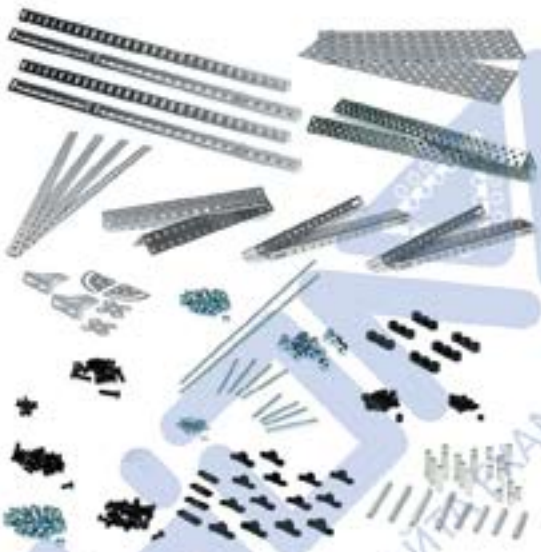
комплект УЗ-сонаров для обнаружения объектов и определения расстояния до них.

Вышеуказанные датчики могут использоваться как по отдельности, в рамках изучения принципов их функционирования и возможностей применения в робототехнике, так и совместно – для создания сенсорных систем робота, позволяющих ему реагировать на изменение окружающей среды. Благодаря этому становится возможным разрабатывать модели роботов, предназначенные для решения практико-ориентированных задач.

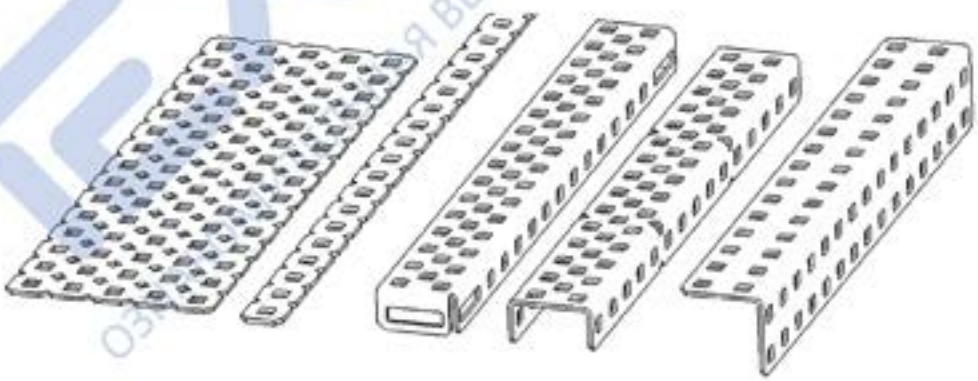
Решение прикладных задач с помощью модуля «Базовый уровень» позволяет повысить качество образовательного процесса за счёт совмещения теории и практики. Применение роботов в соревновательной деятельности и при решении прикладных задач позволяет развивать навыки профессионального проектирования технически сложных систем.

1.2 Конструктивные элементы и комплектующие конструкторов VEX

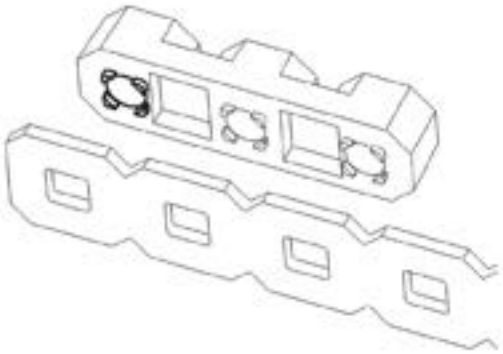
Отличительной особенностью наборов VEX является многообразие различных конструктивных элементов. Конструктивные элементы VEX представляют собой металлические детали и крепёжные элементы, позволяющие разрабатывать на их базе сложные и прочные механизмы.



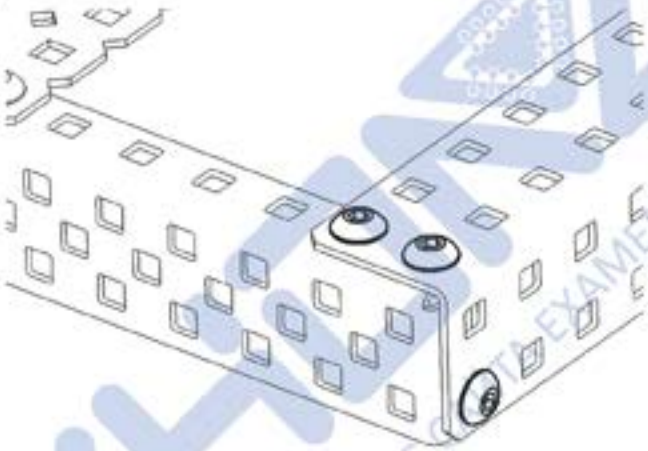
Основными комплектующими VEX являются пластины и уголки из перфорированного алюминия. Благодаря наличию множества отверстий с равным шагом данные детали могут скрепляться друг с другом произвольным образом.



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.

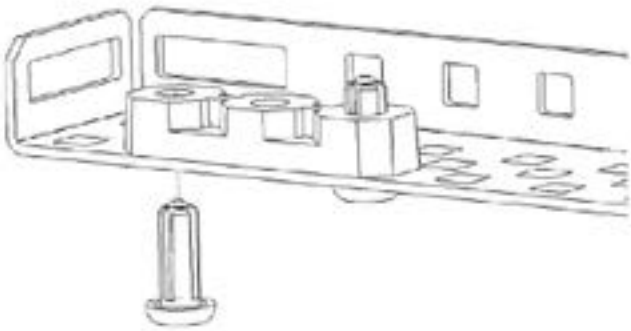


Фиксация элементов осуществляется за счёт специальных бортиков, входящих в углы квадратных отверстий, тем самым задавая ориентацию деталей. В случае закрепления валов или осей квадратные детали прочно и надёжно устанавливаются в соответствующие отверстия.



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.

Также соединение некоторых комплектующих может осуществляться с помощью пластиковых заклепок.

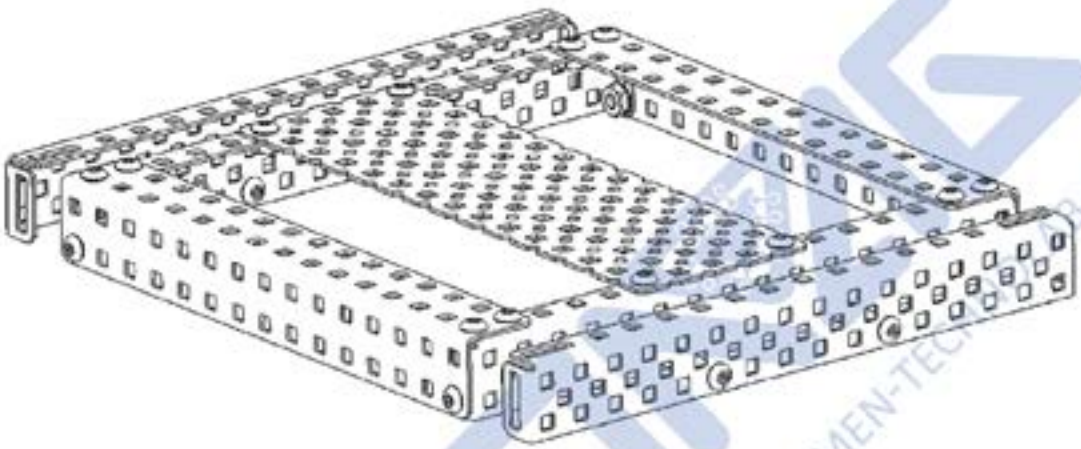


Для закрепления различных конструкций или устройств могут применяться как стандартные детали и крепёжные элементы, так и специализированные стойки. В робототехнический набор VEX входит комплект стоек различной длины.

Одной из ключевых особенностей деталей и комплектующих VEX является то, что любая из плоских пластин может быть согнута или разрезана вдоль специальных направляющих отверстий. Благодаря этому можно создавать детали и конструктивные элементы специальной формы, удовлетворяющей проекту конструкции.

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

Примечание: *При разработке конструкций роботов и прочих механизмов не забывайте о жёсткости конструкции в целом. Для упрочнения конструкции необходимо использовать как стандартные уголки и рёбра жёсткости, так и специализированные элементы.*

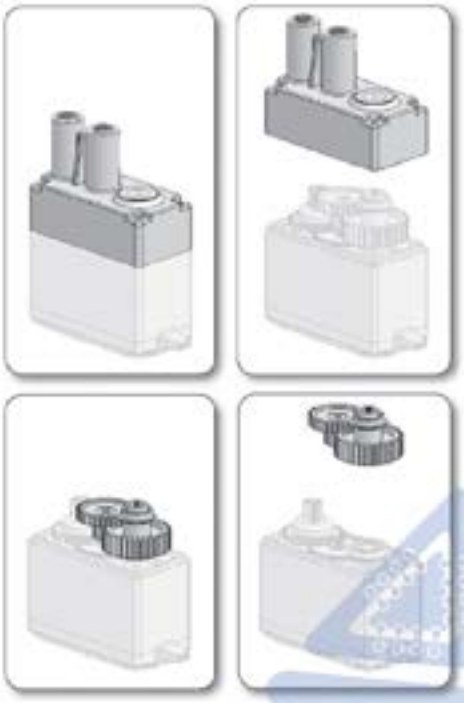


1.3 Исполнительные механизмы конструкторов VEX

Разработка сложных робототехнических устройств и подвижных механизмов помимо надёжных конструктивных элементов требует наличия специализированных исполнительных механизмов, таких как: привода, системы линейного перемещения и элементы зубчатых передач.

В базовый робототехнический набор VEX входят привода и сервопривода на базе двигателей постоянного тока. С помощью данных устройств можно разрабатывать подвижные механизмы и конструкции различного назначения.

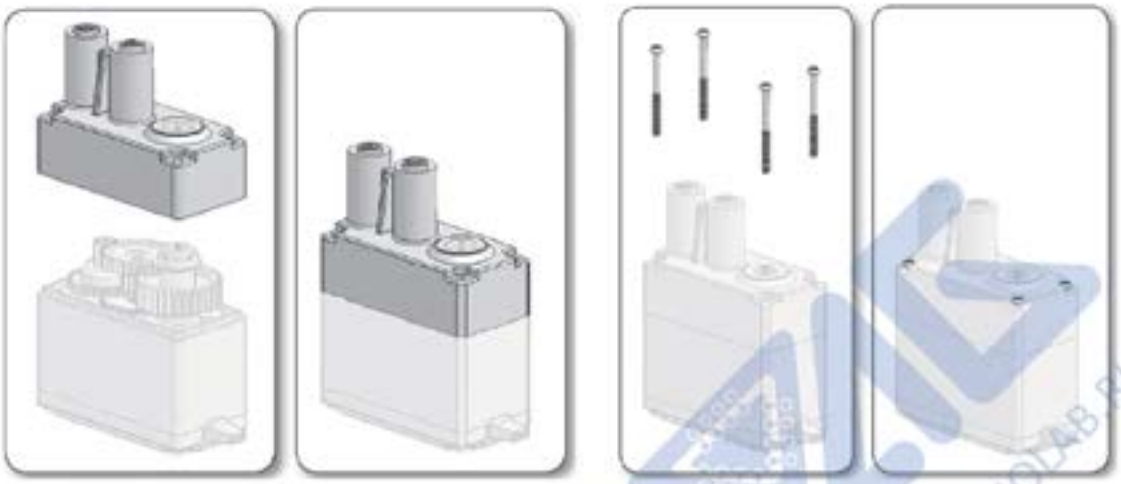
Каждый привод представляет собой электромеханическое устройство, состоящее из двигателя постоянного тока, редуктора и системы управления.



Конструкция каждого из приводов разборная, и пользователь в любой момент может внести изменения в его конструкцию, например, заменить зубчатые колёса редуктора.Замена зубчатых колёс, как правило, осуществляется для изменения передаточного отношения редуктора или с целью ремонта вследствие механической поломки.

Примечание: *При сборке элементов механических зубчатых передач соблюдайте соосность валов и старайтесь не допускать перекосов зубчатых колёс.*

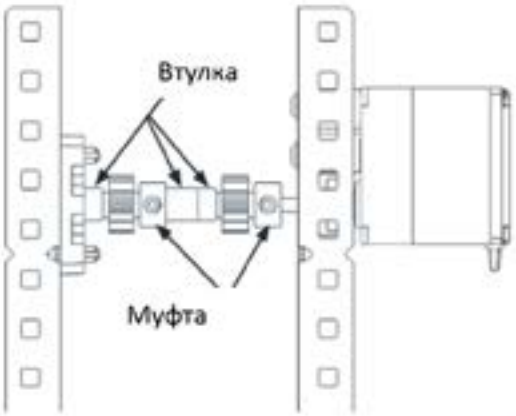
После установки или замены зубчатых колёс необходимо установить крышку, скрывающую механические передачи привода. Крышка обладает специальной конструкцией, благодаря которой привод можно крепить к различным механическим элементам.



Установка привода осуществляется с помощью двух цилиндрических стоек, расположенных на его крышке. Каждая из стоек обладает посадочным фланцем, предназначенным для фиксации в квадратных отверстиях деталей и пластин конструктора VEX. Фиксирование устройства осуществляется с помощью крепёжных винтов.

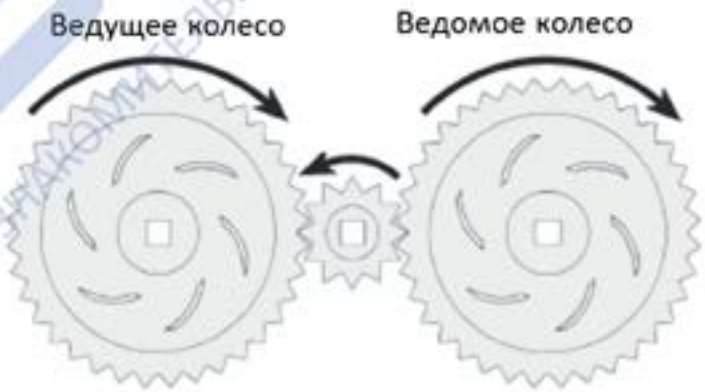
В отличие от большинства аналогичных приводов и сервоприводов, применяемых в робототехнических конструкторах, устройства из базовых робототехнических наборов VEX дают возможность пользователю частично изменять их конструкцию, например, на привода можно устанавливать датчики, определяющие скорость вращения и положение вала; также можно изменять внутреннюю конструкцию редуктора, выходного вала привода и т.п.

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

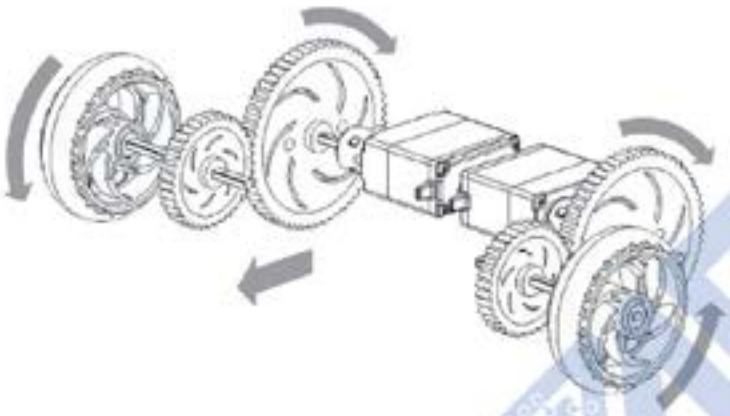


Помимо сменных валов привода могут передавать вращения на различные механизмы, устанавливаемые на валы с помощью фиксирующих втулок. Таким образом можно разрабатывать различные конструкции и механизмы, состоящие из валов и зубчатых передач.

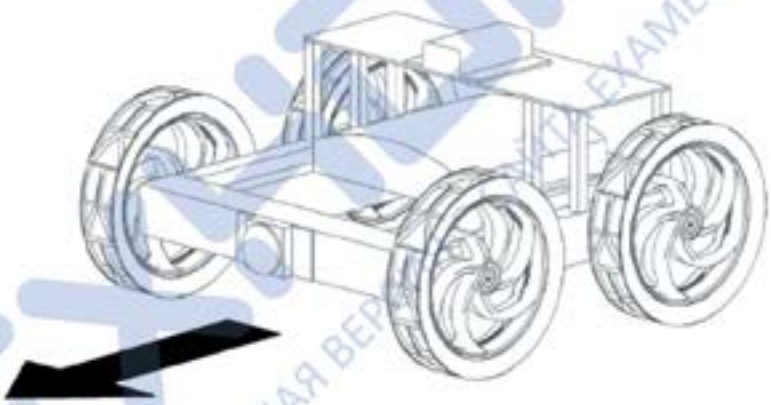
Комбинируя зубчатые колеса различного диаметра, можно конструировать механизмы с различным передаточным отношением, тем самым изменяя скорость вращения и передаваемый ими момент.



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

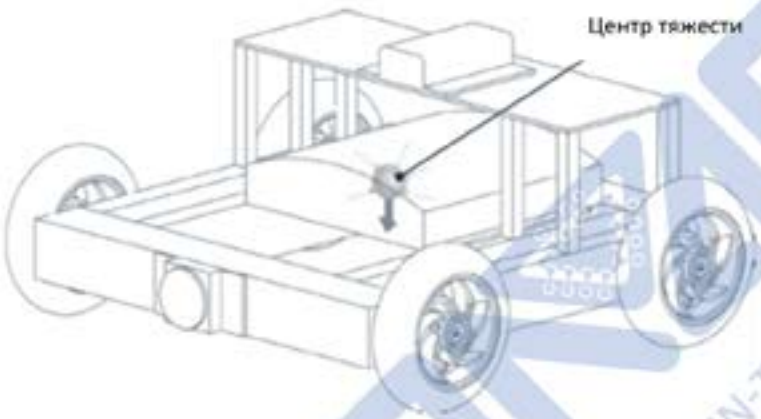


Комплектующие VEX позволяют конструировать различных роботов, решающих широкий спектр задач, начиная от образовательных, исследовательских и соревновательных, вплоть до прикладных задач, решаемых профессиональными роботами.

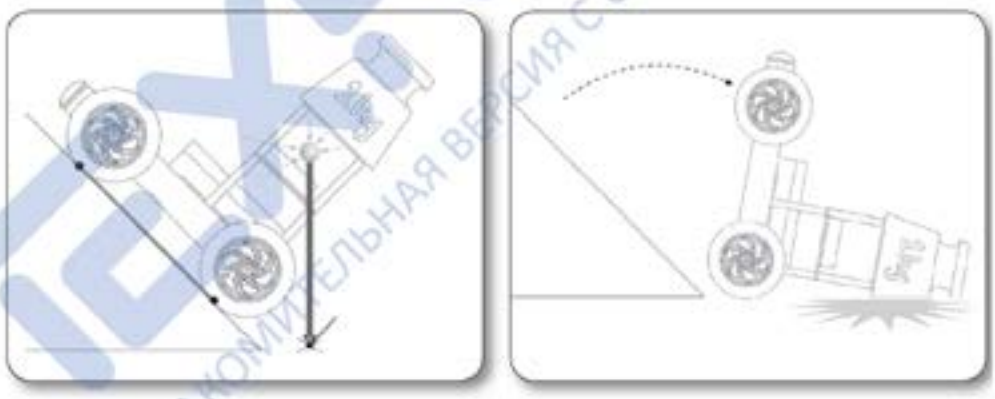


1.4 Базовые принципы проектирования роботов

Робототехнические конструкторы VEX позволяют конструировать достаточно технически сложных роботов, предназначенных для решения сложных задач. В зависимости от сложности робота и решаемых им задач все больше внимания следует уделять надёжности роботов, прочности их конструкций и т.п.



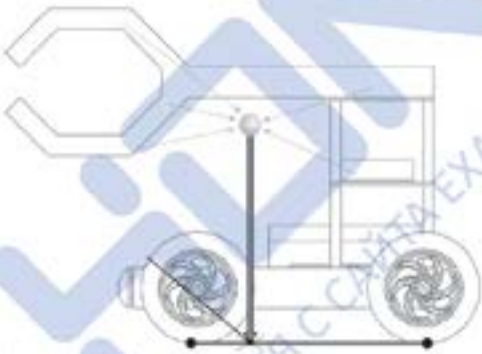
Одно из основных требований к мобильным роботам – это сохранение их устойчивости в процессе движения или работы. В процессе проектирования следует уделять внимание балансировке механизмов робота и равномерному распределению нагрузки по всей конструкции.



В зависимости от решаемых роботом задач необходимо проектировать конструкцию таким образом, чтобы её особенности не препятствовали выполнению основных функций робота.

Например, при перемещении робота по наклонным поверхностям необходимо смещать центр тяжести робота как можно ниже к основанию и к его центру, чтобы препятствовать возможному опрокидыванию при подъёме.

При проектировании роботов, оснащённых захватным устройством, необходимо учитывать возможное смещение центра тяжести робота при манипулировании объектами.

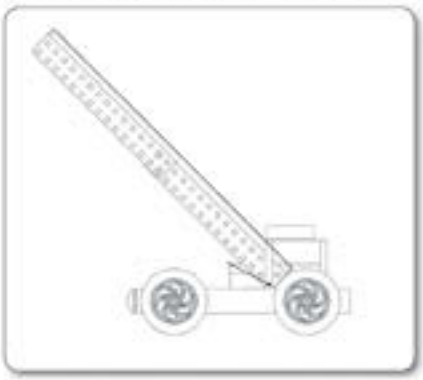


В подобной ситуации следует усовершенствовать либо конструкцию робота, либо умышленно смещать его центр тяжести с целью балансировки конструкции.

Следует уделять повышенное внимание вопросам распределения нагрузки равномерно по всему роботу. Плохо сбалансированная конструкция робота подверже-

на излишним нагрузкам и имеет склонность к заносам и опрокидываниям.Одним из наиболее действенных приёмов балансировки роботов является распределение аккумуляторных батарей по шасси так, чтобы они уравновешивали конструкцию необходимым образом.

При проектировании роботов часто возникают ситуации, когда конструкция или механизмы робота обладают достаточно большими габаритами, в результате чего его собственная конструкция может быть неустойчивой или неуравновешенной.



Для того чтобы конструкция робота сохраняла жёсткость и устойчивость, необходимо устанавливать рёбра жёсткости, поддерживающие основные конструктивные элементы.

При проектировании робота необходимо достичь сохранности всех его внутренних узлов и устройств в процессе работы. Необходимо следить за тем, чтобы никакие управляющие электронные устройства не выступали за габариты робота, чтобы не допустить их повреждения. Также важно следить за тем, чтобы соединяющие кабели и шлейфы не перетирались в процессе движение робота или его механизмов.

Помните, что в независимости от сложности робота и решаемых им задач, процесс проектирования робота должен быть одинаково ответственным. Однако при росте сложности и числа решаемых задач, возлагаемых на робота, необходимо учитывать как можно больше влияющих факторов и заранее прогнозировать результаты работы проектируемого робота. Подобные навыки проектировщиков развиваются исключительно с ростом их опыта работ в конкретной области. Образовательный робототехнический модуль «Базовый уровень» содержит в себе всё необходимое для развития профессиональных навыков проектирования роботов и робототехнических систем.

1.5 Программируемый контроллер

В состав робототехнического модуля «Базовый уровень» входит программируемый контроллер, преемственный контроллерам семейства Arduino. Контроллеры семейства Arduino представляют собой программно-аппаратный комплекс на базе платы ввода/вывода с использованием микроконтроллеров семейства AVR (Atmel) , а также специальной среды программирования на основе языка C/C++.

Отличительная особенность контроллеров семейства Arduino заключается в том, что каждый контроллер данного семейства обладает унифицированным расположением портов и единым стандартом разъёмов, встроенной схемой электропитания, позволяющей использовать большинство распространённых аккумуляторных батарей, а также встроенными средствами для программирования. Несмотря на небольшие базовые функциональные возможности, контроллеры семейства Arduino обладают многообразием дополнительных плат расширения (Shield), позволяющих подключать к базовой плате различные привода, датчики, устройства связи и мультимедийные системы.

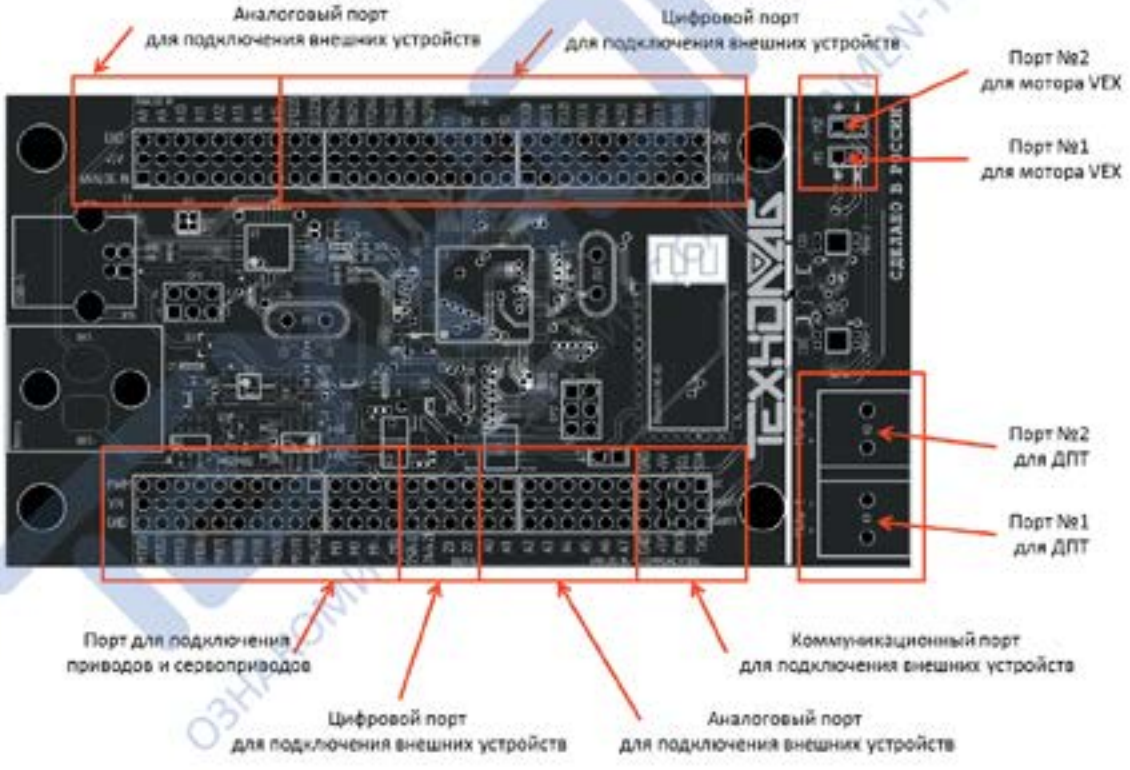
Благодаря этому контроллеры семейства Arduino на сегодняшний день являются одними из наиболее популярных устройств для быстрого прототипирования инженерных проектов.

Семейство контроллеров Arduino выпускается с открытой документацией, благодаря чему любой пользователь или производитель может создать собственный контроллер или аксессуар, преемственный к оригинальным контроллерам Arduino. Единственное ограничение касается торговой марки Arduino – вновь созданные проекты имеют право указывать на собственную преемственность Arduino, но не имеют права использовать это название в своей продукции.

На сегодняшний день в мире существует множество различных Arduino-подобных контроллеров, каждый из которых копирует оригинальные платы. В силу ограниченности функциональных возможностей плат Arduino без дополнительных внешних устройств они не могут в полной мере обеспечить все потребности разработчиков роботов. Чаще всего при разработке роботов возникает необходимость подключения силовой нагрузки (чаще всего двигателя), обеспечения контроля за уровнем питания, а также обеспечения беспроводной связи с внешними элементами систем управления робота.

В состав образовательного робототехнического модуля «Базовый уровень» входит программируемый контроллер, преемственный контроллерам семейства Arduino и специально созданный для применения с робототехническими конструкторами VEX.

Программируемый контроллер является преемственным по отношению к оригинальной плате Arduino Mega 2560, но в отличие от неё обладает встроенной системой контроля заряда внешней аккумуляторной батареи, интегрированной силовой частью для подключения внешних двигателей постоянного тока, встроенным модулем беспроводной связи Bluetooth, а также полной совместимостью с комплектующими VEX.



Программируемый контроллер содержит:

1) 2 силовых порта (с аппаратной поддержкой PWM) для подключения двигателей постоянного тока (ДПТ) и параллельные им порты для подключение моторов, входящих в состав робототехнического конструктора VEX.

2) 26 цифровых портов ввода/вывода (с нумерацией от 0 до 25) для работы с внешними дискретными сигналами.

3) 16 аналоговых портов (с нумерацией от 0 до 15) для взаимодействия с внешними аналоговыми устройствами и сигналами.

4) 14 портов (10 портов с аппаратным PWM и 4 с GPIO) для подключения внешней нагрузки, приводов/моторов и сервоприводов робототехнической платформы VEX EDR.

5) 2 коммуникационных порта для взаимодействия по последовательному интерфейсу UART.

6) 1 коммуникационный порт для взаимодействия по последовательному интерфейсу I2C.

7) 1 встроенный контроллер интерфейса для беспроводной связи Bluetooth.

8) 1 встроенный USB порт для программирования контроллера и передачи данных (UART).

9) 1 встроенный порт для подключения внешнего питания и схему контроля состояния заряда аккумуляторной батареи.

Все порты программируемого контроллера преемственны портам оригинальной платы Arduino. Нумерация портов программируемого контроллера выполнена по мере увеличения их порядкового номера, но нумерация не всех портов совпадает с нумерацией оригинальной платы. В случае если нумерация портов совпадает, то порты маркируются идентично, например, как в случае с аналоговыми портами программируемого контроллера и оригинальной платы Arduino. Если же нумерация портов не совпадает, то при маркировке порта программируемого контроллера в скобках указывается соответствующая маркировка порта оригинальной платы Arduino.

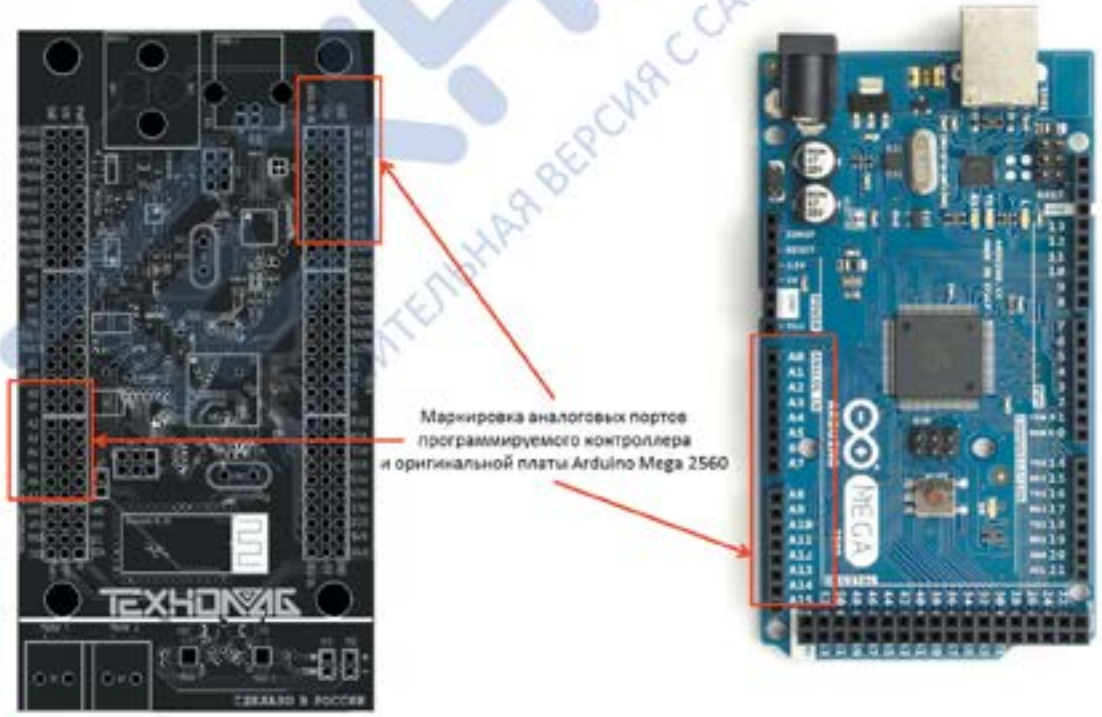


Таблица соответствия маркировки портов			
Тип порта	Маркировка портов		Примечания
	Программируемый контроллер	Arduino Mega 2560	
Коммуникационный	UART1	UART1	Четырёхпиновый разъём с питанием +5В и GND
	UART3	UART3	
Коммуникационный	I2C	I2C	Четырёхпиновый разъём с питанием +5В и GND
Аналоговый AIN	0-15	0-15	Трёхпиновый разъём с питанием +5В и GND
Цифровой GPIO	0	48	Трёхпиновый разъём с питанием +5В и GND
	1	49	
	2	37	
	3	36	
	4	35	
	5	34	
	6	33	
	7	32	
	8	31	
	9	30	
	10	резерв (PJ3)	
	11	резерв (PJ4)	
	12	резерв(PJ5)	
	13	резерв(PJ6)	
	14	29	
	15	28	
	16	27	
	17	26	
	18	25	
	19	24	
	20	23	
	21	22	
	22	резерв(ICP1)	
	23	резерв(ICP3)	
	24	42	
	25	43	

Цифровой PWM	M 0	38	Трёхпиновый разъём с питанием +5В и GND
	M 1	44	
	M 2	45	
	M 3	46	
	M 4	12	
	M 5	11	
	M 6	10	
	M 7	9	
	M 8	8	
	M 9	7	
	M 10	6	
	M 11	3	
	M 12	2	
	M 13	5	
Силовой порт M1	M1 +	6	Двухпиновый разъём VCC и GND
	M1 -	7	
Силовой порт M2	M2 +	46	Двухпиновый разъём VCC и GND
	M2 -	45	
Коммуникационный	Bluetooth	UART2	Беспроводной интерфейс

Для взаимодействия с дополнительной вычислительной платой часть портов зарезервированы как системные. Соответствующие цифровые порты отмечены в таблице соответствия маркировки портов.

§2 РАБОТА С ОСНОВНЫМИ УСТРОЙСТВАМИ И КОМПЛЕКТУЮЩИМИ

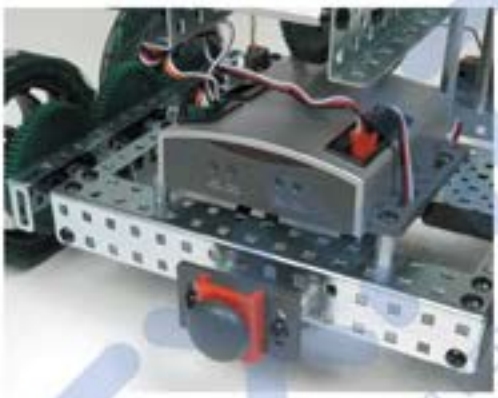
ЭКЗАМЕН ТЕХНОЛАБ

РАБОТА С ОСНОВНЫМИ УСТРОЙСТВАМИ И КОМПЛЕКТУЮЩИМИ

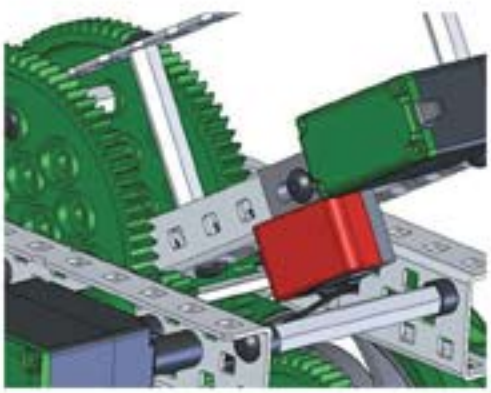
2.1 Пример подключения и работы с тактильными датчиками, концевыми выключателями и кнопками

Тактильные датчики или концевые переключатели являются одним из простейших типов датчиков, представляющих собой механический переключатель, замыкающий или размыкающий цепь управления.

Такие устройства представляют собой наиболее простейшие цифровые датчики, которые генерируют дискретный сигнал в зависимости от собственного состояния. Наиболее часто встречается применение подобных датчиков в качестве кнопок управления или ограничивающих устройств.



Тактильные датчики чаще всего применяются в качестве кнопок или контактных бамперов безопасности, останавливающих движение робота в запрещённом направлении в случае нажатия на них.



Концевые переключатели применяются чаще всего для срабатывания в случае соприкосновения с различными движущимися частями или механизмами робота, например: в случае ограничения угла поворота манипулятора и т.п.

При использовании концевых выключателей особое внимание необходимо уделять их расположению вблизи подвижных частей механизма. Датчик должен быть установлен таким образом, чтобы в требуемом положении на него оказывалось необходимое давление для срабатывания.

Примечание: *Для срабатывания концевого выключателя необходимо развить усилие, эквивалентное массе 0,25 г, на расстоянии 2 см от оси вращения.*

Каждый из рассматриваемых датчиков подключается к цифровым входам программируемого контроллера. Оба датчика идентичны друг другу в плане применения в процессе разработки программы управления. Показания датчиков носят логический характер – «Нажат» или «Не нажат», т.е. «1» или «0».

Подключение подобных датчиков может осуществляться к любому свободному цифровому порту. При нажатии кнопки происходит замыкание контакта на землю, что приводит к падению уровня сигнала на выбранном цифровом порту до низкого логического уровня.

Пример:

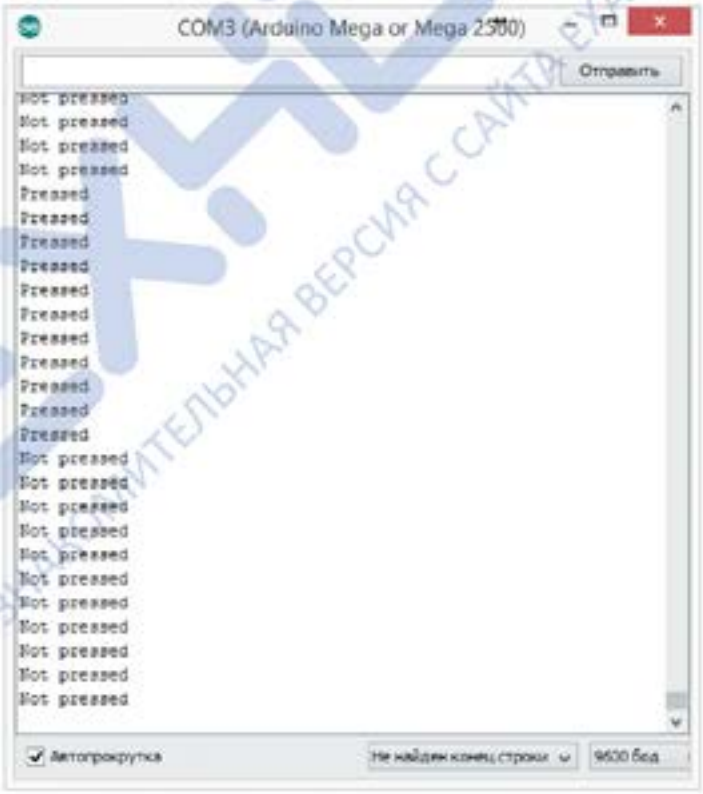
Подключим простую кнопку к порту 0(48) таким образом, чтобы белый провод (сигнальный) смотрел внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//указываем порт, к которому подключена кнопка - 0(48)
const int buttonPort = 48;
//объявляем переменную для работы с кнопкой
int buttonState = 0;

void setup(){
//конфигурируем выбранный порт на чтение
pinMode(buttonPort, INPUT);
//инициализируем соединение с ПК по последовательному порту
//со скоростью 9600 бод
Serial.begin(9600);
}
```

```
void loop(){
//подаём высокий уровень сигнала в выбранный цифровой порт
digitalWrite(buttonPort, HIGH);
//читаем уровень сигнала на выбранном цифровом порту
buttonState=digitalRead(buttonPort);
//выводим в последовательный порт сообщение
в зависимости от состояния кнопки
//если не нажата – замыкания на «землю» нет и на порту
держится высокий уровень
if (buttonState == HIGH)
Serial.println("Not pressed");
else
//если нажата – происходит замыкание на «землю» и уровень
становится низким
Serial.println(«Pressed»);
//задаём паузу между опросами порта в 50 мс
delay(50);
}
```

После загрузки скетча в плату в окне монитора последовательного порта можно будет увидеть результат опроса кнопки.



2.2 Пример подключения и работы с датчиком освещённости

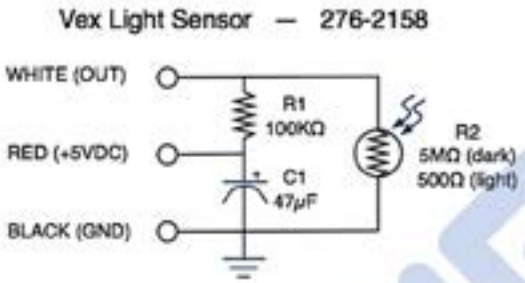
Датчик освещённости предназначен для измерения интенсивности дневного освещения и позволяет определять интенсивность светового потока, благодаря чему можно существенно расширить функциональные возможности роботов.

Датчик освещённости даёт роботу дополнительный источник информации об окружающей среде и позволяет реализовывать алгоритмы автономной работы. Например, датчик освещённости может быть использован для перемещения робота за источником света.

Также датчик освещённости может быть использован в качестве защиты от проникновения робота в труднодоступные места с ограниченной видимостью.

Проявив фантазию, датчик освещённости можно применять для создания алгоритмов энергосбережения заряда аккумулятора, например, для автоматического отключения робота в тёмное время суток и т.п. Если использовать внешние дополнительные световые фильтры, с помощью датчика освещённости становится возможным различать цвета, благодаря чему робот может манипулировать объектами разного цвета.

Датчик освещённости использует в качестве чувствительного элемента фоторезистор на базе материала CdS, который меняет собственное сопротивление в зависимости от интенсивности светового потока.



Фоторезистор является аналоговым датчиком, плавно изменяющим сопротивление в зависимости от интенсивности светового потока, что влечёт за собой изменения напряжения в диапазоне от 0 до 5В на выходных клеммах датчика освещённости.

Программируемый контроллер робота считывает данные от датчика и после АЦП-преобразования выдаёт результат в виде целого числа в диапазоне от 0 до 1023. Причём значение 1023 соответствует минимальному уровню освещённости (темнота), а значение 0 – максимальному уровню освещённости.

Примечание: В зависимости от разрядности АЦП-преобразования результат измерений датчика может быть: в диапазоне от 0 до 255 в случае 8-разрядного преобразования, в диапазоне от 0 до 1023 в случае 10-разрядного преобразования, в диапазоне от 0 до 4095 в случае 12-разрядного преобразования.

Датчик освещённости реагирует на видимый человеческим глазом свет, а его чувствительность позволяет определять интенсивность светового потока от объекта на расстоянии 1,5 – 2 метра.

Поскольку датчик освещённости является устройством аналогового типа, его следует подключать к аналоговым входам программируемого контроллера. Как и любой датчик, входящий в состав образовательного модуля, данный датчик имеет специальный разъём для подключения к аналоговым портам контроллера, благодаря чему его можно подключать к клеммам контроллера, не опасаясь за полярность.

Пример:

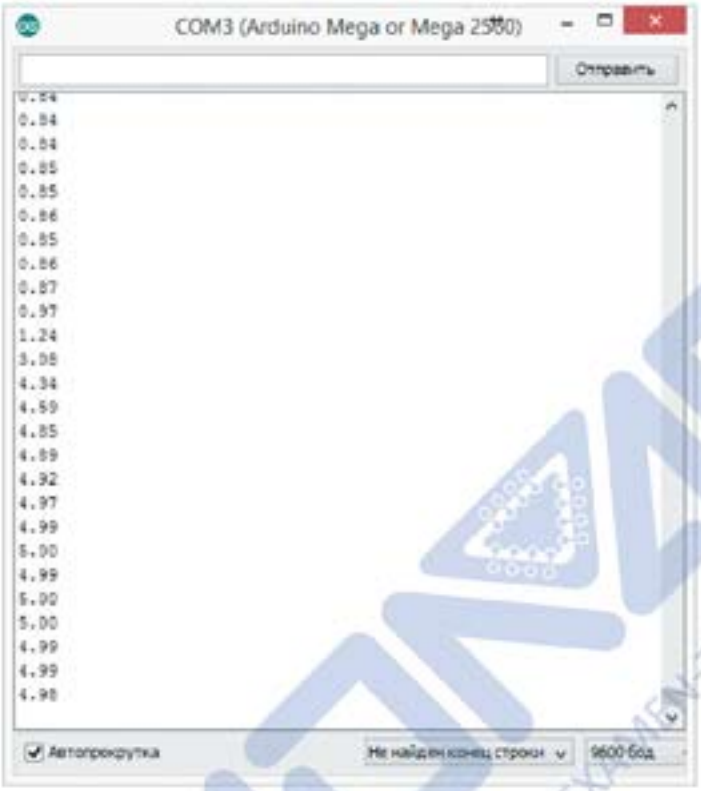
Подключение датчика освещённости выполняется к любому аналоговому порту. Датчик освещённости построен на основе фоторезистора, что позволяет фиксировать не только наличие/отсутствие освещённости, но и определять степень освещённости. Чем ниже освещённость – тем выше напряжение на выходе датчика.

Подключим датчик освещённости к порту A0 таким образом, чтобы белый (сигнальный) провод «смотрел» внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//указывает порт, к которому подключён датчик освещённости:
int sensorPort = A0;
//объявляем переменную для работы с датчиком освещённости
int sensorValue=0;

void setup() {
//инициализируем соединение с ПК по последовательному порту
Serial.begin(9600);
}

void loop() {
//считываем значения с аналогового порта
sensorValue = analogRead(sensorPort);
//выводим в последовательный порт полученные значения
//с конвертацией в вольты
Serial.println(sensorValue*0.0049);
//задаем паузу между опросами порта в 50 мс
delay(50);
}
```

После загрузки скетча в плату в окне монитора последовательного порта можно будет увидеть результат опроса датчика освещённости.



2.3 Пример подключения и работы с ИК-датчиком линии

Датчик определения линии представляет собой инфракрасный датчик, определяющий интенсивность отражённого от рабочей поверхности света.

Датчик предназначен для определения чёрной линии на белом фоне, благодаря чему робот может автономно перемещаться вдоль неё. Подобная технология достаточно часто встречается в транспортно-логистических системах в цехах производственных предприятий и даёт возможность минимизировать ручной труд водителей транспортных средств.

Датчик состоит из ***ИК-светодиода*** и ***ИК-датчика***, реагирующего на интенсивность отражённого света. Принцип функционирования данного датчика основывается на различии отражающих способностей поверхностей разного цвета. Наиболее светлые поверхности отражают падающий на них свет, тёмные поверхности его полностью поглощают.

При проектировании роботов, движущихся вдоль чёрной линии, применяется как минимум два датчика, определяющих положение линии по правому и левому борту робота. Для более точной работы робота следует применять не меньше трёх датчиков, а именно: два датчика по бокам робота и один над направляющей линией.

Подключение детектора линии может быть выполнено как к цифровому, так и к аналоговому порту.

В случае подключения детектора линии к цифровому порту, детектор используется для определения чёрной линии на белом фоне: при попадании на чёрную линию на выходе детектора будет высокий уровень сигнала (есть линия), при попадании на белый фон – низкий уровень (нет линии). Однако, при необходимости можно использовать детектор для различения тёмных оттенков. В таком случае подключение и опрос детектора линии выполняется к аналоговому порту с полной аналогией подключения и опроса датчика освещённости.

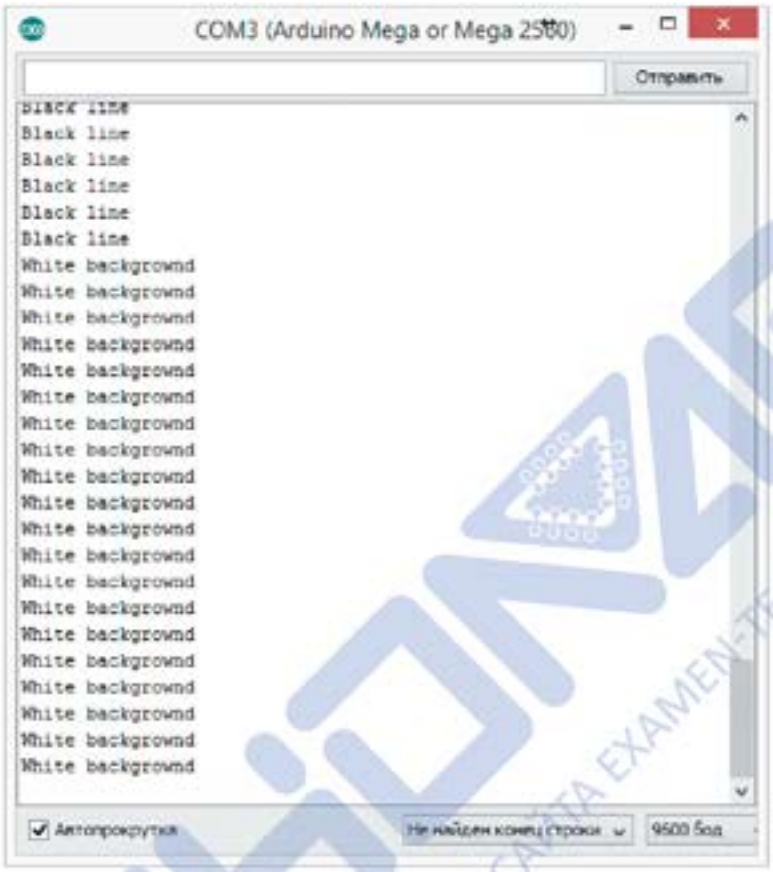
Пример подключения детектора в качестве детектора чёрной линии на белом фоне:

Подключим детектор к порту 0(48) таким образом, чтобы белый провод (сигнальный) смотрел внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//указываем порт, к которому подключён детектор - 0(48)
const int lineTrackerPort = 48;
//объявляем переменную для работы с детектором
int lineTrackerState = 0;

void setup(){
//конфигурируем выбранный порт на чтение
pinMode(lineTrackerPort, INPUT);
//инициализируем соединение с ПК по последовательному порту
//со скоростью 9600 бод
Serial.begin(9600);
}

void loop(){
//читаем уровень сигнала на выбранном цифровом порту
lineTrackerState=digitalRead(lineTrackerPort);
//выводим в последовательный порт сообщение
в зависимости от состояния детектора
//если попал на чёрную линию, уровень становится высоким
if (lineTrackerState == HIGH)
Serial.println("White backgrownd");
else
//если попал на белый фон, на выходе держится низкий уровень
Serial.println(«Black line»);
//задаем паузу между опросами порта в 50 мс
delay(50);
}
```

После загрузки скетча в плату в окне монитора последовательного порта можно будет увидеть результат опроса детектора линии.



2.4 Пример подключения и управления моторами

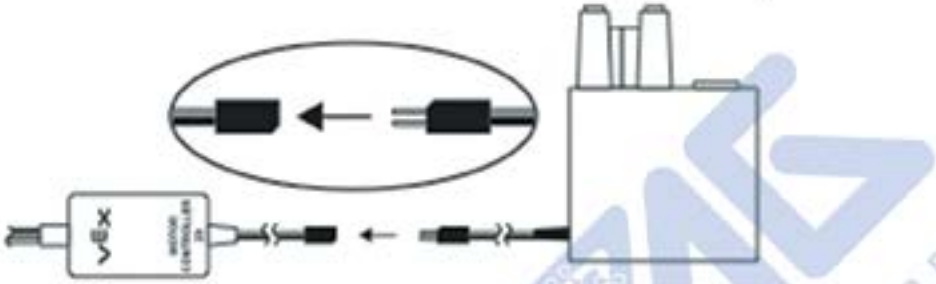
Привод 2-Wire Motor 269 w является двигателем коллекторным постоянного тока с редуктором, состоящим из металлических зубчатых передач. В комплект к двигателю входит набор винтов для его крепления и вал для передачи вращения от двигателя к исполнительному механизму.

Данный привод является наиболее распространённым в роботах на базе наборов из комплектующих VEX, поскольку Motor 269 обладает наиболее оптимальным соотношением технических характеристик и габаритов, а также может быть оснащён внешним инкрементным энкодером. При стандартном для комплектующих VEX питании в 7,2В данный привод развивает момент 0,97 Нм при скорости до 100 об/мин. Поскольку в процессе работы привод потребляет ток величиной до 2,6А, для подключения его к программируемому контроллеру необходимо применять силовую схему (драйвер двигателя).

Устройство Motor Controller 29 представляет собой силовую схему для управления приводами на базе двигателей постоянного тока. С помощью Motor Controller 29 мож-

но подключать к программируемому контроллеру привода с максимальным рабочим током, не превышающим 3А. При подключении стандартного привода к программируемому контроллеру становится возможным регулировать скорость его вращения с помощью программируемого значения выходного ШИМ сигнала контроллера.

Для подключения между собой устройства 2-Wire Motor 269 и Motor Controller 29 необходимо соединить с помощью двухпроводного разъёма.



При соединении устройств между собой следует обращать внимание на полярность каждого из них. В случае, если устройства были соединены с нарушением взаимной полярности, вращение привода будет происходить в противоположном от заданного программой направления. Для исправления данной ошибки достаточно осуществить подключение устройств заново или изменить направление вращения привода в программе.

Подключение устройства Motor Controller 29 к программируемому контроллеру осуществляется с помощью специально отведённых портов котроллера для работы с приводами.

Управление приводом осуществляется за счёт изменение мощности на выходном канале ШИМ программируемого контроллера. Изменение сигнала ШИМ осуществляется программным образом, для этого выбирается порт, к которому подключён привод, и ему присваивается значение в диапазоне от -127 до 127, что соответствует максимальным оборотам привода в прямом и обратном направлении.

Пример подключения и управления мотором через драйвер мотора:

Выполним соединение мотора и драйвера мотора таким образом, чтобы чёрный провод мотора совпал с чёрным проводом драйвера, а красный провод драйвера совпал с красным проводом мотора. Затем подключим 3-штырьковый разъём к порту M13(5) таким образом, чтобы белый (сигнальный) провод «смотрел» внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//подключаем библиотеку для работы с моторами по PWM шине
#include <Servo.h>

//инициализируем наш мотор присвоением ему имени
Servo myservo;

void setup() {
//указываем порт, к которому подключён мотор M13(5)
myservo.attach(5);
}

void loop() {
//задаём ширину импульсов управляющей последовательности
в диапазоне 1000 - 2000 мс.
//значение 1500 мс означает остановку двигателя
myservo.writeMicroseconds(1300);
//задаём паузу в 2 секунды
delay (2000);
//останавливаем мотор
myservo.writeMicroseconds(1500);
//задаем паузу в 2 секунды
delay (2000);
}
```

После загрузки скетча в плату мотор начнёт вращаться в течение 2 секунд с 2-секундными перерывами.

Пример подключения управления мотором без использования внешнего драйвера мотора:

Выполним подключение двухпинового разъёма мотора к порту M1. При подключении необходимо соблюдать полярность.

Напишем в среде Arduino IDE следующий скетч:

```
//указываем пины, отвечающие за управление портом M1
const int motorPinOne = 6;
const int motorPinTwo = 7;

void setup() {
//конфигурируем пины порта M1 на вывод сигнала
```

```
pinMode(6, OUTPUT);
pinMode(7, OUTPUT);
}

void loop() {
//устанавливаем различные уровни на пинах порта -
//мотор начинает вращение
digitalWrite (motorPinOne, HIGH);
digitalWrite (motorPinTwo, LOW);
//ждём 2 секунды
delay(2000);
//устанавливаем одинаковые уровни на пинах порта -
//мотор останавливается
digitalWrite (motorPinOne, LOW);
digitalWrite (motorPinTwo, LOW);
//ждём 2 секунды
delay(2000);
//меняем уровни на пинах порта на противоположные -
//мотор меняет направление вращения
digitalWrite (motorPinOne, LOW);
digitalWrite (motorPinTwo, HIGH);
//ждём 2 секунды
delay(2000);
//устанавливаем одинаковые уровни на пинах порта -
//мотор останавливается
digitalWrite (motorPinOne, LOW);
digitalWrite (motorPinTwo, LOW);
//ждём 2 секунды
delay(2000);
}
```

После загрузки скетча в плату мотор начинает вращение, затем через 2 секунды останавливается на 2 секунды и затем снова начинает вращение, но в другую сторону. Затем через 2 секунды он снова останавливается и снова меняет направление своего вращения.

2.5 Пример подключения и управления сервоприводом

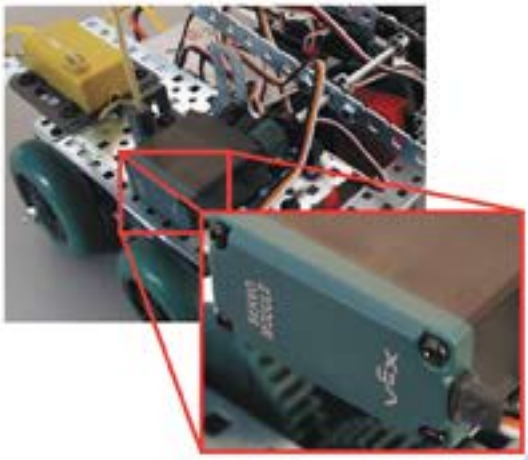
Сервопривод является специализированным устройством, предназначенным для осуществления точных перемещений исполнительных механизмов роботов и робототехнических устройств.

Сервопривод является устройством на базе привода постоянного тока и схемы управления, определяющей положение выходного вала с помощью датчика обратной связи. Чаще всего в качестве датчиков обратной связи применяются потенциометры, имеющие ограниченный угол поворота, в данном случае рабочий угол сервопривода лежит в пределах 100 градусов.

Управление положением выходного вала сервопривода осуществляется с помощью ШИМ-сигнала, т.е. угол поворота привода изменяется пропорционально частоте генерации ШИМ. Трёхпроводный интерфейс сервопривода содержит: чёрный провод – «земля», оранжевый провод – «питание», белый провод – ШИМ. Благодаря этому сервопривод подключается к программируемому контроллеру с помощью трёхпроводных разъёмов, как обычный привод.

В процессе работы сервопривод может развивать момент до 0,73 Нм при рабочем токе в диапазоне от 20мА-1,5А. Вращение сервопривода возможно как в прямом, так и в обратном направлении в пределах рабочего угла.

Обычно сервоприводы применяются в конструкциях и механизмах, где необходимо осуществлять точные перемещения в ограниченном диапазоне, например, захватные устройства механизмов, манипуляторы, поворотные основания и т.п.



Способ присоединения сервопривода идентичен обычному приводу, поэтому они могут применяться в одинаковых конструкциях или заменять друг друга.

Управление сервоприводом сводится к заданию его конечной координаты в пределах допустимого диапазона от -127 до 127.

Пример подключения и управления сервоприводом:

Подключим сервопривод к порту M13(5) таким образом, чтобы белый (сигнальный) провод смотрел внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//подключаем библиотеку для работы с сервоприводами по PWM шине
#include <Servo.h>
//инициализируем наш сервопривод присвоением ему имени
Servo myservo;

void setup() {
//указываем порт, к которому подключён сервопривод M13(5)
myservo.attach(5);
}

void loop() {
// устанавливаем сервопривод в положение 0 градусов
myservo.write(0);
//ждём 1 секунду
delay(1000);
// устанавливаем сервопривод в положение 180 градусов
myservo.write(180);
//ждём 1 секунду
delay(1000);
 }
```

После загрузки в плату сервопривод начнёт с периодичностью в 1 секунду менять своё положение из 0 градусов в 180 и обратно.

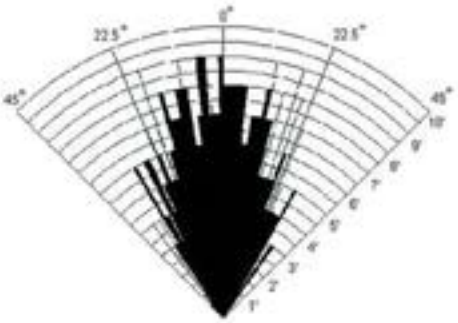
2.6 Пример подключения и работы с УЗ-сонаром

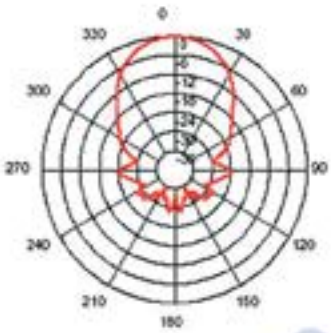
Ультразвуковой датчик является дальномером, измеряющим расстояние до объектов с помощью отражённого от поверхности объекта ультразвукового сигнала.

В отличие от тактильных датчиков и концевых выключателей данное устройство может заблаговременно предупредить о приближении робота к объекту, благодаря чему робот может передвигаться в среде с различными препятствиями и планировать свой маршрут.

Примечание: *Следует обращать внимание на то, что отражающая способность поверхности влияет на чувствительность датчика и точность измерений. Помимо этого существенное влияние на процесс измерения расстояния до объекта оказывает его форма и взаимное расположение с роботом.*

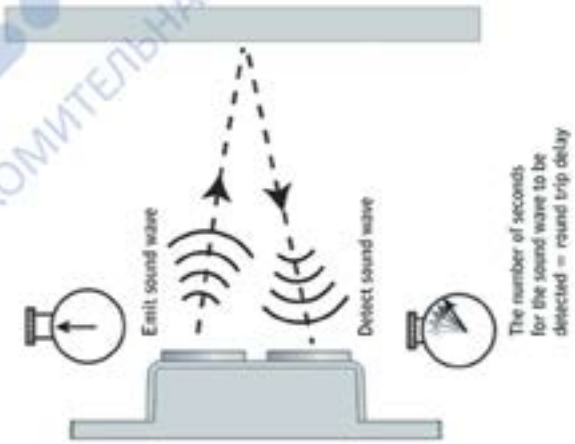
Ультразвуковой дальномер обладает диаграммой направленности, т.е. рабочей зоной, в пределах которой он может обнаруживать объекты. Если какой-либо объект находится в пределах диаграммы направленности , то он будет обнаружен данным датчиком и до него будет измерено расстояние. Точность измерения расстояния зависит от разрешающей способности датчика и его взаимного расположения с объектом.





Для расширения области примения ультразвукового датчика и увеличения его зоны сканирования можно установить его на поворотное основание. В этом случае, вращая датчик, можно обозревать окружающее пространство в более широком секторе, тем самым имея возможность обнаруживать объекты по обе стороны от робота.

Ультразвуковой дальномер определяет расстояние до препятствия за счёт вычисления времени полёта звуковой волны с момента её распространения и до возвращения после отражения от объекта. Излучение звуковой волны осуществляется с частотой 40 кГц и позволяет измерять расстояние в диапазоне от 3 см до 3 м.



Алгоритмически процесс работы датчика делится на ряд отдельных этапов. На первом этапе датчик отправляет звуковую волну и начинает отсчёт времени. После того как датчик принимает отражённый сигнал, отсчёт времени прекращается. Расстояние до объекта рассчитывается как скорость распространения волны, умноженная на затраченное время.

Подключение ультразвукового датчика расстояния к программируемому контроллеру выполняется с помощью двух 2-пиновых коннекторов: INPUT и OUTPUT. Вывод INPUT отвечает за подачу управляющего импульса на датчик, вывод OUTPUT необходим для отправки полученного, результирующего импульса.

Пример подключения и опроса ультразвукового датчика расстояния:

Подключим вывод OUTPUT к порту M11(3) таким образом, чтобы оранжевый провод смотрел внутрь платы. Вывод INPUT подключим к порту M12(2) так, чтобы жёлтый провод смотрел внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//указываем порт, к которому подключён вход INPUT
датчика расстояния M12(2)
const int Trig = 2;
//указываем порт, к которому подключён выход OUTPUT
датчика расстояния M11(3)
const int Echo = 3;

void setup() {
//настраиваем порт входа датчика расстояния на вывод сигнала
pinMode(Trig, OUTPUT);
//настраиваем порт выхода датчика расстояния на ввод сигнала
pinMode(Echo, INPUT);
//инициализируем соединение с ПК по последовательному порту
//со скоростью 9600 бод
Serial.begin(9600);
}

//объявляем и инициализируем переменную для работы с временем
unsigned int time_us=0;
//объявляем и инициализируем переменную для работы с расстоянием
unsigned int distance_sm=0;

void loop() {
// подаём сигнал на вход датчика
digitalWrite(Trig, HIGH);
// удерживаем его 10 микросекунд
delayMicroseconds(10);
```

```
// убираем сигнал со входа датчика
digitalWrite(Trig, LOW);
// замеряем длину импульса на выходе датчика
time_us=pulseIn(Echo, HIGH);
// пересчитываем полученные данные в сантиметры
distance_sm=time_us/58;
// выводим в последовательный порт полученные значения расстояния
Serial.println(distance_sm);
//пауза между опросами датчика расстояния - 100 мс
delay(100);
}
```

После загрузки скетча на плату в окне монитора последовательного порта можно будет увидеть результат опроса датчика расстояния в сантиметрах.

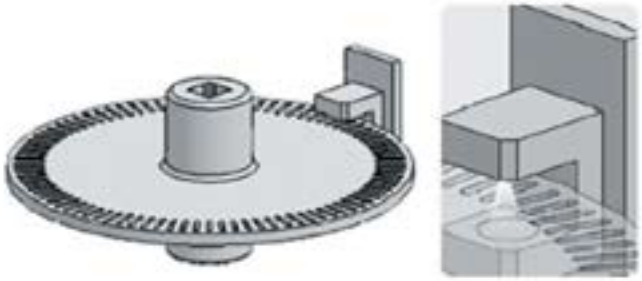
2.7 Пример подключения и работы с оптическим энкодером

Данный датчик представляет собой оптический квадратурный энкодер, предназначенный для установки на подвижные части и механизмы роботов с целью определения их положения.

Оптический энкодер может измерять скорость вращения и количество оборотов вала привода, тем самым позволяет регулировать скорость движения роботов и дальность их перемещения.

Конструкция датчика позволяет ему принимать вращение от сменных валов и осей, благодаря чему его можно устанавливать в произвольных местах и использовать для определения параметров вращения не только приводов, но и непосредственно исполнительных механизмов робота.

Оптический энкодер представляет собой датчик, состоящий из ИК-излучателя и приёмника, расположенных по обе стороны от диска с прорезями. В процессе работы излучатель генерирует световой поток, который либо проходит сквозь прорези в диске, либо отражается от него. В случае, если световой поток проходит сквозь прорези диска, он попадает на приёмник, и датчик выдаёт свидетельствующий об этом сигнал.



Считая количество подобных прерываний и зная шаг последовательности прорезей на диске и его размеры, можно определить угол поворота вала привода. В свою очередь, проведя аналогичные расчёты за единицу времени, можно определить скорость вращения вала привода.

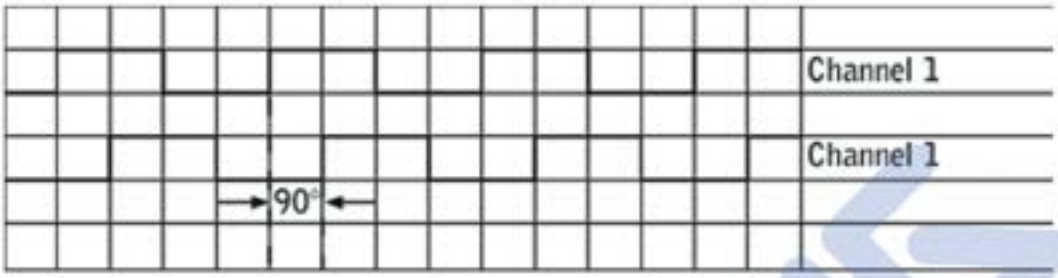
Примечание: *Точность определения скорости вращения привода и его положения определяется количеством прорезей на диске оптического энкодера. Оптические энкодеры являются относительными датчиками, т.е. они позволяют вычислить угол поворота оси относительно какого-то начального положения.*

Благодаря использованию оптического энкодера можно определить перемещение робота, вычислив путь, пройденный его колёсами. Для того чтобы вычислить пройденный колесом робота путь, необходимо умножить его периметр на число совершённых им оборотов.

Таким образом, точность перемещения робота зависит от разрешающей способности оптического энкодера и диаметра колеса робота. Очевидно, что при равной точности показаний датчика точность перемещений робота будет в случае использования колёс с минимальным диаметром.

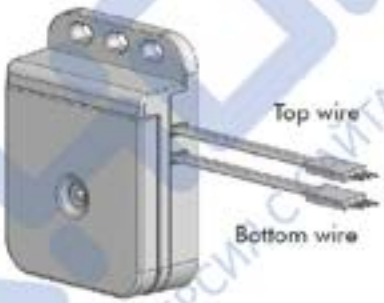
С помощью оптического энкодера можно вести подсчёт до 1700 импульсов в секунду, что соответствует 1133 оборотам в минуту. Не рекомендуется применять данный датчик на более высоких скоростях вращения, т.к. это может привести к ошибкам измерения положения или скорости.

Оптический квадратурный инкрементный энкодер является цифровым двухканальным датчиком. Благодаря тому, что конструкция датчика содержит два приёмника отражённого света, смещённых относительно друг друга, в результате измерений датчик выдаёт два меандра – последовательности прямоугольных импульсов, смещённых по фазе относительно друг друга.



Каждый из каналов датчика содержит последовательность импульсов амплитудой 0 или 5В в зависимости от того, в каком положении находится диск с прорезями относительно приёмника отражённого света.

С помощью двухканального энкодера можно определить направление вращения вала. В случае если первый канал опережает по фазе второй канал, то вращение вала происходит по часовой стрелке. В противоположном случае вращение происходит в обратном направлении.



Поскольку диск энкодера вращается с большой скоростью и частота следования импульсов достаточно велика, для того чтобы в процессе вычислений программируемый контроллер робота не пропустил ни одного из импульсов, датчики следует подключать к специальным портам контроллера, выделенным под внешние прерывания.

Подключение энкодера выполняется с помощью двух 3-пиновых выводов, которые подключаются к цифровым портам аппаратной платформы. Результатом работы энкодера является получение меандра - сигнала, в котором содержится информация о вращении диска энкодера.

Пример подключения и опроса сдвигового энкодера:

Выполним подключение обоих выводов сдвигового энкодера к портам 0(48) и 1(49) таким образом, чтобы белые (сигнальные) провода смотрели внутрь платы.

Напишем в среде Arduino IDE следующий скетч:

```
//укажем порты, к которым подключены выводы сдвигового энкодера
enum { ENC_PORT1 = 48, ENC_PORT2 = 49 };

void setup(){
 //сконфигурируем порты на ввод сигналов
 pinMode(ENC_PORT1, INPUT);
 pinMode(ENC_PORT2, INPUT);
 //инициализируем соединение с ПК по последовательному порту
//со скоростью 9600 бод
 Serial.begin(9600);
}

/* Функция декодирования кода Грея (см. Википедию).
* Принимает число в коде Грея,
возвращает обычное его представление.
*/
unsigned graydecode(unsigned gray){
 unsigned bin;
 for (bin = 0; gray; gray >>= 1)
 bin ^= gray;
 return bin;
}

void loop(){
 //объявим и проинициализируем переменную
для предыдущего считанного кода
 static uint8_t previous_code = 0;
 /* gray_code - считанное с энкодера значение
 * code - декодированное значение
 */
 //считываем значение кода Грея с выводов и декодируем его
 uint8_t gray_code = digitalRead(ENC_PORT1) | (digitalRead(ENC_
PORT2) << 1),
 code = graydecode(gray_code);
 /* Если считался нуль, значит, был произведён щелчок
ручкой энкодера */
 if (code == 0)
 {
 /* Если переход к нулю был из состояния 3 - ручка вращалась
* по часовой стрелке, если из 1 - против.
*/
 if (previous_code == 3)
 Serial.println("->");
```

```
   else if (previous_code == 1)
   Serial.println("<-");
   }
  /* Сохраняем код и ждём 1 мс - вполне достаточно опрашивать эн-
кодер
   * не более 1000 раз в секунду.
   */
   previous_code = code;
   delay(1);
  }
```

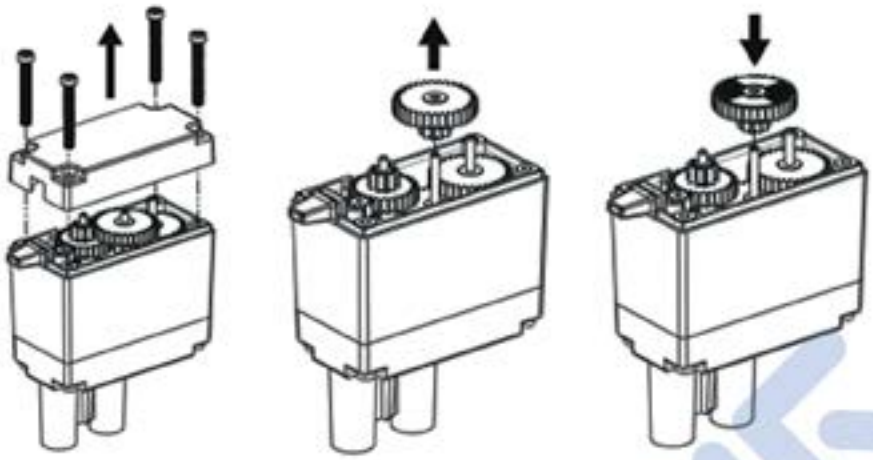
После загрузки скетча на плату в окне монитора последовательного порта можно будет увидеть результат опроса энкодера в виде указания направления вращения его диска.

2.8 Пример подключения и работы с инкрементным энкодером

Устройство ***Motor 269 Integrated Encoder Module*** представляет собой встраиваемый в обычный привод инкрементный энкодер, позволяющий определять скорость и направление вращения привода.

Инкрементный энкодер представляет собой оптический датчик, содержащий диск с различной отражающей способностью. Данный диск располагается напротив ИК-излучателя и в процессе своего вращения прерывает поток излучения, в результате чего становится возможным определить скорость вращения двигателя по интенсивности импульсов, приходящих с датчика.

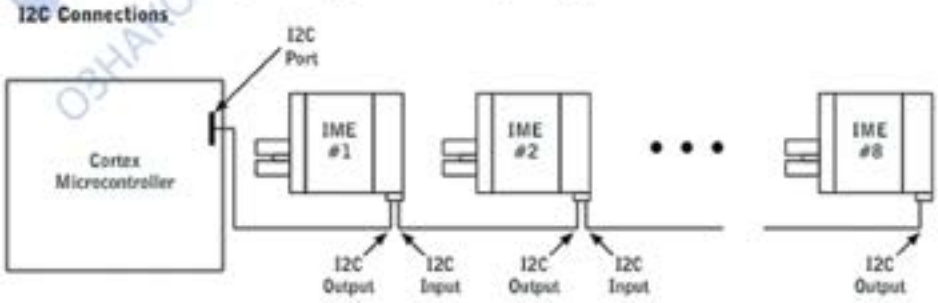
Для установки датчика необходимо открутить 4 винта крышки привода и снять её, после чего необходимо установить отражающий диск вместо шестерни, как это показано на рисунке.



После замены шестерёнки необходимо удостовериться в том, что выходной вал привода прокручивается без заклинивания. Далее можно установить взамен старой крышки привода новую с выходными каналами для энкодера.

В завершение необходимо окончательно удостовериться в правильности сборки привода и проверить его на плавность вращения и отсутствие заклиниваний.

Устройство Motor 269 Integrated Encoder Module подключается к программируемому контроллеру с помощью интерфейса I2C. Данный интерфейс позволяет подключать к одной шине все энкодеры, применяемые в роботе разом, т.е. до 8 штук в соответствии с общим числом портов управления приводами.



Подключение приводов к шине I2C осуществляется последовательным образом, т.е. путём соединения выхода первого устройства со входом второго и т.д. Важно заметить, что последовательность подключения энкодеров не обязательно должна совпадать с последовательностью подключения приводов к портам программируемого контроллера. Так, например, первый привод может быть подключён к порту №1, второй же к порту №10, но энкодеры будут все равно подключены последовательно друг к другу.

Подключение интегрируемого энкодера выполняется по I2C шине одним 4-пиновым проводом.

Для работы с этим энкодером необходимо использование сторонней библиотеки для Arduino IDE - I2CEncoder.h.

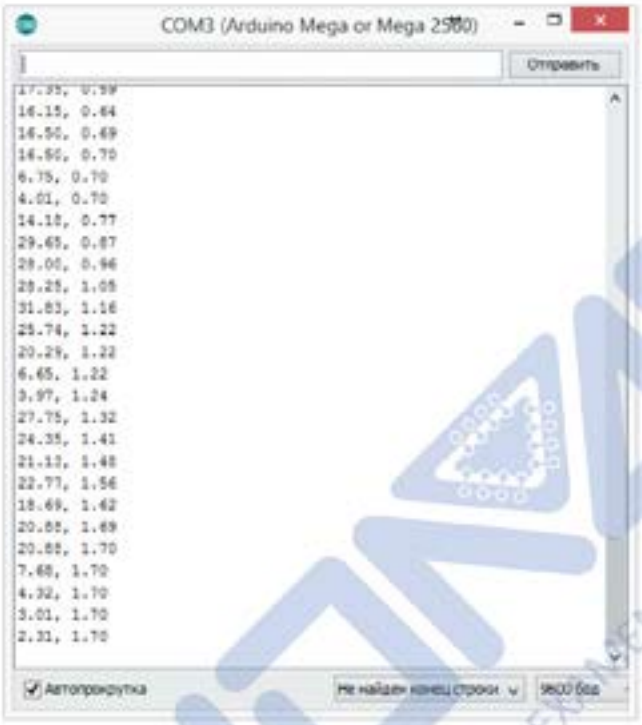
Пример подключения и опроса интегрируемого энкодера:

Подключим интегрируемый энкодер к I2C порту аппаратной платформы так, чтобы вывод GND соответствовал чёрному проводу вывода энкодера, а SDA – белому.

Напишем в среде Arduino IDE следующий скетч:

```
//подключение библиотеки для работы с i2c шиной
#include <Wire.h>
//подключение библиотеки для работы с I2C энкодером
#include <I2CEncoder.h>
//объявляем энкодер путём присвоения ему имени
I2CEncoder encoder;

void setup() {
// инициализируем подключение по i2c шине
Wire.begin();
//инициализируем соединение с ПК по последовательному порту
 //со скоростью 9600 бод
 Serial.begin(9600);
 //инициализируем энкодер, установленный на 269 моторе
 encoder.init(MOTOR_269_ROTATIONS, MOTOR_269_TIME_DELTA);
}
void loop() {
//получаем значение скорости в оборотах в минуту
//и отправляем данные в последовательный порт
Serial.print(encoder.getSpeed());
Serial.print(“, “);
//получаем значение положения в оборотах
Serial.println(encoder.getPosition());
//пауза между опросами энкодера 200 мс
delay(200);
}
```

После загрузки скетча на плату в окне монитора последовательного порта можно будет увидеть результат опроса интегрируемого энкодера в виде скорости вращения мотора и его положения.



2.9 Работа со встроенным Bluetooth-модулем

В программируемом контроллере имеется встроенный Bluetooth-модуль, позволяющий организовывать обмен данными между платой и мобильным устройством, таким как планшет. Для использования Bluetooth-модуля необходимо снять на аппаратной платформе перемычку AT и перезапустить платформу путём отключения питания от неё. Затем на мобильном устройстве необходимо выполнить обычным поиском ближайшие Bluetooth-устройства и выполнить подключение к устройству HC-05. При запросе PIN-кода используйте «0000» или «1234».

Пример организации обмена данными между ПК и мобильным устройством:

Напишем в среде Arduino IDE следующий скетч:

```
void setup()
{//устанавливаем соединение с bluetooth-модулем
на скорости 115200
Serial2.begin(115200);
//устанавливаем соединение с персональным компьютером
на скорости 9600
Serial.begin(9600);
}
void loop()
{
//если на модуль пришли данные со стороны мобильного устройства
if(Serial2.available())
{ //читаем их
byte a=Serial2.read();
//отправляем на ПК
Serial.write(a);
}
//если пришли данные со стороны ПК
if(Serial.available())
{//читаем их
byte a=Serial.read();
//отправляем на ПК
Serial2.write(a);
}
}
```

После загрузки скетча на плату, подключившись к Bluetooth-модулю с мобильного устройства и открыв на компьютере монитор последовательного порта, можно обмениваться данными. Однако важно заметить, что Bluetooth-модуль может быть настроен с помощью AT-команд. С помощью этих AT-команд можно изменить скорость обмена данными модуля, его имя, пин-код, режим работы и т.д.

Примеры АТ команд:

```
AT проверить статус подключения
AT+VERSION? узнать версию прошивки.
AT+UART? узнать установленную скорость.
AT+UART=38400, 0,0 установить скорость 38400.
AT+NAME? узнать имя.
AT+NAME=HC-05 BLUE установить имя HC-05 BLUE.
AT+PSWD? узнать пароль.
AT+PSWD=0000 установить пароль 0000.
AT+ORGL сброс на заводские настройки
```

Для перехода в режим настройки модуля необходимо загрузить скетч (обязательно необходимо изменить скорости обмена на 38400), представленный выше, установить перемыкатель АТ и перезагрузить платформу. После чего открыть монитор последовательного порта, выбрать скорость 38400, режим «Оба NL & CR» вместо «Не найден конец строки» и отправить команду «АТ». Если в ответ придёт «ОК», то модуль успешно вошёл в режим настройки АТ-командами.

Пример настройки модуля в качестве ведомого (slave):

```
Сброс предыдущих настроек: AT+ORGL
Сброс спаренных устройств: AT+RMAAD
Установка пароля: AT+PSWD=1234
Включение режима ведомого: AT+ROLE=0
Дополнительно можно узнать адрес устройства
(понадобится для настройки спаренного модуля): AT+ADDR?
В ответ получим сам адрес: ADDR=xx:x:xxxxxx
```

Пример настройки модуля в качестве ведущего:

```
Сброс предыдущих настроек: AT+ORGL
Сброс спаренных устройств: AT+RMAAD
Включение режима ведущего: AT+ROLE=1
Рестарт после смены роли: AT+RESET
```

Если необходимо связать ведомого и ведущего, используются команды:

```
Установка пароля ведомого: AT+PSWD=1234
Указываем парное устройство: AT+PAIR=<адрес>,<таймаут>
(пример: AT+PAIR=12,6,143117, 5)
Связываем с конкретным адресом: AT+BIND=<адрес>
(пример: AT+BIND=12,6,143117)
Запрещаем соединяться с другими адресами: AT+CMODE=0
```

После настройки отключаем перемыкатель АТ и перезапускаем модуль

ЭКЗАМЕН ТЕХНОЛАБ

РАЗРАБОТКА МАКЕТА РОБОТА

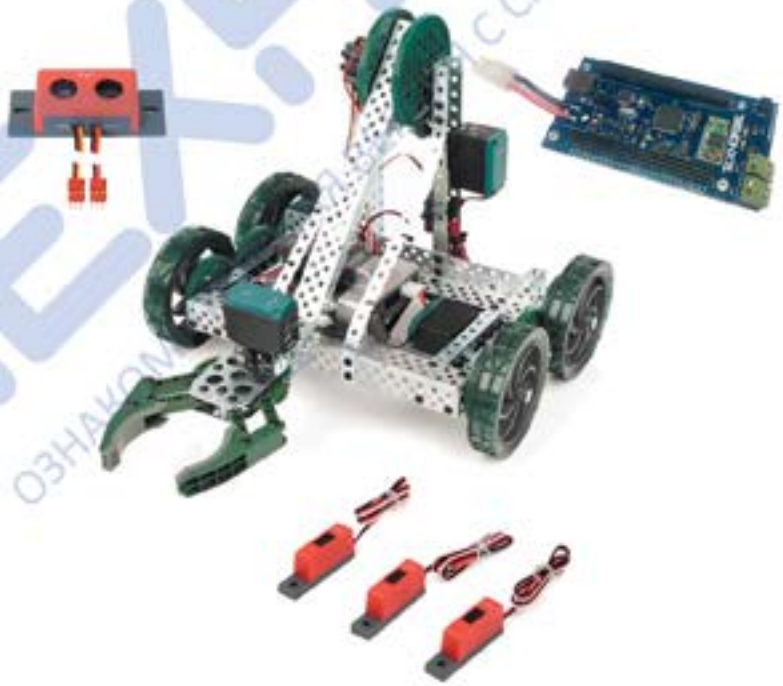
§3 Разработка макета робота

В рамках процесса разработки действующего макета робота предлагается решить последовательно ряд следующих задач:

1. Обеспечить подвижность робота (вперёд-назад, повороты).
2. Обеспечить движение манипулятора (хватательное + выброс).
3. Обеспечить опрос ультразвукового дальномера и начать его использование совместно с работой манипулятора (убирать препятствие на пути робота).
4. Обеспечить опрос трёх детекторов линии и использовать их при движении робота.
5. Объединить полученные решения в одну программу, решающую задачу движения робота по линии и убирания препятствия на пути.

Сам робот ClawBot собирается из входящих в состав набора компонентов, по входящей в набор инструкции, однако приведённые ниже примеры разработаны для слегка модифицированного робота – три детектора линии установлены на нижней части схвата, а ультразвуковой дальномер установлен непосредственно на мотор схвата. В некоторых главах крепление детекторов линии и дальномера могут предлагаться иначе.

Питание аппаратной платформы, моторов и компонентов робота осуществляется от стандартного аккумулятора VEX, имеющего 7,2В на выходе. После сборки робота и установки на него программируемого контроллера можно приступать к решению задач по обеспечению подвижности робота.



3.1 Движение робота вперёд-назад и осуществление поворотов

В качестве моторов для движения робота используются два двухпроводных мотора 393, установленные на левой и правой частях платформы соответственно. Подключение и управление ими осуществляется напрямую от аппаратной платформы, без использования внешних драйверов мотора. Для подключения моторов к аппаратной платформе необходимо соединить кабель правого мотора с клеммой M2, а левого с клеммой M1 аппаратной платформы с соблюдением полярности.

К каждой клемме подключено два порта с поддержкой ШИМ (PWM).

Направление вращения каждого мотора управляется подачей на один порт высокого уровня (HIGH), и на второй порт низкого уровня (LOW). Например, чтобы левый мотор начал вращаться, можно запрограммировать следующие команды:

```
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
```

А чтобы левый мотор начал вращаться в обратную сторону, потребуются поменять уровни на соответствующих портах следующими командами:

```
digitalWrite (motorM1PinOne, HIGH);
digitalWrite (motorM1PinTwo, LOW);
```

Обратите внимание, что в данном примере мы не управляем плавно мощностью двигателей (про это вы узнаете в следующей главе), а даем команды мотору вращаться в нужном направлении на максимально возможной мощности.

Применим эти команды для управления движением роботом Clawbot вперед-назад и влево-вправо по таймингу, т.е. задав направление движения робота и время для выполнения маневра.

Загрузите в контроллер из Arduino IDE следующий скетч:

```
//указываем пины, отвечающие за управление портом M1
const int motorM1PinOne = 6;
const int motorM1PinTwo = 7;

//указываем пины, отвечающие за управление портом M2
const int motorM2PinOne = 46;
const int motorM2PinTwo = 45;

void setup() {
//конфигурируем пины порта M1 на вывод сигнала
pinMode(motorM1PinOne, OUTPUT);
pinMode(motorM1PinTwo, OUTPUT);
//конфигурируем пины порта M2 на вывод сигнала
pinMode(motorM2PinOne, OUTPUT);
pinMode(motorM2PinTwo, OUTPUT);
```

```
//едем вперед в течение 2х секунд
forvard ();
delay(2000);
//поворот направо
right ();
delay(1000);
//поворот налево
left ();
delay(1000);
//едем назад в течение 2 секунд
reverse ();
delay(2000);
//останавливаемся
stopMove ();
}

void loop() {
}

void forvard (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}

void right (){
//правый мотор
digitalWrite (motorM2PinOne, HIGH);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}

void left (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, HIGH);
digitalWrite (motorM1PinTwo, LOW);
}
```

```
void reverse (){
//правый мотор
digitalWrite (motorM2PinOne, HIGH);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, HIGH);
digitalWrite (motorM1PinTwo, LOW);
}
void stopMove (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, LOW);
}
```

После загрузки на плату робот:

1) начнёт движение вперёд и будет двигаться в этом направлении в течение 2 секунд,

2) затем начнёт двигаться сначала по часовой стрелке,

3) затем против часовой стрелки,

4) после чего поедет назад и остановится.

В будущем для выполнения поворотов на определённый угол необходимо будет изменять значение времени, в течение которого робот двигается по часовой или против часовой стрелки, например:

```
//поворот направо
right ();
delay(1000); - в этом месте надо указать необходимое время
движения в миллисекундах
```

Такое действие необходимо, поскольку в данном примере не используются энкодеры.

3.2 Движение робота и повороты по энкодерам

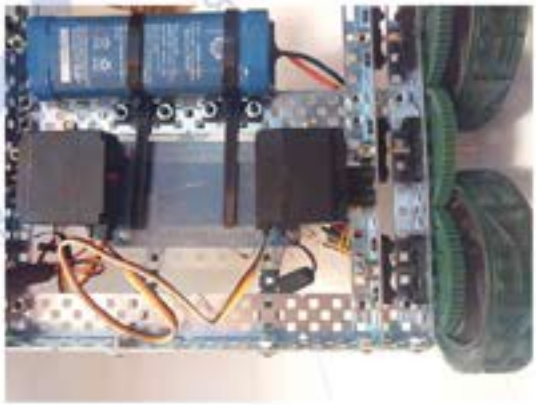
Одометрия – использование данных о движении приводов для оценки перемещения.

Как мы уже рассмотрели в предыдущей главе, моторы VEX 393, вращающие колеса робота ClawBot, можно дополнительно оборудовать инкреметными энкодерами и последовательно подключить к шине I2C. Это позволит точнее программировать контроллер Технолаб (совместим с Arduino Mega) на прохождение роботом заданного расстояния, при этом точнее выдерживать прямолинейное движение и осуществлять повороты на заданный угол.

Пример последовательного подключения двух энкодеров ***(Motor 393 Integrated Encoder Module)*** по шине I2C и получения от них информации.

Подключите первый (левый) энкодер к I2C порту контроллера так, чтобы вывод GND соответствовал черному проводу, а SDA — белому.

Второй (правый) энкодер подсоедините четырехжильным кабелем к первому энкодеру.



Для корректной работы I2C шины, необходимо, чтобы все устройства имели уникальные I2C-адреса, но при подаче питания на энкодеры Motor 393 Integrated Encoder

Module, адреса у них будут совпадать, что вызовет конфликты в шине I2C. Для правильной инициализации двух последовательно соединенных инкрементных энкодеров, нужно следовать следующим шагам:

1. Инициализируем первый (левый) в цепочке энкодер с автоматическим назначением ему нового уникального адреса:

```
encoderLeft.init(MOTOR_393_TORQUE_ROTATIONS, MOTOR_393_TIME_DELTA);
```

2. Теперь укажем, что этот энкодер не замыкающий в цепочке, и к нему подключен следующий, чтобы он передавал сигналы по шине I2C дальше:

```
encoderLeft.unTerminate();
```

3. Инициализируем второй (правый) и заключительный (по умолчанию) в цепочке энкодер с автоматическим назначением ему нового уникального адреса:

```
encoderRight.init(MOTOR_393_TORQUE_ROTATIONS, MOTOR_393_TIME_DELTA);
```

Все. Инкрементные энкодеры после этого будут настроены для работы. Можно еще использовать команду `encoderLeft.setReversed(true);` для указания энкодеру, что при движении робота вперед, двигатель установлен на роботе так, что фактически вращается назад, поэтому нужно программно скорректировать показания энкодера о вращении этого колеса.

Наберите в Arduino IDE и загрузите в контроллер следующий скетч:

```
/*
Пример получения данных от энкодеров на двигателях 393,
1й (левый) энкодер подключен в разъем GND-5V-SCL-SDA
контроллера Технолаб,
2й (правый) энкодер подключен проводом к первому энкодеру

При вращении левых и правых колес показания энкодеров
видны в окне Монитора порта.

При одном обороте колеса позиция
энкодера меняется на 2.0 единцы.

Направление вращения отображается положительным (вперед)
или отрицательным (назад) приращением позиции энкодера.
*/
//подключение библиотеки для работы с i2c шиной
#include <Wire.h>
//подключение библиотеки для работы с I2C энкодером
#include <I2CEncoder.h>
//объявляем энкодер путем присовения ему имени
I2CEncoder encoderLeft, encoderRight;
```

```
  void setup() {
    // инициализируем подключение по i2c шине
    Wire.begin();
    //инициализируем соединение с ПК по последовательному порту
    //со скоростью 9600 бод
    Serial.begin(9600);
    //инициализируем первый (левый) энкодер,
    //установленный на 393 моторе
      encoderLeft.init(MOTOR_393_TORQUE_ROTATIONS,  MOTOR_393_TIME_
DELTA);
    //Левый мотор развернут и вращается назад
    //при движении робота вперед, отметим
    //это для левого энкодера
    encoderLeft.setReversed(true);
    //К первому энкодеру подключен второй
    //поэтому первый энкодер не последний
    //и должен передавать данные дальше по цепочке
    encoderLeft.unTerminate();
    //инициализируем второй (правый) энкодер,
    //установленный на 393 моторе
     encoderRight.init(MOTOR_393_TORQUE_ROTATIONS, MOTOR_393_TIME_
DELTA);
  }

  void loop() {
    //получаем значение скорости в оборотах в минуту
    //и отправляем данные в последовательный порт
    Serial.print(«Left adr:»);
    //получаем i2c-адрес энкодера
    Serial.print(encoderLeft.getAddress());
    Serial.print(«, Speed:»);
    //получаем текущую скорость
    Serial.print(encoderLeft.getSpeed());
    Serial.print(«, Pos:»);
    //получаем значение положения в оборотах
    Serial.print(encoderLeft.getPosition());

    Serial.print(«, Right adr:»);
    //получаем i2c-адрес энкодера
    Serial.print(encoderRight.getAddress());
    Serial.print(«, Speed:»);
    //получаем текущую скорость
    Serial.print(encoderRight.getSpeed());
    Serial.print(«, Pos:»);
```

```
    //получаем значение положения в оборотах
    Serial.println(encoderRight.getPosition());
    //пауза между опросами энкодера 200 мс
    delay(200);
}
```

После загрузки скетча на плату в окне монитора последовательного порта можно будет видеть присвоенные энкодерам I2C-адреса и следить за показаниями левого и правого энкодоров: скоростью вращения и положением. При этом следует учитывать, что одному обороту вала мотора соответствует 2 единицы энкодера.

В предыдущей главе мы управляли маневрами робота по таймингу. Такое управление не позволяет точно задавать перемещения и повороты робота, т.к. много влияющих факторов, вносят погрешность, например, степень разряда батареи питания, коэффициент трения поверхности дороги, трение в моторах и зубчатых передачах и т.п. Теперь пользуясь данными с энкодеров, давайте попробуем точнее управлять передвижением ClawBot на заданное расстояние и поворотами на заданный угол в градусах.

Мотор 393, вращающий левые колеса, подключите к клемме M1, а мотор 393, вращающий правые колеса, подключите к клемме M2. К этим клеммам подведены порты контроллера с поддержкой ШИМ (PWM), что позволит нам управлять плавным изменением мощностей левого и правого моторов, а следовательно и скоростью вращения колес в нужном направлении. Для управления мощностью вращения в заданном направлении вала мотора на один порт будем подавать низкий уровень (`LOW`), а на другой порт будем подавать ШИМ-сигнал в диапазоне от 1 до 254. Чтобы вал мотора вращался в обратном направлении, подаваемые на порты сигналы поменяем местами. Для более удобного управления левым и правым моторами в скетче напишем процедуры `powerMotor1()`, `powerMotor2()` и `stopAll()`. Последняя процедура будет написана для остановки всех моторов робота ClawBot.

Для расчета количества оборотов двигателя для прохождения роботом `l` нужного расстояния, нам потребуется измерить `d` диаметр колес (10.3 см). Помня, как вычисляется длина окружности колеса (`πd`), мы сможем рассчитать число оборотов колеса для прохождения колесом заданного расстояния:

$$n = l/(\pi d),$$

где `l` — расстояние, которое должен проехать робот,
`d` —диаметр колеса 10.3 см,
`π` — число пи.
Учтем, что показания энкодера будут в два раза больше числа оборотов моторов.

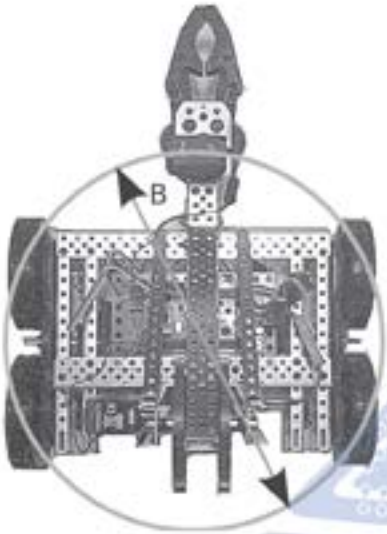
Для расчета числа оборотов моторов при повороте робота на нужный угол в градусах, воспользуемся формулой:

$$n = a*B/(360*d),$$

где `a` — угол поворота робота в градусах,

B — колея при повороте на месте (не путать с колеей при прямолинейном движении робота) 33 см,

d —диаметр колеса 10.3 см.



При повороте робота его колеса с одной стороны вращаются вперед, с другой стороны - назад, тогда робот поворачивается на месте. Причем колея при повороте робота ClawBot на месте должна измеряться между центрами отпечатков колес на дороге, т.е. колес расположенных по диагонали. В нашем случае это 33 см.

Перед маневром нужно запомнить текущее положение, выдаваемое энкодером, затем начать маневр и в цикле ожидать, пока энкодер не вернет рассчитанное для завершения маневра значение, после чего маневр считается завершенным.

Еще один важный момент — это синхронность поворотов колес при прямолинейном движении робота, а также при повороте. Получая данные с энкодеров, мы будем вычислять расхождение между показаниями левого и правого энкодеров и на основе этих данных вычислять корректирующее управляющее воздействие u на двигатели, т.е. при отставании одного колеса от другого мы будем увеличивать мощность отстающего и уменьшать мощность обгоняющего колес. Расчет управляющего воздействия и корректировку мощности моторов поместим в тело цикла при осуществлении маневра.

Для точного управления движения робота по энкодерам запрограммируем процедуры `forward()` и `backward()`, параметром которых будет вещественное число, указывающее расстояние в сантиметрах для движения робота. Также запрограммируем процедуры для точных поворотов робота на месте влево и вправо по данным энкодеров, обязательно учитывая то, что левые и правые колеса должны вращаться в разных направлениях.

Ниже приведен скетч, в котором в циклической процедуре `loop()` запрограммировано движение ClawBot'а вперед на 50см, затем поворот налево на 90 градусов. Таким образом, робот будет ездить по квадрату.

Загрузите в робота скетч:

```
/*
ClawBot движется по квадрату, используя энкодеры
1 движется прямо 50 см
2 поворот налево на 90 гр.*
3 переход на п1.

*погрешность угла поворота сильно
зависит от коэффициента трения «дороги»
*/

#define DEBUG true //показывать сообщения

//подключение библиотеки для работы с i2c шиной
#include <Wire.h>
//подключение библиотеки для работы с I2C энкодером
#include <I2CEncoder.h>
//объявляем энкодер путем присовения ему имени
I2CEncoder encoderLeft, encoderRight;

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM1PinTwo = 45;
// motor2 M2 (Right)
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;

//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;

//скорость моторов на колесах
int speedMotor = 150;

//коэффициент пропорциональности
int kp = 50;

//Число Пи
#define pi 3.1416
```

```
void setup() {
  // инициализируем подключение по i2c шине
  Wire.begin();
  //инициализируем соединение с ПК по последовательному порту
  //со скоростью 9600 бод
  Serial.begin(9600);
  //инициализируем первый (левый) энкодер,
  //установленный на 393 моторе
    encoderLeft.init(MOTOR_393_TORQUE_ROTATIONS,  MOTOR_393_TIME_
DELTA);
  //Левый мотор развернут и вращается назад
  //при движении робота вперед, отметим
  //это для левого энкодера
  encoderLeft.setReversed(true);
  //К первому энкодеру подключен второй
  //поэтому первый энкодер не последний
  //и должен передавать данные дальше по цепочке
  encoderLeft.unTerminate();
  //инициализируем второй (правый) энкодер,
  //установленный на 393 моторе
   encoderRight.init(MOTOR_393_TORQUE_ROTATIONS, MOTOR_393_TIME_
DELTA);

  //настраиваем на вывод порты M1
  pinMode(motorM1PinOne, OUTPUT);
  pinMode(motorM1PinTwo, OUTPUT);

  //настраиваем на вывод порты M2
  pinMode(motorM2PinOne, OUTPUT);
  pinMode(motorM2PinTwo, OUTPUT);

  if (DEBUG) Serial.println(«Start programm»);
  delay(100);
}
void loop() {
  //движение робота прямо на 50 см
  forward(50.0);

  stopAll(); //Стоп моторы

  //поворот робота налево на 90гр, погрешность +/-10гр
  left(90.0);

  stopAll(); //Стоп моторы
}
```

```
// движение вперед на заданное число см
void forward(float l) {
    //Запомнить текущие позиции энкодеров
  double oldPosLeft = encoderLeft.getPosition();
  double oldPosRight = encoderRight.getPosition();
  float u=0.0;
  // левый и правый моторы вперед со
  // скоростью speedMotor
  powerMotor1(speedMotor);
  powerMotor2(speedMotor);
  //пока левое колесо не проедет 50 см оставаться в цикле
  while (abs(oldPosLeft-encoderLeft.getPosition()) < 2.0 * l/(pi
* 10.3)){
      //от разницы энкодеров рассчитаем поправочное
      //управляющее воздействие на оба мотора
      u = kp*((oldPosLeft-encoderLeft.getPosition()) -
          (oldPosRight-encoderRight.getPosition()));
      //Если одно колесо отстает, то его мотору добавим
      // мощности, а другому убавим.
      powerMotor2(speedMotor - u);
      powerMotor1(speedMotor + u);
      if (DEBUG){
        Serial.print(«Forward u:»);
        Serial.println(u);
      }
    }
}
// движение назад на заданное число см
void backward(float l) {
    //Запомнить текущие позиции энкодеров
  double oldPosLeft = encoderLeft.getPosition();
  double oldPosRight = encoderRight.getPosition();
  float u=0.0;
  // левый и правый моторы вперед со
  // скоростью speedMotor
  powerMotor1(-speedMotor);
  powerMotor2(-speedMotor);
  //пока левое колесо не проедет 50 см оставаться в цикле
  while (abs(oldPosLeft-encoderLeft.getPosition()) < 2.0 * l/(pi
* 10.3)){
      //от разницы энкодеров рассчитаем поправочное
      //управляющее воздействие на оба мотора
      u = kp*((oldPosLeft-encoderLeft.getPosition()) -
```

```
            (oldPosRight-encoderRight.getPosition()));
        //Если одно колесо отстает, то его мотору добавим
        // мощности, а другому убавим.
        powerMotor2(-(speedMotor + u));
        powerMotor1(-(speedMotor - u));
        if (DEBUG){
          Serial.print(«Backward u:»);
          Serial.println(u);
        }
      }
  }
  //поворот робота налево на заданный угол в градусах
  void left(float a){
    //Запомнить текущие позиции энкодеров
    double oldPosLeft = encoderLeft.getPosition();
    double oldPosRight = encoderRight.getPosition();
    float u=0.0;
    // левый мотор назад, правый мотор вперед со
    // скоростью speedMotor
    powerMotor1(-speedMotor);//»-» вращать назад
    powerMotor2(speedMotor);
    //пока колесо не проедет дугу = a*B/(360*d) для поворота робота
    //на заданный угол a, где B - колея при повороте, d - диаметр
  колеса
    float B = 33.0;
    float d = 10.3;
    //учтем в условии, что 1 оборот колеса = 2 ед. энкодера
     while (abs(oldPosLeft-encoderLeft.getPosition()) < 2.0*a*B/
(360.0 * d)){
        //от разницы энкодеров рассчитаем поправочное
        //управляющее воздействие на оба мотора
        //учтем, что левый мотор - назад, поэтому показания энкоде-
ров плюсуем
        u = kp*((oldPosLeft-encoderLeft.getPosition()) +
            (oldPosRight-encoderRight.getPosition()));
        powerMotor1(-speedMotor+u);//»-» вращать назад
        powerMotor2(speedMotor+u);
        if (DEBUG) {
          Serial.print(«Left u:»);
          Serial.println(u);
        }
      }
  }
```

```
  //поворот робота налево на заданный угол в градусах
  void right(float a){
    //Запомнить текущие позиции энкодеров
    double oldPosLeft = encoderLeft.getPosition();
    double oldPosRight = encoderRight.getPosition();
    float u=0.0;

    // левый мотор назад, правый мотор вперед со
    // скоростью speedMotor
    powerMotor1(speedMotor);
    powerMotor2(-speedMotor); //»-» вращать назад
    //пока колесо не проедет дугу = a*B/(360*d) для поворота робота
    //на заданный угол a, где B - колея при повороте, d - диаметр
колеса
    float B = 33.0;
    float d = 10.3;
    //учтем в условии, что 1 оборот колеса = 2 ед. энкодера
     while (abs(oldPosLeft-encoderLeft.getPosition()) < 2.0*a*B/
(360.0 * d)){
        //от разницы энкодеров рассчитаем поправочное
        //управляющее воздействие на оба мотора

        //учтем, что левый мотор - назад, поэтому показания энкоде-
ров плюсуем
        u = kp*((oldPosLeft-encoderLeft.getPosition()) +
            (oldPosRight-encoderRight.getPosition()));
        powerMotor1(speedMotor+u);
        powerMotor2(-(speedMotor-u)); //»-» вращать назад
        if (DEBUG) {
          Serial.print(«Right u:»);
          Serial.println(u);
        }
    }
  }

  //остановить все моторы
  void stopAll(){
    //стоп манипулятор (Arm)
    analogWrite (armServoPin, 188);
    //стоп клешня (Claw)
    analogWrite (clawServoPin, 188);

    //останавливаем моторы на колесах
    powerMotor1(0); //левый мотор
    powerMotor2(0); //правый мотор
```

```
  if (DEBUG) Serial.println(«Stop all motors»);

  //задаем паузу в 1 секунду
  delay (1000);
}
//управление мотором1
void powerMotor1(int speedValue){
  //направление вращения
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor1:»);
      Serial.println(speedValue);
  }
}
//управление мотором2
void powerMotor2(int speedValue){
  //направление вращения
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
```

```
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

Попробуйте запрограммировать робота ClawBot для движения с поворотами по более сложной траектории, изменяя в скетче команды и их параметры в процедуре `loop()`, например, движение не по квадрату, а по правильному пятиугольнику, шестиугольнику.

После этого попробуйте выполнить более сложное задание: траектория робота должна напоминать букву «Н» или цифру «8».

3.3 Управление манипулятором робота

«Хватательное» движение манипулятора может быть обеспечено либо сервоприводом, либо мотором. Это необходимо учитывать при программировании этого движения. В случае использования мотора для хватательного движения «клешни» используется мотор 269, подключённый через внешний драйвер мотора. Движение манипулятора («руки») вперёд обеспечивается мотором 396, установленным на ось штанги. Подключение и управление им осуществляется через внешний драйвер мотора. Подключим мотор, отвечающий за движение штанги, к порту M13(5) через внешний драйвер мотора, а мотор, отвечающий за работу «клешни», также через драйвер мотора подключим к порту M12(2)

Напишем в среде Arduino IDE следующий скетч:

```
#include <Servo.h>

Servo grab;
//объявили мотор, отвечающий за хватательное движение Servo roll;
//объявили мотор, отвечающий за выбрасывательное движение

void setup()
{
grab.attach(2); //указываем порт, к которому подключён мотор клешни
roll.attach(5); //указываем порт, к которому подключён мотор штанги

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
delay (2000); // ожидание 2 сек.

reroll_up(); // поднимаем руку
delay (2000); // ожидание 2 сек.

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
reroll_down(); //опускаем руку
}

void reveal (){ grab.writeMicroseconds(1800); delay (900);
grab.writeMicroseconds(1500);
}
```

```
void toclose (){ grab.writeMicroseconds(1200);
delay (900); grab.writeMicroseconds(1500);
}

void reroll_up (){ roll.writeMicroseconds(1800);
delay (1000); roll.writeMicroseconds(1500);
}

void reroll_down (){ roll.writeMicroseconds(1200);
delay (1000); roll.writeMicroseconds(1500);
}

void loop (){
//здесь ничего не пишем, т.к. зацикливать нечего - программа
отрабатывает один раз
}
```

После загрузки скетча в платформу робот выполнит следующие действия:
- откроет «клешню» и, чуть подождав, закроет ее
- перекинет через себя манипулятор
- снова откроет и закроет «клешню»
- вернёт манипулятор в исходное состояние

Однако, при использовании библиотеки Servo из-за ее особенностей, нельзя будет управлять мотором, подключенным к клемме M1 (порты 46, 45) с помощью ШИМ сигнала, следовательно не будет возможности плавно регулировать мощность мотора M1.

Возможен вариант управления через ШИМ-сигналы моторами «клешни» и «руки» (подключенными через адаптеры Motor Controller 29) без библиотеки Servo с помощью обычной команды:

analogWrite(порт, значение);

Тогда без использования библиотеки Servo будет возможно и плавное управление мощностью вращения колес робота ClawBot с помощью моторов, подключенных к клеммам M1 и M2.

Для этого важно правильно задать нужное значение:

- от 122 до 184 – вращение мотора по часовой стрелке, причем 122 – максимальная мощность, а 184 – минимальная мощность вращения,
- 188 – остановка мотора,
- от 192 до 254 – вращение мотора против часовой стрелки, причем 192 – минимальная мощность вращения, 254 – максимальная мощность вращения.

Зная необходимые значения, перепишем вышеприведенный скетч по-другому:

```
/*
Пример управления моторами «клешни» и «руки» через
драйвер Motor Controller 29, без использования
библиотеки Servo, которая блокирует порты M1(46,45)
*/

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

void setup()
{
//указываем порт, к которому подключён мотор клешни
pinMode(armServoPin, OUTPUT);
//указываем порт, к которому подключён мотор штанги
pinMode(clawServoPin, OUTPUT);

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
delay (2000); // ожидание 2 сек.

reroll_up(); // поднимаем руку
delay (2000); // ожидание 2 сек.

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
reroll_down(); //опускаем руку
}
void reveal (){
analogWrite(clawServoPin, 221);
delay (900);
analogWrite(clawServoPin, 188);
}
void toclose (){
analogWrite(clawServoPin, 155);
delay (900);
analogWrite(clawServoPin, 188);
}
```

```
void reroll_up (){
analogWrite(armServoPin, 155);
delay (1000);
analogWrite(armServoPin, 188);
}

void reroll_down (){
analogWrite(armServoPin, 221);
delay (1000);
analogWrite(armServoPin, 188);
}

void loop (){
//здесь ничего не пишем, т.к. зацикливать нечего - программа
отрабатывает один раз
}
```

В случае использования сервопривода вместо мотора для управления клешней схвата подключение сервопривода может быть произведено к порту M12(2), и код скетча будет выглядеть несколько иначе:

```
#include <Servo.h>

Servo grab;
//объявили мотор, отвечающий за хватательное движение
Servo roll;
//объявили мотор, отвечающий за выбрасывательное движение

void setup()
{
grab.attach(2); //указываем порт, к которому подключён мотор клешни
roll.attach(5); //указываем порт, к которому подключён мотор штанги

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
delay (2000); // ожидание 2 сек.

reroll_up(); // поднимаем руку
delay (2000); // ожидание 2 сек.

reveal(); // вызов функции раскрытия хвата
delay (2000); // ожидание 2 сек.

toclose(); // вызов функции сомкнутия хвата
reroll_down(); //опускаем руку
}
```

```
void reveal (){
grab.write(180); // раскрытие на 180 градусов
}
void toclose (){
grab.write(50); //сомкнутие до 50 градусов
}
void reroll_up (){
roll.write(1000); // движение против часовой стрелки
delay (1700);
roll.write(1500); // остановка
}
void reroll_down (){
roll.write(1800); // движение по часовой стрелке
delay (1600);
roll.write(1500); // остановка
}
void loop (){
//здесь ничего не пишем, т.к. зацикливать нам нечего - программа
отрабатывает один раз
}
```

Отличием данного скетча от предыдущего является то, что управление сервоприводом осуществляется путём задания его положения в градусах, тогда как в предыдущем скетче управление осуществлялось путём PWM импульсами с использованием искусственных временных задержек.

3.4 Подключение ультразвукового дальномера

В рамках данной задачи ультразвуковой дальномер будет использоваться для обнаружения какого-либо объекта на пути робота и определения расстояния до него с целью последующего его захвата манипулятором. В данном случае дальномер подключается своим выводом INPUT к порту 0(48), а выводом OUTPUT к порту 1(49).

Скетч для работы с дальномером и манипулятором выглядит следующим образом:

```
#include <Servo.h>

Servo grab; //объявили сервомодуль,
отвечающий за «хватательное» движение
Servo roll; //объявили сервомодуль,
отвечающий за «выбрасывательное» движение

const int input = 48; // объявили порт для управляющего сигнала
const int output = 49; // объявили порт для считанного сигнала

void setup()
{
pinMode(input, OUTPUT);
//сконфигурировали порт для управляющего сигнала на выход
pinMode(output, INPUT);
//сконфигурировали порт для считанного сигнала на вход

grab.attach(2);
//указываем порт, к которому подключён мотор «клешни»
roll.attach(5);
//указываем порт, к которому подключён мотор штанги
}
unsigned int time_us=0;
//объявили переменную для времени
unsigned int distance_sm=0;
//объявили переменную для расстояния
unsigned int distance=0; //объявили переменную для расстояния

void loop() {
distance = ultrasonic();
if (distance <= 20) {
//если дистанция до объекта менее 20 см, то захватываем его
 reveal();
```

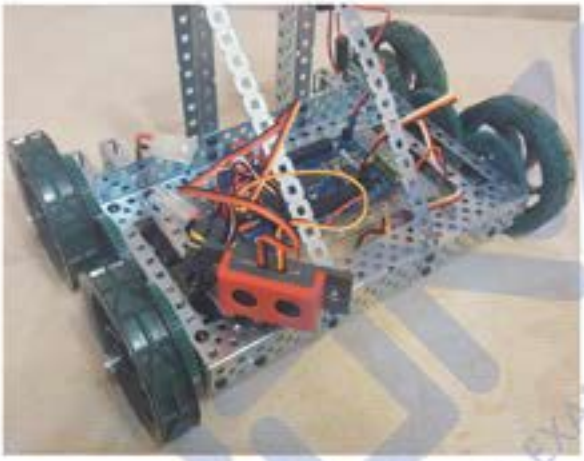
```
// вызов функции раскрытия хвата
 delay(1100);
toclose();
// вызов функции закрытия хвата
 delay(500);

reroll_up();
// поднимаем руку
 delay (2000);
// ожидание 2 сек.

 reveal();
// вызов функции раскрытия хвата
 delay (2000);
// ожидание 2 сек.

 toclose();
// вызов функции закрытия хвата
 reroll_down();
//опускаем руку
}
delay(100);
}

unsigned int ultrasonic (){
digitalWrite(input, HIGH);
// подаём управляющий сигнал
delayMicroseconds(10);
// удерживаем 10 микросекунд
digitalWrite(input, LOW);
// убираем управляющий сигнал

time_us=pulseIn(output, HIGH);
// замеряем длину импульса
distance_sm=time_us/58;
// пересчитываем в сантиметры
Serial.println(distance_sm);
// передаём значение в порт
return distance_sm;
}

void reveal (){
grab.writeMicroseconds(1800);
delay (700);
 grab.writeMicroseconds(1500);
}
```

```
void toclose (){
 grab.writeMicroseconds(1200);
delay (700);
grab.writeMicroseconds(1500);
}

void reroll_up (){
roll.writeMicroseconds(1800);
delay (2000);
roll.writeMicroseconds(1500);
}

void reroll_down (){
roll.writeMicroseconds(1200);
delay (2000);
roll.writeMicroseconds(1500);
}
```

Данный скетч реализует непрерывный опрос датчика расстояния, и в случае обнаружения какого-либо объекта на расстоянии менее 20 см производит открытие хвата, захват объекта и его последующий выброс позади себя. После чего робот снова начинает опрашивать дальномер.

3.5 Движение по лабиринту

Для движения ClawBot по лабиринту снимем временно с робота манипулятор и установим ультразвуковой дальномер, причем при установке направим его не прямо вперед, а закрепим под углом 45 градусов вправо. Это позволит нам запрограммировать робота на прохождение лабиринта по правилу правой руки. Представьте себе, что у нас завязаны глаза, и мы движемся в незнакомом здании по лабиринту коридоров и комнат, касаясь за стену правой рукой. Даже если мы свернем в комнату, мы все равно выйдем из нее, обойдя ее по периметру. Рано или поздно мы выйдем из лабиринта благодаря этому правилу.



Наш робот будет выдерживать расстояние в 50 см до стены ультразвуковым дальномером, повернутым на 45 градусов вправо. Причем нам потребуется запрограммировать робота так, чтобы при отдалении от стены более 50 см, робот подруливал к ней моторами, вращая левый мотор быстрее правого, а при приближении к стене менее 50 см — отруливал от нее, вращая правый быстрее левого.

Для управления моторами мы применим пропорциональный регулятор: будем считать управляющее воздействие на моторы (переменная u) от разницы расстояния до стены, полученного от дальномера, и нормального расстояния в 50 см.

```
distance – 50
```

Когда робот, двигаясь вдоль стенки, подъедет к повороту направо, то дальномер может показать такое большое значение, что робот совершит резкий поворот и врежется в стенку справа. Чтобы робот плавно входил в поворот, мы ограничим число, полученное от дальномера с помощью функции constrain(переменная, минимум, максимум). Она возвращает значение переменной, но не меньше минимума и не больше максимума:

```
constrain(distance,0,80) -  50
```

Чтобы подруливание моторами было эффективным (не маленьким и не большим), полученную разницу умножим на коэффициент пропорциональности, который подберем экспериментально. Нужно будет несколько раз менять в программе коэффициент и запускать робота в лабиринте, пока не добьемся удовлетворительного движения. Например, в нашем случае коэффициент пропорциональности kp равен 4.5, тогда управляющее воздействие на моторы (переменная u) высчитывается пропорционально дистанции такой командой:

```
u=kp*(constrain(distance,0,80) -  50);
```

Значение переменной u будем вычитать из мощности одного мотора и прибавлять к мощности другого мотора, тогда робот будет маневрировать пропорционально показаниям дальномера. В этом и есть суть пропорционального регулятора.

Наберите в Arduino IDE следующий скетч и загрузите его в контроллер:

```
/*
Движение ClawBot с контроллером Технолаб
по лабиринту по правилу «правой руки»
*/

#define DEBUG true //показывать сообщения

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM1PinTwo = 45;
// motor2 M2 (Right)
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;

//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;

//скорость моторов на колесах
int speedMotor = 150;
```

```
  //коэффициент пропорциональности
  float kp = 4.5;

  //указываем порт, к которому подключён вход INPUT датчика рассто-
яния 20(23)
  const int Trig = 23;
  //указываем порт, к которому подключён выход OUTPUT датчика рас-
стояния 21(22)
  const int Echo = 22;
  //для хранения данных о расстоянии
  int distance=0;
  //объявляем и инициализируем переменную для работы с временем
  unsigned int time_us=0;
  //объявляем и инициализируем переменную для работы с расстоянием
  unsigned int distance_sm=0;

  void setup() {
    //настраиваем порт входа датчика расстояния на вывод сигнала
    pinMode(Trig, OUTPUT);
    //настраиваем порт выхода датчика расстояния на ввод сигнала
    pinMode(Echo, INPUT);
    //инициализируем соединение с ПК по последовательному порту
    //со скоростью 9600 бод
    Serial.begin(9600);
    //настраиваем на вывод порты M1
    pinMode(motorM1PinOne, OUTPUT);
    pinMode(motorM1PinTwo, OUTPUT);

    //настраиваем на вывод порты M2
    pinMode(motorM2PinOne, OUTPUT);
    pinMode(motorM2PinTwo, OUTPUT);

    if (DEBUG) Serial.println(«Start programm»);
    delay(100);
  }
  void loop() {
  // подаём сигнал на вход датчика
  digitalWrite(Trig, HIGH);
  // удерживаем его 10 микросекунд
  delayMicroseconds(10);
  // убираем сигнал со входа датчика
  digitalWrite(Trig, LOW);
  // замеряем длину импульса на выходе датчика
  //таймаут 50 миллисекунд
```

```
time_us=pulseIn(Echo, HIGH, 50000);
// пересчитываем полученные данные в сантиметры
distance_sm=time_us/58;
// выводим в последовательный порт полученные значения расстояния
if (DEBUG) Serial.println(distance_sm);
//фильтр по среднему значению по минимизации помех
distance=(distance+distance_sm)/2;

//управление моторами
float u=0.0;
//нормальное расстояние до стенки лабиринта 50 см
//если расстояние менее 30 см, то
//впереди - стенка! Сделать поворот налево
if (distance < 30) {
    powerMotor1(-speedMotor);//»-» вращать назад
    powerMotor2(speedMotor);
    //тайминг для завершения маневра
    delay(400);
    u=-2*speedMotor;
  }
  else u=kp*(constrain(distance,0,80) -  50);
powerMotor1(speedMotor + u); //левый мотор
powerMotor2(speedMotor - u); //правый мотор

//пауза между опросами датчика расстояния - 100 мс
delay(100);
}

//управление мотором1
void powerMotor1(int speedValue){
  //направление вращения
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
```

```
    }
  if (DEBUG) {
      Serial.print(«motor1:»);
      Serial.println(speedValue);
  }
}

//управление мотором2
void powerMotor2(int speedValue){
  //направление вращения
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

Постройте лабиринт, например, из картонных коробок и запустите в него робота. Если робот не будет вписываться в правый поворот, попробуйте увеличить коэффициент пропорциональности, чтобы робот сильнее поворачивал. Если робот, заехав в тупик, недостаточно поворачивает налево, увеличьте тайминг для этого маневра.

3.6 Работа с ИК-датчиками для обнаружения линии

Одним из способов удержания робота на определённом маршруте является езда по линии. Линия (чёрного цвета), изображённая на контрастном фоне (белого цвета), легко фиксируется роботом с помощью детекторов линии, что позволяет роботу двигаться строго вдоль заранее заданной линии, не уходя с неё. В нашем случае используются 3 детектора линии, установленные параллельно друг другу, таким образом, чтобы расстояние от нижнего края детектора до поверхности, по которой едет робот, было минимальным. В случае когда центральный детектор фиксирует наличие линии, робот едет прямо, а если наличие линии фиксирует один из крайних детекторов, то робот поворачивает в нужную сторону до тех пор, пока центральный детектор не зафиксирует линию.

Подключение детекторов линии выполняется следующим образом: левый детектор линии подключается к порту 2(37), центральный – к порту 3(36) и правый – к 4(35).

Скетч для простого опроса всех трёх детекторов одновременно выглядит следующим образом:

```
int sensorPort1 = 37;
//указываем порт, к которому подключён Line Tracker 1
int sensorPort2 = 36;
//указываем порт, к которому подключён Line Tracker 2
int sensorPort3 = 35;
//указываем порт, к которому подключён Line Tracker 3

int sensorValue1=0;
//объявляем переменную для Line Tracker 1
int sensorValue2=0;
//объявляем переменную для Line Tracker 2
int sensorValue3=0;
//объявляем переменную для Line Tracker 3

void setup() {
pinMode(sensorPort1, INPUT);
pinMode(sensorPort2, INPUT);
pinMode(sensorPort3, INPUT);
//инициализация подключения по com-порту
Serial.begin(9600);
}
```

```
void loop() {
//считываем значения с портов
sensorValue1 = digitalRead(sensorPort1);
sensorValue2 = digitalRead(sensorPort2);
sensorValue3 = digitalRead(sensorPort3);
//выводим на последовательный порт
с конвертацией значений в вольты
Serial.println(sensorValue1);
Serial.println(sensorValue2);
Serial.println(sensorValue3);
delay(100);
 }
```

Выполнив объединение скетчей для езды и скетча для опроса детекторов, получаем скетч для езды робота по линии:

```
//указываем пины, отвечающие за управление портом M1
const int motorM1PinOne = 6;
const int motorM1PinTwo = 7;

//указываем пины, отвечающие за управление портом M2
const int motorM2PinOne = 46;
const int motorM2PinTwo = 45;

int sensorPort1 = 37;
//указываем порт, к которому подключён Line Tracker 1
int sensorPort2 = 36;
//указываем порт, к которому подключён Line Tracker 2
int sensorPort3 = 35;
//указываем порт, к которому подключён Line Tracker 3

int sensorValue1=0;
//объявляем переменную для Line Tracker 1
int sensorValue2=0;
//объявляем переменную для Line Tracker 2
int sensorValue3=0;
//объявляем переменную для Line Tracker 3

void setup() {
pinMode(sensorPort1, INPUT);
pinMode(sensorPort2, INPUT);
pinMode(sensorPort3, INPUT);
```

```
//конфигурируем пины порта M1 на вывод сигнала
pinMode(motorM1PinOne, OUTPUT);
pinMode(motorM1PinTwo, OUTPUT);
//конфигурируем пины порта M2 на вывод сигнала
pinMode(motorM2PinOne, OUTPUT);
pinMode(motorM2PinTwo, OUTPUT);
}

void loop() {
forvard();
 //считываем значения с портов
sensorValue1 = digitalRead(sensorPort1);
sensorValue2 = digitalRead(sensorPort2);
sensorValue3 = digitalRead(sensorPort3);

if (sensorValue3){
//контролируем значения с правого line tracker
right();
//если меньше определённого - подворачиваем
}

if (sensorValue1){
// контролируем значения с левого line tracker
left();
//если меньше определённого - подворачиваем
}

if (sensorValue1 && sensorValue2 && sensorValue3) {
//если все 3 детектора фиксируют линию
stopMove();
//остановка
}
delay(10);
}

void forvard (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}
```

```
void right (){
//правый мотор
digitalWrite (motorM2PinOne, HIGH);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}

void left (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, HIGH);
digitalWrite (motorM1PinTwo, LOW);
}

void stopMove (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, LOW);
}
```


3.7 Работа с ИК-датчиками для движения по линии. Релейный регулятор

Следование робота по черной линии одна из классических задач для юных робототехников. Существует много алгоритмов следования по линии и самый простой из них — релейный регулятор. Достаточно одного датчика линии, чтобы использовать этот регулятор. Для следования ClawBot по линии установим в передней части по центру датчик линии Line Tracker. Для нормальной работы датчика расстояние от него до пола не должно превышать 1 см. Датчик линии подключите к порту A8 контроллера. Для удобства временно можно снять с робота манипулятор. Моторы, вращающие колеса ClawBot, подключите к клеммам контроллера: левый мотор к M1, правый мотор к M2.

Суть релейного алгоритма заключается в том, чтобы отслеживать датчиком наличие черной линии. Если линия под датчиком есть, то включаем левый мотор вперед, а правый останавливаем. Когда под датчиком окажется белое поле, то останавливаем левый мотор и включаем вперед правый. В результате робот будет ***«вилять»*** влево-вправо по границе черной линии и белого поля, но продвигаться вперед по черной линии.

Наберите такой скетч в Arduino IDE и загрузите его в контроллер:

```
/*
Следование ClawBot с контроллером Технолаб
по линии по релейному алгоритму.
Датчик линии включен в режим «цифровой»
*/
#define DEBUG true //показывать сообщения

//указываем порт, к которому подключён детектор линии - A8
const int lineTrackerPort = A8;

//объявляем переменную для работы с детектором линии
int lineTrackerState = 0;

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM1PinTwo = 45;
// motor1 M2 (Right)
```

```
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;

//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;

//скорость движения моторов на колесах
int speedMotor = 150;

void setup() {
  //конфигурируем выбранный порт на чтение
  pinMode(lineTrackerPort, INPUT);
  //инициализируем соединение с ПК по последовательному порту
  //со скоростью 9600 бод
  Serial.begin(9600);

  //настраиваем на вывод порты мотора M1
  pinMode(motorM1PinOne, OUTPUT);
  pinMode(motorM1PinTwo, OUTPUT);

  //настраиваем на вывод порты мотора M2
  pinMode(motorM2PinOne, OUTPUT);
  pinMode(motorM2PinTwo, OUTPUT);

  if (DEBUG) Serial.println(«Start programm»);
  delay(100);
}

void loop() {
//читаем уровень сигнала на выбранном цифровом порту
lineTrackerState=digitalRead(lineTrackerPort);
//выводим в последовательный порт сообщение в зависимости
от состояния детектора
//если попал на чёрную линию, уровень становится высоким
if (lineTrackerState == HIGH){
  if (DEBUG) Serial.println(«Black line»);
  //поворот ClawBot направо
  //левое колесо вперед
  powerMotor1(speedMotor);
  //правое колесо - стоп
  powerMotor2(0);
```

```
  }
else {
//если попал на белый фон, на выходе держится низкий уровень
  if (DEBUG) Serial.println(«White backgrownd»);
  //поворот ClawBot налево
  //левое колесо - стоп
  powerMotor1(0);
  //правое колесо - вперед
  powerMotor2(speedMotor);
}
//задаем паузу в 100 мс
delay(100);
}

//управление мотором1
void powerMotor1(int speedValue){
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (DEBUG) {
     Serial.print(«motor1:»);
     Serial.println(speedValue);
  }
}

//управление мотором2
void powerMotor2(int speedValue){
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);
```

```
  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

Затем запускайте робота по учебному полю с черной линией. Оптимальная ширина черной линии 5 см, однако, в качестве черной линии может подойти и черная изолента шириной 20..25мм, наклеенная на лист ватмана. От скорости робота и ширины черной линии зависит точность следования робота по линии. Если робот будет терять линию, то уменьшайте скорость робота значением переменной speedMotor.

Релейный алгоритм самый простой и надежный регулятор, его несложно настроить, подобрав оптимальную скорость робота. Однако, следуя по линии робот, управляемый этим регулятором, постоянно виляет из стороны в сторону, его моторы, то разгоняясь, то затормаживая, работают не самым эффективным образом, поэтому для получения максимальных скоростей на трассе данный регулятор мало подходит. Также, если на трассе имеются перекрестки или пунктиры, то робот часто теряет на них направление и сходит с трассы.

3.8 Работа с ИК-датчиками для движения по линии. Пропорциональный регулятор

Пропорциональный регулятор позволяет запрограммировать робота так, что он будет более плавно следовать по линии, чем под управлением релейного регулятора. Важно, чтобы датчик линии работал в аналоговом режиме, тогда при смещении его от белого поля до черной линии показания датчика будут меняться в широком диапазоне. Это позволяет плавно управлять мощностью моторов и, следовательно, скоростью вращения колес робота, чтобы ClawBot мог плавно подруливать в нужную сторону.

Для следования робота ClawBot по линии установим в передней части робота по центру датчик линии ***Line Tracker.*** Для нормальной работы датчика расстояние от него до пола не должно превышать 1 см. Датчик линии нужно подключить к аналоговому порту A8 контроллера. Для удобства временно можно снять с робота манипулятор. Моторы, вращающие колеса ClawBot, подключите к клеммам контроллера: левый мотор к `M1`, правый мотор к `M2`.

Нам нужно запрограммировать робота так, чтобы при идеальном состоянии, когда датчик линии находится точно над границей черной линии и белого поля, робот двигался прямо. Для этого перед началом движения по линии установим робота идеально над линией, чтобы датчик линии находился в среднем положении над линией. Если в этот момент посмотреть через камеру вашего смартфона (она может улавливать инфракрасное излучение!) на поле под датчиком, вы сможете рассмотреть на поле световое пятно, которое излучает датчик линии. Отраженное от этого светового пятна излучение датчик линии измеряет и передает данные на контроллер. Важно, чтобы это световое пятно от датчика наполовину попадало на черную линию и наполовину на белое поле.

Сразу после точной установки датчика над линией и включении робота запомним значение показания датчика линии в переменной `averageValue`. Эта операция называется калибровка. Калибровку важно проводить перед каждым стартом, потому, что на показания датчика сильно влияет изменение освещения в помещении, в котором поедет робот, свойства поля с линией, по которому будет двигаться робот: контрастность линии, ее цвет, светопоглощение материала поля, глянец покрытия и т.п.

Затем, при движении робота по линии мы будем вычислять величину смещения датчика от среднего положения над линией: `(averageValue – lineTrackerValue)`,- в зависимости от положительного или отрицательного значения этой величины, роботу нужно будет подруливать либо влево, либо вправо. Например, если нужно подруливать влево, то скорость левого мотора нужно замедлить на некоторую величину и на эту же величину повысить скорость правого мотора. Если отклонение датчика линии от среднего положения небольшое, то подрулить нужно чуть-чуть, но если отклонение большое, то величина изменения скорости моторов должна быть больше.

Следовательно, величина изменения скорости моторов должна быть пропорциональна отклонению датчика линии от среднего положения над линией.

Для подруливания рассчитаем управляющее воздействие на левый и правый двигатели робота, значение которого присвоим переменной u:

```
int u = kp*(averageValue - lineTrackerValue);
```

Коэффициент пропорциональности 0.2 подбирается для каждого робота экспериментально. Чем он больше, тем резче будут повороты робота при отклонении от линии, но при очень резком подруливании робот может сойти с трассы. Чем он меньше, тем плавнее будут повороты, но при очень медленном подруливании робот может не вписаться в поворот и тоже сойти с трассы.

Скетч, в котором используется пропорциональный регулятор для следования ClawBot'а по линии, приведен ниже:

```
/*
Следование ClawBot с контроллером Технолаб
по линии по пропорциональному алгоритму.
Датчик линии включен в режим «аналоговый»

Перед включением робота датчик линии нужно точно
выставить над границей черной линии и белого поля
*/

#define DEBUG true //показывать сообщения

//указываем порт, к которому подключён детектор - A8
const int lineTrackerPort = A8;
//объявляем переменную для работы с детектором
int lineTrackerValue = 0;
//переменные для сохранения значений детектора
int averageValue=512;

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM1PinTwo = 45;
// motor1 M2 (Right)
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;
```

```
//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;

//скорость моторов на колесах
int speedMotor = 80;

//коэффициент пропорциональности
float kp = 0.2;

void setup() {
  //конфигурируем выбранный порт на чтение
  pinMode(lineTrackerPort, INPUT);
  //инициализируем соединение с ПК по последовательному порту
  //со скоростью 9600 бод
  Serial.begin(9600);
  //настраиваем на вывод порты M1
  pinMode(motorM1PinOne, OUTPUT);
  pinMode(motorM1PinTwo, OUTPUT);

  //настраиваем на вывод порты M2
  pinMode(motorM2PinOne, OUTPUT);
  pinMode(motorM2PinTwo, OUTPUT);

  if (DEBUG) Serial.println(«Start programm»);
  delay(100);
  //при старте датчик робота точно над границей черной линии
  //читаем уровень сигнала на выбранном аналоговом порту
  averageValue = analogRead(lineTrackerPort);
  if (DEBUG) Serial.println(averageValue);
}

void loop() {
  //читаем уровень сигнала на выбранном аналоговом порту
  //используем фильтр по среднему арифметическому
  lineTrackerValue=(lineTrackerValue +
  analogRead(lineTrackerPort))/2;
  //выводим в последовательный порт сообщение с данными детектора
  if (DEBUG) {
    Serial.print(«LineTracker: «);
    Serial.println(lineTrackerValue);
  }
```

```
  //рассчитаем управляющее воздействие на моторы в зависимости
  //от смещения датчика от среднего положения над линией
  int u = kp*(averageValue - lineTrackerValue);

  //выводим в последовательный порт сообщение со значением
  //управляющего воздействия
  if (DEBUG) {
    Serial.print(«u:»);
    Serial.println(u);
  }

  //управление моторами
  powerMotor1(speedMotor + u); //левый
  powerMotor2(speedMotor - u); //правый

  //задаем паузу в 50 мс
  delay(50);
}

//управление мотором1
void powerMotor1(int speedValue){
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor1:»);
      Serial.println(speedValue);
  }
}

//управление мотором2
void powerMotor2(int speedValue){
```

```
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

Настройка пропорционального регулятора потребует дольше времени, чем релейного регулятора, но преимущества этого регулятора в более плавном движении робота вдоль лини, в увеличении скорости прохождения трассы, в уменьшении колебаний робота из стороны в сторону.

К недостаткам пропорционального регулятора можно отнести то, что колебания робота хоть и уменьшатся по сравнению с релейным регулятором, но избавиться от них совсем не удастся, а также то, что при прохождении перекрестков или пунктиров робот будет сходить с трассы.

3.9 Работа с ИК-датчиками для движения по линии. Двухдатчиковая схема

Мы уже экспериментально убедились, что при использовании ***пропорционального регулятора*** робот с одним датчиком Line Tracker при следовании по линии движется более плавно, чем при использовании ***релейного регулятора***. Однако, на перекрестках или пунктирах робот движется не прямо, а норовит свернуть с линии в сторону, и поэтому может сойти с трассы.

Для более уверенного прохождения роботом перекрестков и пунктиров на трассе мы добавим второй датчик линии Line Tracker. Второй датчик Line Tracker подключите к порту A9 и закрепите в передней части робота, так, чтобы оба датчика располагались над противоположными краями черной линии. Расстояние от датчика до поля не должно превышать 1 см. Моторы колес подключите к клеммам M1(левый) м M2(правый).

Пропорциональный регулятор поможет нам рассчитать управляющее воздействие на моторы в зависимости от разности показаний левого и правого датчиков линии. Действительно, когда оба датчика расположены слева и справа от линии или одновременно находятся над перекрестком, то их показания будут одинаковы, разница между ними будет близка к нулю, и робот будет двигаться вперед. Но если один из датчиков окажется над черной линией, а второй над белым полем, то разница между их показаниями е увеличится, а значит, управляющее воздействие тоже ***пропорционально*** увеличится и один мотор начнет вращаться быстрее, а второй медленнее, чтобы робот повернул для возврата на линию.

Для настройки поворотов нужно менять значение ***коэффициента пропорциональности kp.*** Чем он больше, тем резче повороты, чем меньше — тем плавнее движется робот по линии на поворотах.

Обратите внимание, что при включении питания робот должен точно стоять над линией. В процедуре start() в этот момент считываются показания левого и правого датчиков линии и рассчитывается разница показаний датчиков на старте:

```
es = leftLineTrackerValue - rightLineTrackerValue;
```

Делаем это потому, что датчики из-за разной установки или собственных характеристик могут выдавать несколько различные показания. Эту разницу мы потом для точности будем учитывать при расчете ***e - ошибки отклонений датчиков линии во время движения робота.***

Еще один интересный момент применен в данной программе — ***подтормаживание робота*** будет тем больше, чем больше ошибка отклонений датчиков линии. Для этого в переменную vt присваиваем модуль ошибки отклонений датчиков линии деленный на 150 (число в знаменателе подбирается экспериментально; чем оно больше, тем меньше эффект торможения при отклонении от линии):

```
vt = abs(e)/150;
```

Значение переменной vt будет одновременно вычитаться при расчете мощности для левого и правого моторов, а значит скорость робота при отклонении от линии будет пропорционально снижаться. Это позволит уменьшить вероятность схода робота с трассы.

Скетч, в котором используется пропорциональный регулятор для следования по линии ClawBot'а с двумя датчиками линии, приведен ниже:

```
/*
Следование ClawBot с контроллером Технолаб
по линии по пропорциональному алгоритму.
Двухдатчиковая схема.
Датчики линии включены в режим «аналоговый»
*/

#define DEBUG true //показывать сообщения

//порты, к которым подключёны датчики линии
//Правый датчик линии
const int rightLineTrackerPort = A8;
//Левый датчик линии
const int leftLineTrackerPort = A9;
//объявляем переменную для работы с датчиками
int rightLineTrackerValue = 0;
int leftLineTrackerValue = 0;
//переменные для сохранения значений детектора
int averageValue = 512;
//разница показаний датчиков на старте
int es = 0;
//разница показаний датчиков во время движения
int e = 0;
//коэффициент пропорциональности
float kp = 0.13;
//скорость подтормаживания
int vt = 0;

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM2PinTwo = 45;
// motor1 M2 (Right)
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;
```

```
//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;
//скорость моторов на колесах
int speedMotor = 100;

void setup() {
  //конфигурируем выбранные порты на чтение
  pinMode(rightLineTrackerPort, INPUT);
  pinMode(leftLineTrackerPort, INPUT);

  //инициализируем соединение с ПК по последовательному порту
  //со скоростью 9600 бод
  Serial.begin(9600);
  //настраиваем на вывод порты M1
  pinMode(motorM1PinOne, OUTPUT);
  pinMode(motorM1PinTwo, OUTPUT);

  //настраиваем на вывод порты M2
  pinMode(motorM2PinOne, OUTPUT);
  pinMode(motorM2PinTwo, OUTPUT);

  if (DEBUG) Serial.println(«Start programm»);
  delay(100);

  //при старте датчики робота точно над границей черной линии
  //читаем уровни сигнала на выбранном аналоговом порту
  rightLineTrackerValue=analogRead(rightLineTrackerPort);
  leftLineTrackerValue=analogRead(leftLineTrackerPort);
  //разница показаний датчиков на старте
  es = leftLineTrackerValue - rightLineTrackerValue;
  if (DEBUG) Serial.println(es);
}
void loop() {
//читаем показания датчиков линии на аналоговых портах
rightLineTrackerValue=analogRead(rightLineTrackerPort);
leftLineTrackerValue=analogRead(leftLineTrackerPort);
//ошибка отклонения датчиков от нормального положения над линией
//фильтр по среднему арифметическому
e = (e + (leftLineTrackerValue - rightLineTrackerValue - es))/2;
if (DEBUG){
  Serial.print(«e:»);
  Serial.println(e);
}
```

```
//рассчитаем управляющее воздействие на моторы в зависимости
//от смещения датчика от среднего положения над линией
int u=kp * e;
if (DEBUG){
  Serial.print(«u:»);
  Serial.println(u);
}

//подтормаживание при отклонении датчиков от нормального положения
vt = abs(e)/150;
if (DEBUG){
  Serial.print(«vt:»);
  Serial.println(vt);
}

//расчет мощности левого мотора
powerMotor1(speedMotor - u - vt);
//расчет мощности правого мотора
powerMotor2(speedMotor + u - vt);

//задаем паузу в 10 мс
delay(10);
}

//управление мотором1
void powerMotor1(int speedValue){
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor1:»);
      Serial.println(speedValue);
```

```
  }
}

//управление мотором2
void powerMotor2(int speedValue){
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

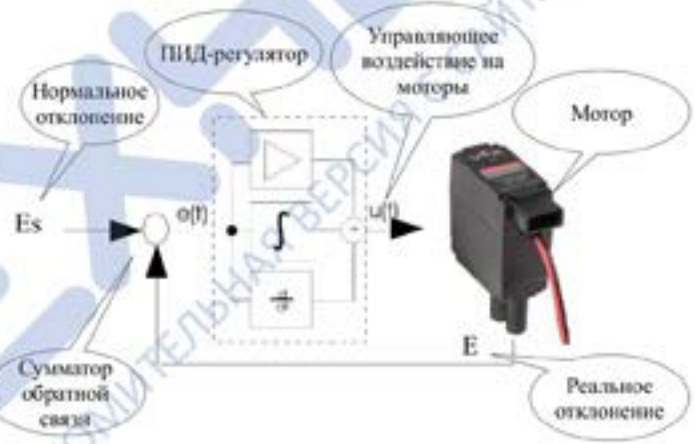
Перед стартом установите робота ClawBot на поле таким образом, чтобы оба его датчика линии находились над одинаковым цветом поля. При включении питания робот откалибрует показания датчиков, после чего его нужно установить на трассу для следования по линии. Вы можете настроить этот скетч под своего робота, изменяя скорость, коэффициент пропорциональности и скорость подтормаживания на поворотах. Во время прохождения робота по трассе убедитесь, что робот уверенно проходит перекрестки и пунктиры линии, не сбиваясь с трассы.

3.10 Работа с ИК-датчиками для движения по линии. ПИД-регулятор.

В предыдущей главе мы использовали в конструкции робота двухдачиковую схему и в программе контроллера применили пропорциональный регулятор, поэтому робот уверенно проходит перекрестки, движется плавно, но все еще заметны колебания при следовании робота вдоль линии и на поворотах. Как же сделать так, чтобы при движении робот следовал этой линии как можно точнее, «как по рельсам»? Для этого необходимо постоянно собирать статистику о реальном отклонении от линии и управлять мощностью двигателей, основываясь на получаемых значениях.

Нам поможет ***ПИД-регулятор.*** Он может быть как аппаратным, т.е. быть выполнен в виде отдельной платы, так и программным, т.е. описанным в программе. ПИД-регулятор можно легко реализовать в скетче Arduino, поэтому рассмотрим именно программный вариант.

Аббревиатура ПИД означает «Пропорциональный, Интегральный, Дифференциальный». Эти три термина описывают три элемента ПИД-регулятора. Каждый из элементов можно сравнить с некой «математической шкатулкой», которая получает данные о желаемом и о реальном состоянии, как-то их перерабатывает, а на выходе выдаёт рекомендованный сигнал, который должен приблизить реальное состояние к желаемому. Выходы этих элементов складываются между собой и формируют итоговый управляющий сигнал для устройства.



Допустим, что мы хотим, чтобы робот с заданной нами скоростью двигался вдоль линии с нормальным отклонением Es датчиков от линии. Для этого подаём на моторы мощность `speedMotor`, которая заставит колеса робота вращаться с заданной скоростью. При этом датчики линии позволяют узнать реальное отклонение E. Как нужно изменить мощность моторов, т.е. как рассчитать ***управляющее воздействие*** на моторы, чтобы робот во время движения выполнил подруливание и вернулся к положению над линией с нормальным отклонением?

```
u = kp * error + ki * errorIntegral + kd * errorDerivatile
```

Как можно видеть, управляющее воздействие получается из трёх слагаемых элементов ПИД регулятора. Рассмотрим эти три элемента по отдельности.

Пропорциональная составляющая

Пропорциональная составляющая `kp * error` состоит из коэффициента kp и текущей ошибки error . Текущая ошибка — это просто разница между желаемым отклонением (es) от линии и реальным. В двухдатчиковой схеме реальное отклонение робота от линии — это разница между показаниями левого и правого датчиков линии:

```
error = leftLineTrackerValue - rightLineTrackerValue - es;
```

Коэффициент kp — это константа, которую мы должны подобрать единожды для нашей системы. Если коэффициент окажется слишком маленьким, подруливание будет происходить медленее, чем нужно было бы. А если слишком большим — подруливание будет избыточным, робот будет вилять, «болтаться», двигаясь вдоль линии.

Интегральная составляющая

Интегральная составляющая `ki * errorIntegral` добавляет системе стабильности на основе суммы предыдущих ошибок (отклонений). Т.е. основываясь на истории реакции системы. Значение накопленной ошибки errorIntegral рассчитывается как сумма всех ошибок error во времени.

Если рассматривать этот расчёт в наших реалиях, значение накопленной ошибки можно вычислить такой командой:

```
errorIntegral += error * dt;
```

в которой:

- `errorIntegral` — значение накопленной ошибки (ноль на первом шаге),
- `error` — текущая пропорциональная ошибка,
- `dt` — время, прошедшее с предыдущего шага.

Коэффициент `ki` , опять же — константа, подобранная единожды для нашей системы. Слишком малое значение снизит скорость реакции системы, слишком большое — осцилляцию (колебания).

Дифференциальная составляющая

Дифференциальная составляющая kd * errorDerivatile ещё больше улучшает систему, делая прогноз изменения реального значения в зависимости от скорости изменения ошибки. В нашем случае, значение дифференциальной ошибки errorDerivatile можно рассчитать по формуле:

```
errorDerivatile = (error - errorOld)/dt;
```

где:

- `error` — текущее значение пропорциональной ошибки
- `errorOld` — значение пропорциональной ошибки на предыдущем шаге
- `dt` — время, прошедшее с предыдущего шага

И снова, коэффициент `kd` — константа, подобранная единожды для нашей системы. Последствия от выбора слишком малого или слишком большого значения те же, что и в остальных случаях.

Всё вместе

Теперь мы знаем, что пропорциональное управление соответствует использованию текущей «сиюминутной» информации о системе, интегральное управление использует информацию о «прошлом» системы. Дифференциальное управление соответствует использованию «прогнозной» информации о системе для ее стабилизации. Коэффициенты ПИД-регулятора подбираются для каждого робота индивидуально, т.к. много различных параметров влияют на скорость робота: от конструктивных особенностей робота, до отклонений характеристик некоторых деталей. Для начала можно задать значения коэффициентов ПИД-регулятора и потом настроить методом подбора их наиболее оптимальные значения.

Настройка коэффициентов ПИД-регулятора подбором

Сначала обнулите коэффициенты ПИД-регулятора (`kp, ki, kd`), которые определяются вначале скетча и подберите значение `speedMotor` — это мощность двигателей, с которой робот нормально будет двигаться (не быстро). Затем изменяйте и экспериментально подбирайте такой коэффициент пропорциональной составляющей `kp`, чтобы робот уверенно двигался по линии (очень желательно добиться плавного движения, чтобы не было резких рывков). Если значение `kp` будет недостаточным, то робот просто будет терять линию на поворотах, но если значение превысит оптимальное, то робот начнет двигаться рывками. Причем, чем больше значение `kp` — тем больше возможные «рывки». Оптимальным значением `kp` будет такое, при котором робот движется плавно, и выше которого робот начнет «дергаться» при движении.

Следующим шагом нужно настроить значение коэффициента интегральной составляющей `ki`. От его значения зависит стабилизация скорости вращения колес. Верный признак оптимального значения `ki` — робот начинает хорошо входить в повороты, даже если искусственно слегка сталкивать с линии, например, рукой. Однако, если параметр `ki` превышает свое оптимальное значение, то движение робота дестабилизируется: когда должен двигаться прямо, он «виляет» на трассе, либо внезапно «срывается» с линии в вираж.

Последний шаг — настройка коэффициента дифференциальной составляющей `kd`. Он призван «сглаживать» резкие изменения двух других составляющих ПИД-регулятора, поэтому значение его должно быть отрицательным. Его можно сравнить с парашютом или с гидравлическим стабилизатором в стойке автомобиля. Он «сглаживает» мелкие «рывки» робота, но если его значение превысит оптимальное, то колеса ро-

бота начнут резко менять направление вращения вперед-назад, что может повредить редукторы на моторах.

Смело экспериментируйте со значениями этих коэффициентов, подбирая для своего робота оптимальные коэффициенты для ПИД-регулятора.

Настройка коэффициентов ПИД-регулятора методом Зиглера Никольса

Есть более «оптимизированный» метод подбора коэффициентов –метод Зиглера–Никольса. Данный метод работает не для любой системы, результаты получаются не самыми оптимальными. Но, зато, метод очень простой и годится для базовой настройки регулятора в большинстве систем.

Суть метода состоит в следующем:

1. Выставляем все коэффициенты (`Kp, Ki, Kd`) в `0`.
2. Начинаем постепенно увеличивать значение `Kp` и следим за реакцией системы. Нам нужно добиться, чтобы в системе начались устойчивые колебания (вызванные перерегулированием). Увеличиваем `Kp`, пока колебания системы не стабилизируются (перестанут затухать), и робот уверенно будет следовать вдоль линии.
3. Запоминаем текущее значение `Kp` (обозначим его `Ku`) и замеряем период колебаний системы (`Tu`). Замерить `Tu` просто: пускаем робота по линии, включаем секундомер, например, на смартфоне, и в течение 20 или 30 секунд считаем, сколько колебаний (подруливаний) влево (вправо не считаем!) сделает робот, двигаясь по линии.

Допустим, за 15 с робот сделал сделал 30 колебаний, тогда `Tu = 15 / 30 = 0.5 с.`

Все. Теперь используем полученные значения `Ku` и `Tu` для расчета всех параметров ПИД регулятора по формулам:

```
Kp = 0.6 * Ku
Ki = 2 * Kp / Tu
Kd = Kp * Tu / 8
```

Готово. Именно этот метод мы использовали в приведенном ниже скетче, поэтому значительно сэкономили время, которое потратили бы на подбор коэффициентов первым методом.

Наберите и загрузите в контроллер следующий скетч:

```
/*
Следование ClawBot с контроллером Технолаб
по линии по ПИД-регулятору.
2 датчика линии включены в режим «аналоговый»
*/

#define DEBUG true //показывать сообщения

//порты, к которым подключёны датчики линии
```

```
//Правый датчик линии
const int rightLineTrackerPort = A8;
//Левый датчик линии
const int leftLineTrackerPort = A9;
//объявляем переменную для работы с датчиками
int rightLineTrackerValue = 0;
int leftLineTrackerValue = 0;
//переменные для сохранения значений детектора
int averageValue = 512;
//разница показаний датчиков на старте
int es = 0;
//разница (ошибка) показаний датчиков во время движения
int error = 0;
//коэффициент пропорциональной составляющей
//при ki=0, kd=0 устойчивый kp = 0.14, присвоим его ku
float ku = 0.14;
float kp = 0.6 * ku;
//период колебаний системы при ku (с)
float tu = 0.95;
//коэффициент интегральной составляющей
float ki = 2*kp/tu;
//коэффициент дифференциальной составляющей
float kd = kp*tu/8;
//время предыдущего шага
long timeOld = 0;
//разница (ошибка) предыдущего шага
int errorOld = 0;
//разница (ошибка) интегральная
float errorIntegral = 0;

//скорость подтормаживания
int vt = 0;

//порт, к которому подключён мотор Arm через Motor Controller 29
const int armServoPin = 5;
//порт, к которому подключён мотор Claw через Motor Controller 29
const int clawServoPin = 2;

//порты, к которым подключены Моторы колес
// motor1 M1 (Left)
const int motorM1PinOne = 46;
const int motorM1PinTwo = 45;
// motor1 M2 (Right)
const int motorM2PinOne = 6;
const int motorM2PinTwo = 7;
```

```
//задаем направление моторов вращения в роботе
// motor1(левый) развернут и при движении робота вперед
//вращается назад, т.е. против часовой
int motor1SetReversed= true;
int motor2SetReversed= false;

//скорость моторов на колесах
int speedMotor = 100;

void setup() {
  //конфигурируем выбранные порты на чтение
  pinMode(rightLineTrackerPort, INPUT);
  pinMode(leftLineTrackerPort, INPUT);
  //инициализируем соединение с ПК по последовательному порту
  //со скоростью 9600 бод
  Serial.begin(9600);
  //настраиваем на вывод порты M1
  pinMode(motorM1PinOne, OUTPUT);
  pinMode(motorM1PinTwo, OUTPUT);

  //настраиваем на вывод порты M2
  pinMode(motorM2PinOne, OUTPUT);
  pinMode(motorM2PinTwo, OUTPUT);

  if (DEBUG) Serial.println(«Start programm»);
  delay(100);

  //при старте датчики робота точно над границей черной линии
  //читаем уровни сигнала на выбранном аналоговом порту
  rightLineTrackerValue=analogRead(rightLineTrackerPort);
  leftLineTrackerValue=analogRead(leftLineTrackerPort);
  //разница показаний датчиков на старте
  es = leftLineTrackerValue - rightLineTrackerValue;
  if (DEBUG) Serial.println(es);
}

void loop() {
//читаем уровни сигнала на выбранном аналоговом порту
rightLineTrackerValue=analogRead(rightLineTrackerPort);
leftLineTrackerValue=analogRead(leftLineTrackerPort);

// получаем текущее время (мс)и вычисляем сколько
// времени (с) прошло с предыдущего шага
long time = millis();
float dt = (time - timeOld)/1000.;
timeOld = time;
//ошибка отклонения датчиков от нормального положения над линией
```

```
error = (error + (leftLineTrackerValue - rightLineTrackerValue - es))/2;
if (DEBUG){
  Serial.print(«error:»);
  Serial.println(error);
}

//ошибка дифференциальная
float errorDerivatile = (error - errorOld)/dt;
//сохраним занчение ошибки
errorOld = error;
//ошибка интегральная
errorIntegral += error * dt;
//рассчет по ПИД-регулятору управляющего воздействия на моторы
int u = kp * error +
    ki * errorIntegral +
    kd * errorDerivatile;
if (DEBUG){
  Serial.print(«u:»);
  Serial.println(u);
}

//подтормаживание при отклонении датчиков от нормального положения
vt = abs(error)/150;
if (DEBUG){
  Serial.print(«vt:»);
  Serial.println(vt);
}

//расчет мощности левого мотора
powerMotor1(speedMotor - u - vt);
//расчет мощности правого мотора
powerMotor2(speedMotor + u - vt);

//задаем паузу в 2 мс
delay(2);
}

//управление мотором1
void powerMotor1(int speedValue){
  int direction = motor1SetReversed ? -1 : 1;
  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM1PinOne, LOW);
    digitalWrite(motorM1PinTwo, LOW);
   }
```

```
  if (direction*speedValue > 0)  {
    digitalWrite(motorM1PinOne, LOW);
    analogWrite(motorM1PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM1PinOne, abs(speedValue));
    digitalWrite(motorM1PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor1:»);
      Serial.println(speedValue);
  }
}

//управление мотором2
void powerMotor2(int speedValue){
  int direction = motor2SetReversed ? -1 : 1;

  //ограничить скорость в допустимых пределах
  speedValue = constrain(speedValue,-254,254);

  if (speedValue == 0)  {
    digitalWrite(motorM2PinOne, LOW);
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (direction*speedValue > 0)  {
    digitalWrite(motorM2PinOne, LOW);
    analogWrite(motorM2PinTwo, abs(speedValue));
   }
  if (direction*speedValue < 0)  {
    analogWrite(motorM2PinOne, abs(speedValue));
    digitalWrite(motorM2PinTwo, LOW);
   }
  if (DEBUG) {
      Serial.print(«motor2:»);
      Serial.println(speedValue);
  }
}
```

Теперь проведите испытания и попробуйте оптимально настроить коэффициенты ПИД регулятора для своего робота. В случае успеха ваш робот будет следовать вдоль черной линии так четко, словно он движется по рельсам.

3.11 Разработка комплексной системы управления робота

Выполнив слияние всех скетчей, написанных по отдельности, можно получить готовый скетч для робота, который будет двигаться по линии, а также детектировать объекты, подхватывать их манипулятором и убирать с пути.

Полностью готовый скетч выглядит следующим образом:

```
#include <Servo.h>

Servo grab; //объявили сервомодуль, отвечающий за хватательное движение
Servo roll; //объявили сервомодуль, отвечающий за выбрасывательное движение

//указываем пины, отвечающие за управление портом M1
const int motorM1PinOne = 6;
const int motorM1PinTwo = 7;

//указываем пины, отвечающие за управление портом M2
const int motorM2PinOne = 46;
const int motorM2PinTwo = 45;

int sensorPort1 = 37;//указываем порт,
к которому подключён Line Tracker 1
int sensorPort2 = 36;//указываем порт,
к которому подключён Line Tracker 2
int sensorPort3 = 35;//указываем порт,
к которому подключён Line Tracker 3
```

```
int sensorValue1=0; //объявляем переменную для Line Tracker 1
int sensorValue2=0; //объявляем переменную для Line Tracker 2
int sensorValue3=0; //объявляем переменную для Line Tracker 3

const int input = 48; //объявили порт для управляющего сигнала
const int output = 49; //объявили порт для считанного сигнала

unsigned int time_us=0; //объявили переменную для времени
unsigned int distance_sm=0; //объявили переменную для расстояния
unsigned int distance=0; //объявили переменную для расстояния

void setup() {
pinMode(sensorPort1, INPUT);
pinMode(sensorPort2, INPUT);
pinMode(sensorPort3, INPUT);

//конфигурируем пины порта M1 на вывод сигнала
pinMode(motorM1PinOne, OUTPUT);
pinMode(motorM1PinTwo, OUTPUT);
//конфигурируем пины порта M2 на вывод сигнала
pinMode(motorM2PinOne, OUTPUT);
pinMode(motorM2PinTwo, OUTPUT);

pinMode(input, OUTPUT);
//сконфигурировали порт для управляющего сигнала на выход
pinMode(output, INPUT);
//сконфигурировали порт для считанного сигнала на вход

grab.attach(2);
//указываем порт, к которому подключён мотор «клешни»
roll.attach(5);
//указываем порт, к которому подключён мотор штанги
Serial.begin(9600);
}

void loop() {
forvard();
distance = ultrasonic();

Serial.println(distance);
if (distance <= 20) {
//если дистанция до объекта менее 20 см, то захватываем его
 reveal();
// вызов функции раскрытия хвата
 delay(1100);
stopMove();
toclose();
```

```
//вызов функции сомкнутия хвата
 delay(500);

reroll_up();
//поднимаем руку
 delay (2000);
//ожидание 2 сек.

 reveal();
//вызов функции раскрытия хвата
 delay (2000);
//ожидание 2 сек.

 toclose();
//вызов функции сомкнутия хвата
 reroll_down();
//опускаем руку
}

// считываем значения с портов
sensorValue1 = digitalRead(sensorPort1);
sensorValue2 = digitalRead(sensorPort2);
sensorValue3 = digitalRead(sensorPort3);

if (sensorValue3){
//контролируем значения с правого line tracker
right();
//если меньше определённого - подворачиваем
}

if (sensorValue1){
//контролируем значения с левого line tracker
left();
// если меньше определённого - подворачиваем
}

if (sensorValue1 && sensorValue2 && sensorValue3) {
//если все 3 детектора фиксируют линию
stopMove();
//остановка
}

delay(10);
}
```

```
void forvard (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}

void right (){
//правый мотор
digitalWrite (motorM2PinOne, HIGH);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, HIGH);
}

void left (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, HIGH);
//левый мотор
digitalWrite (motorM1PinOne, HIGH);
digitalWrite (motorM1PinTwo, LOW);
}

void stopMove (){
//правый мотор
digitalWrite (motorM2PinOne, LOW);
digitalWrite (motorM2PinTwo, LOW);
//левый мотор
digitalWrite (motorM1PinOne, LOW);
digitalWrite (motorM1PinTwo, LOW);
}

unsigned int ultrasonic (){
digitalWrite(input, HIGH);
//подаём управляющий сигнал
delayMicroseconds(10);
//удерживаем 10 микросекунд
digitalWrite(input, LOW);
//убираем управляющий сигнал

time_us=pulseIn(output, HIGH);
//замеряем длину импульса
```

```
distance_sm=time_us/58;
//пересчитываем в сантиметры
Serial.println(distance_sm);
//передаём значение в порт
return distance_sm;
}

void reveal (){
grab.writeMicroseconds(1800);
delay (700);

 grab.writeMicroseconds(1500);
}

void toclose (){
grab.writeMicroseconds(1200);
delay (700);
grab.writeMicroseconds(1500);
}

void reroll_up (){
roll.writeMicroseconds(1800);
delay (2000);
roll.writeMicroseconds(1500);
}

void reroll_down (){
roll.writeMicroseconds(1200);
delay (2000);
roll.writeMicroseconds(1500);
}
```


ПРИЛОЖЕНИЕ №1 РУКОВОДСТВО ПО СБОРКЕ CLAWBOT

ЭКЗАМЕН ТЕХНОЛАБ

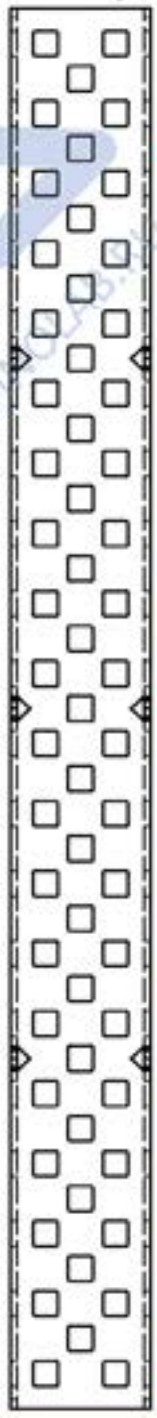
РУКОВОДСТВО ПО СБОРКЕ CLAWBOT

Приложение № 1. Руководство по сборке Clawbot

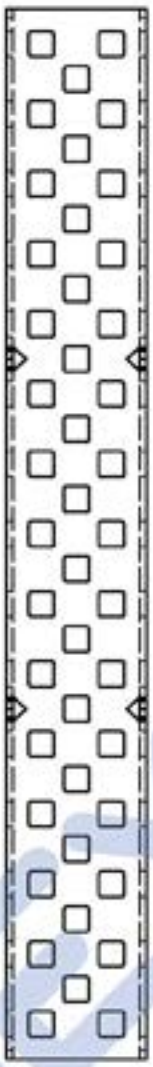
[A20] Рама угловая (20 отверстий в длину)

[R16] Рама (16 отверстий в длину)

[C20] Соединительная пластина (20 отверстий в длину)



[C15] Соединительная пластина (15 отверстий в длину)



[B20] Пластина (с 20 отверстиями)

[SH-3] Вал (3 дюйма в длину)

[ZIP] Стяжка

[BF] ОПОРНАЯ ПЛАНКА

[ST-1]
РАЗДЕЛИТЕЛЬ

[BR-I]
ЗАКЛЕПКА (ВНУТР.)

[BR-O]
ЗАКЛЕПКА (ВНЕШН.)

[CPLR]
СОЕДИНИТЕЛЬ ВАЛОВ

[CP]
СОЕДИНИТЕЛЬНЫЙ ЭЛЕМЕНТ (ВНУТР.)

[SP4.8]
РАЗДЕЛИТЕЛЬ (4.8 ММ)

[NK]
#8-32 ГАЙКА

[NL]
#8-32 ГАЙКА С НЕЙЛОНОВЫМ ДЕРЖАТЕЛЕМ

[COL]
НАКОНЕНЕЧНИК ВАЛА С 8-32 x 1/8 ДЮЙМА ВИНТОМ

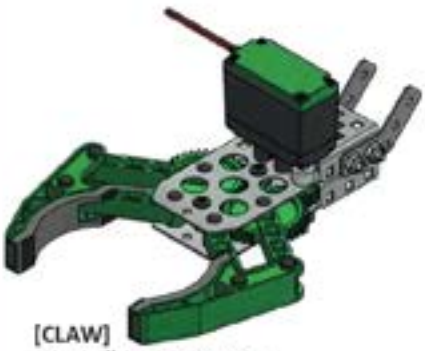
[SS-L]
ДЛИННЫЙ ВИНТ МОТОРА С НЕЙЛОНОВЫМ ДЕРЖАТЕЛЕМ

[SS-S]
КОРОТКИЙ ВИНТ МОТОРА С НЕЙЛОНОВЫМ ДЕРЖАТЕЛЕМ

[S12]
ВИНТ 8-32x1.5 ДЮЙМА

[S4]
ВИНТ 8-32x1/2 ДЮЙМА

[S2]
ВИНТ 8-32x1/4 ДЮЙМА



[CLAW]
УСТРОЙСТВО СХВАТА
(показано не в масштабе)

[M393]
ДВУХПРОВОДНОЙ МОТОР 393
(показано не в масштабе)

[MC29]
КОНТРОЛЛЕР МОТОРА 29
(показано не в масштабе)

[BST]
КРЕПЛЕНИЕ БАТАРЕИ
(показано не в масштабе)

[BATT]
ПЕРЕЗАРЯЖАЕМАЯ
БАТАРЕЯ 7.2В
(показано не в масштабе)

[CTX]
МИКРОКОНРОЛЛЕР
(показано не в масштабе)

[VNET]
КЛЮЧ VEXnet
(показано не в масштабе)

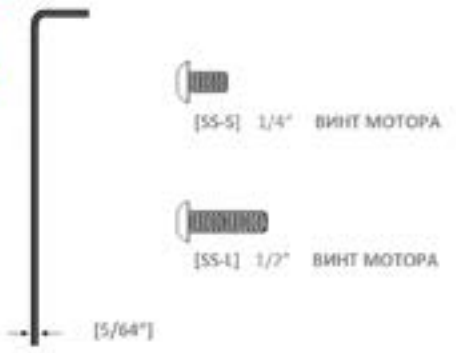
[G12]
ШЕСТЕРНЯ
(12 ЗУБЬЕВ)
(показано не в масштабе)

[G60]
ШЕСТЕРНЯ
(60 ЗУБЬЕВ)
(показано не в масштабе)

[G84]
ШЕСТЕРНЯ
(84 ЗУБЬЕВ)
(показано не в масштабе)

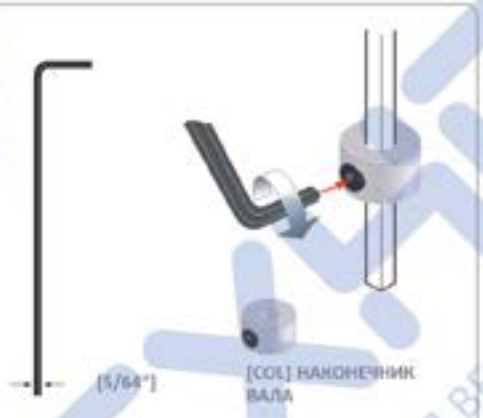
[W4]
КОЛЕСО 4 ДЮЙМА
(показано не в масштабе)

1



2

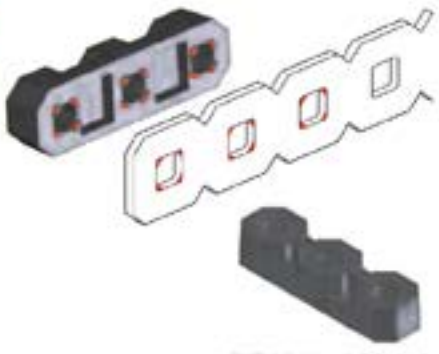
3



4

5

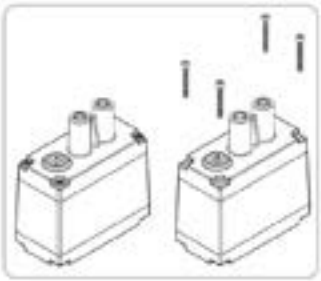
6



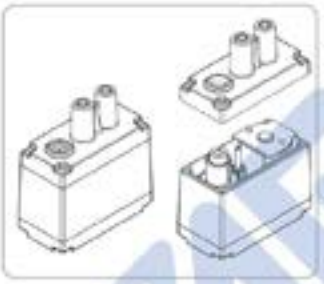
ВЫСОКАЯ СКОРОСТЬ/НИЗКИЙ КРУТЯЩИЙ МОМЕНТ для моторов (необязательная опция):

Для модификации двухпроводного мотора **393** в режим повышенной скорости - просто замените шестерню мотора на одну из прилагаемых шестерней для замены, следуя инструкции

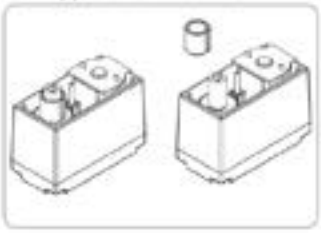
1. Открутите и удалите винты с передней части мотора.



2. Снимите крышку с передней части мотора. Не трогайте шестерни внутри.



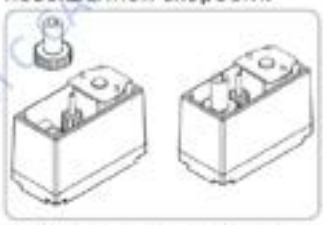
3. Снимите втулку, как показано на рисунке, и отложите в сторону. Она понадобится позже.



4. Снимите центральную шестерню и шестерню вала.

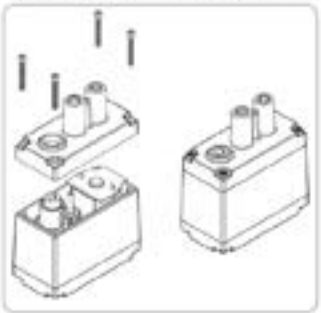
5. Установите центральную шестерню повышенной скорости.

6. Установите шестерню вала повышенной скорости.

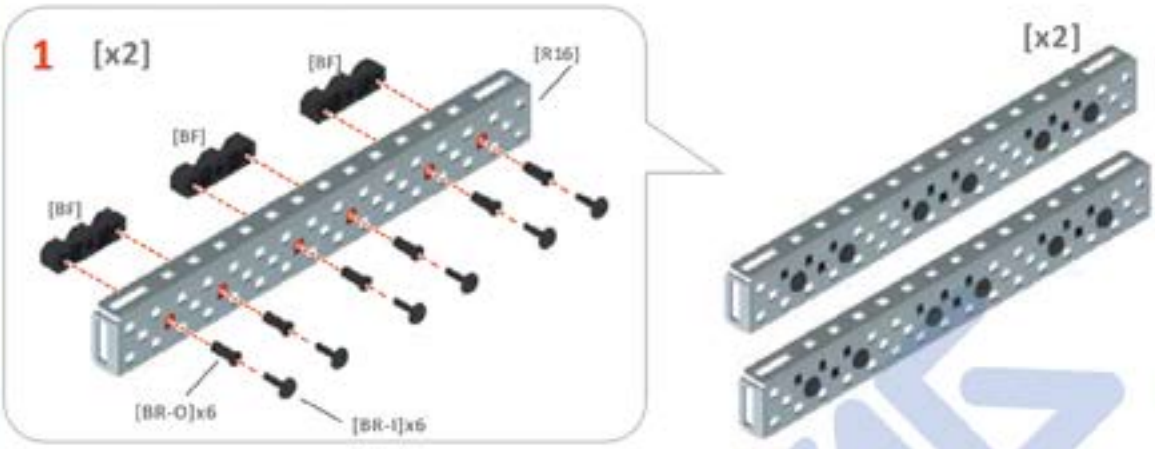


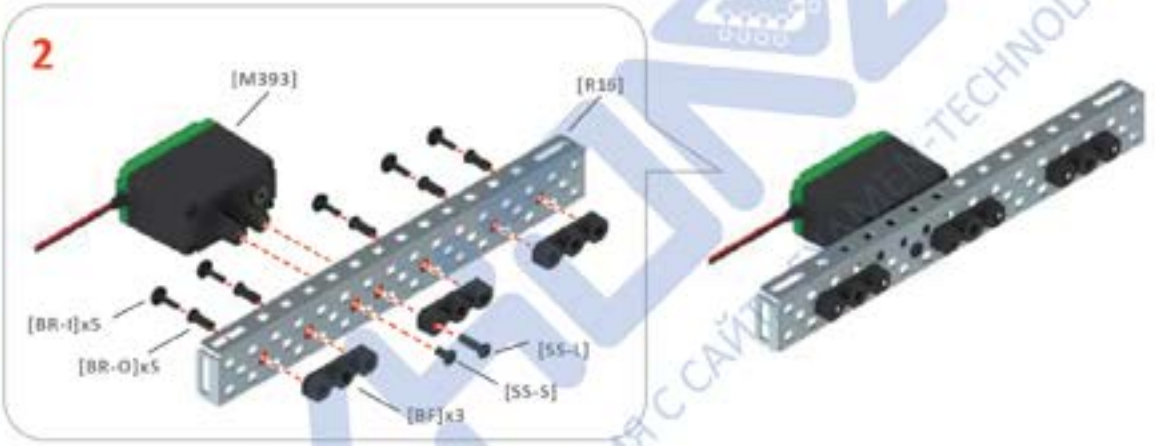
7. Установите втулку, снятую в шаге №3. Убедитесь, что она установлена как на рисунке.

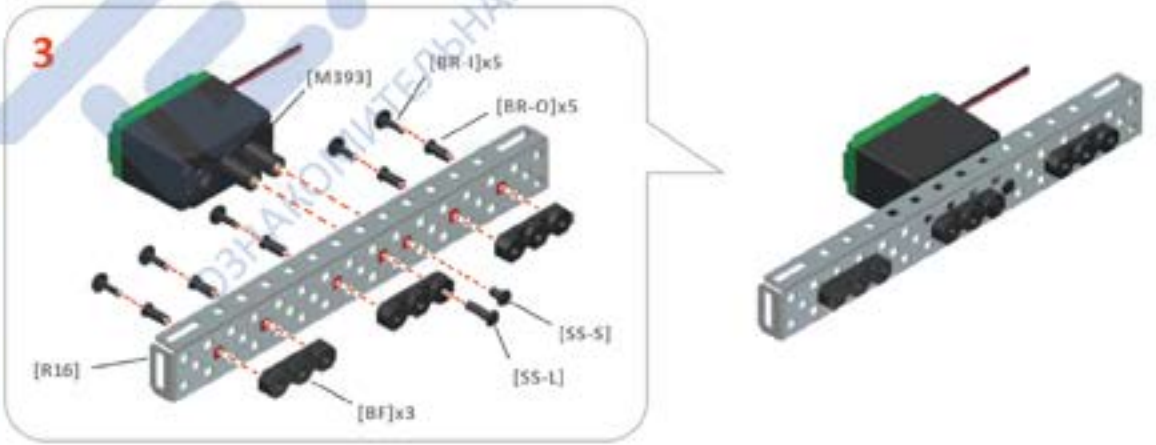
8. Установите обратно крышку и винты, снятые в шагах №1 и №2



Режим повышенной скорости обеспечивает ускорение вращения вала на 60% и снижение крутящего момента на 60%.

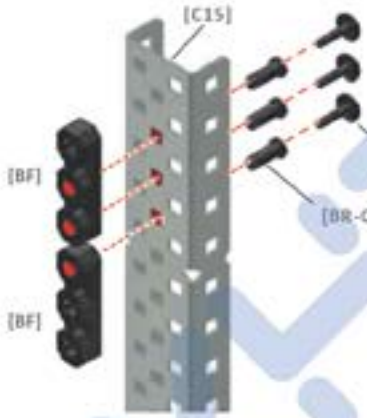




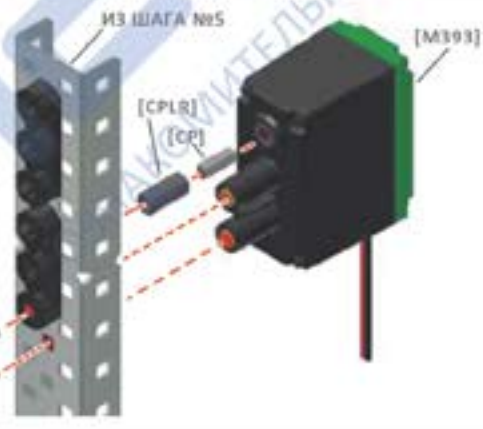


4

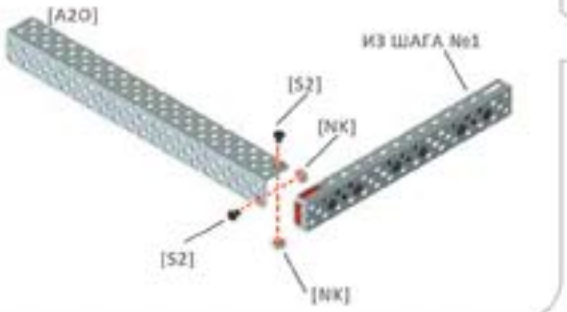
5



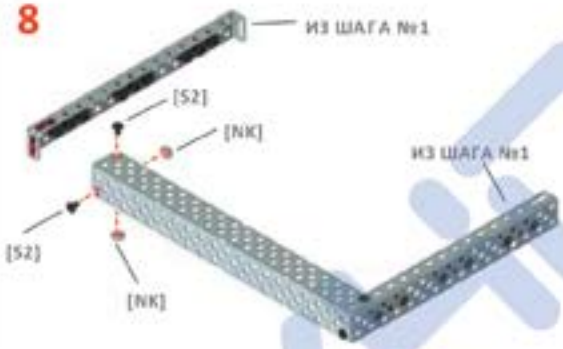
6



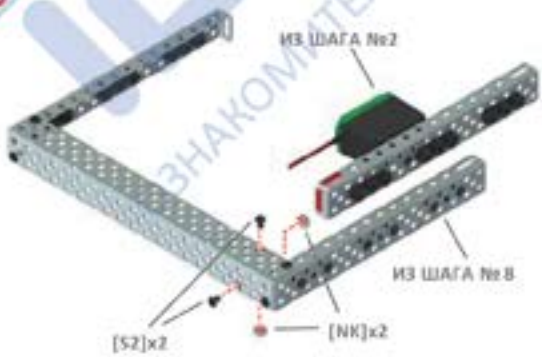
7



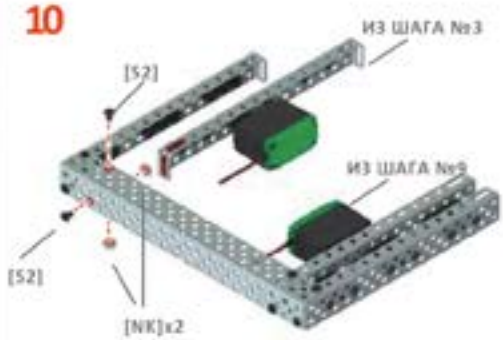
8



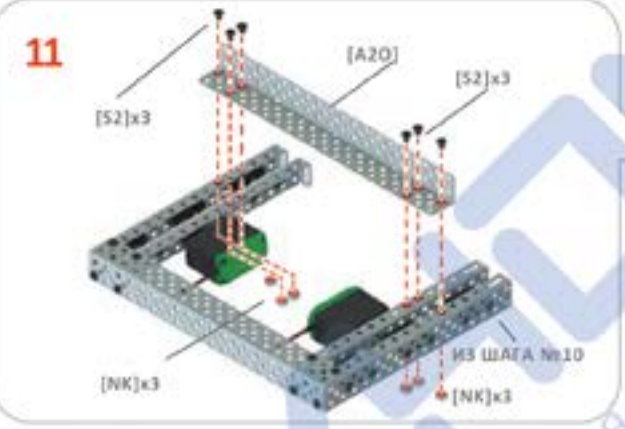
9



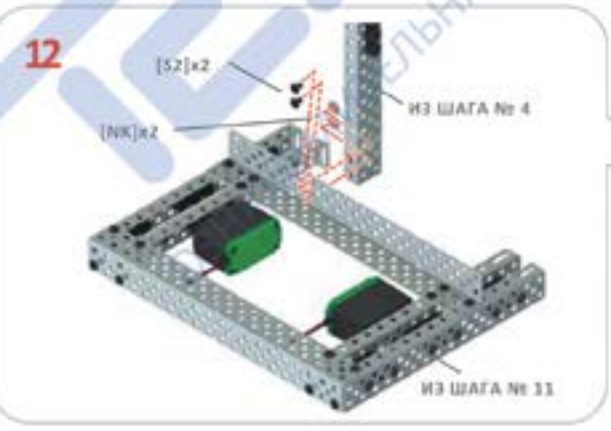
10



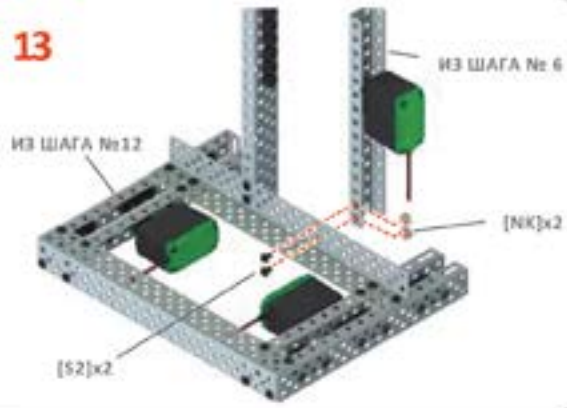
11

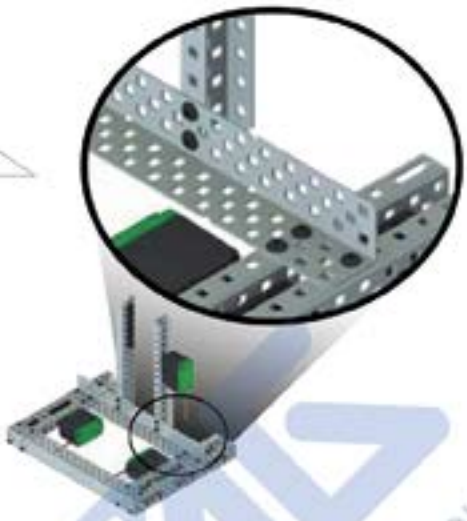


12

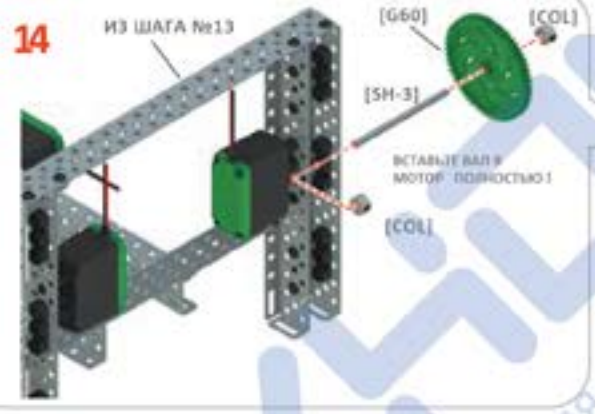


13

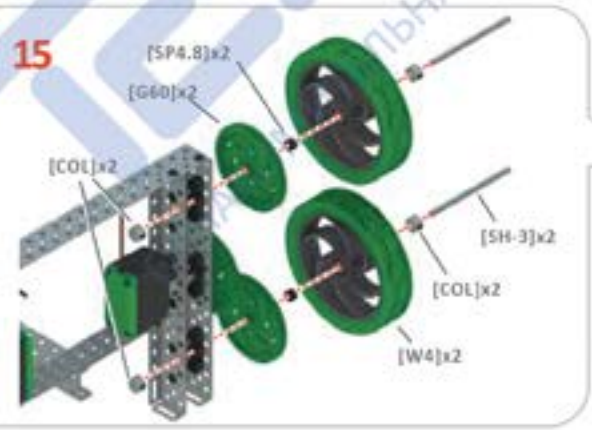


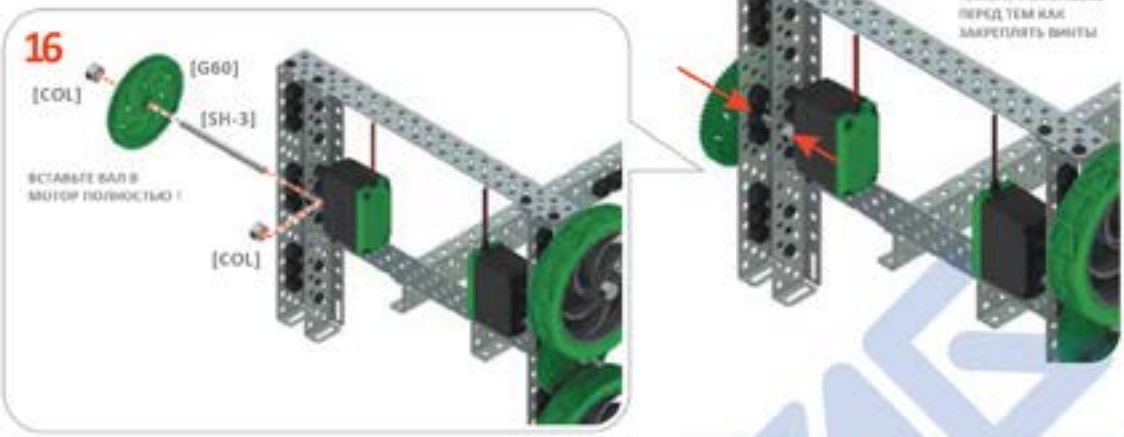


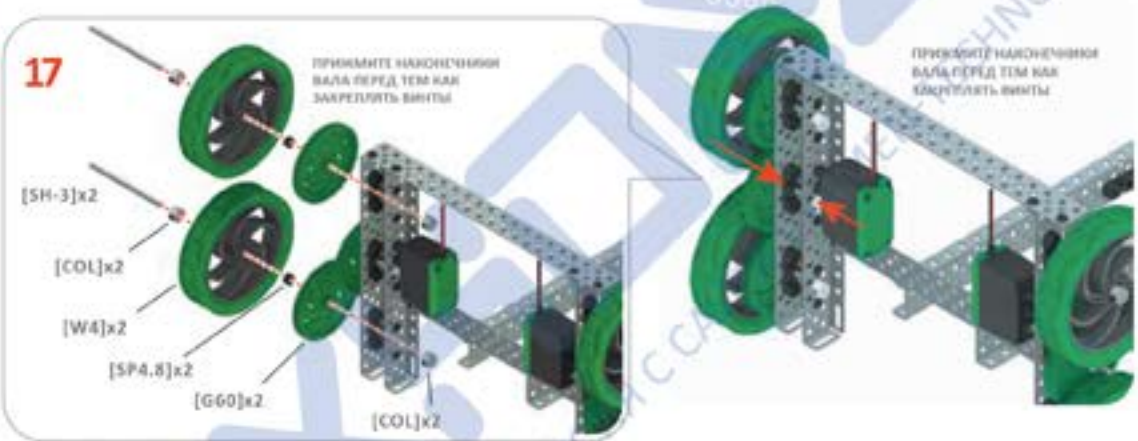
14

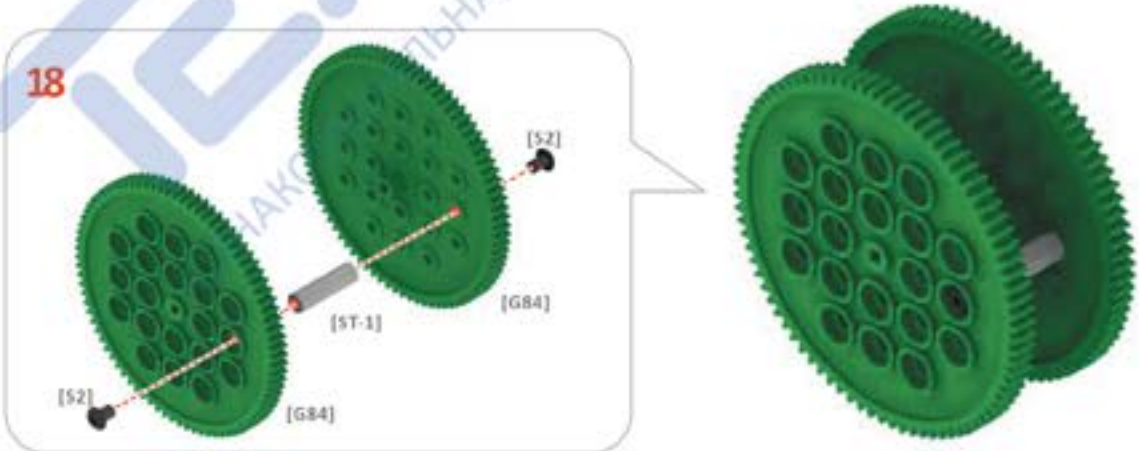


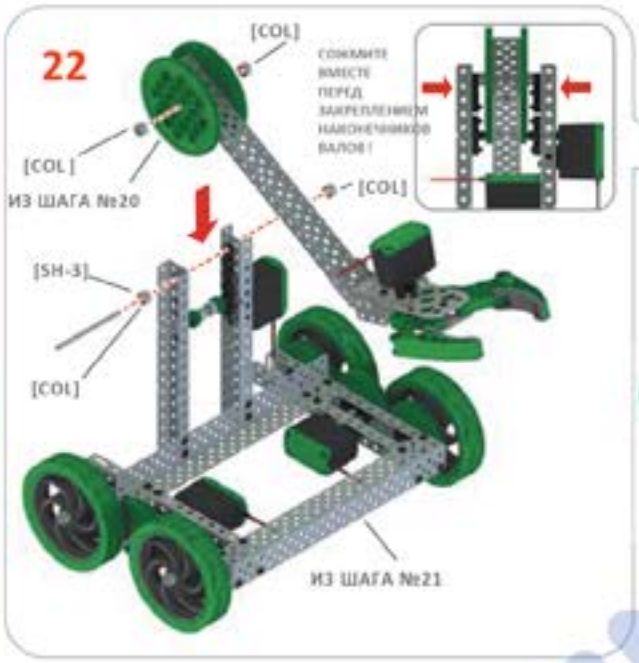
15

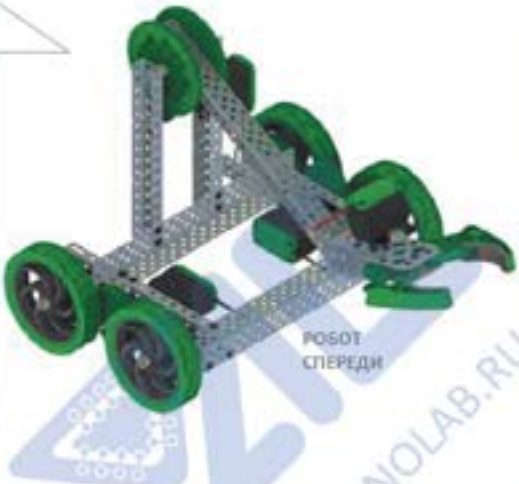



16
[COL]
[G60]
[SH-3]
ВСТАВЬТЕ ВАЛ В МОТОР ПОЛНОСТЬЮ !
[COL]
ПРИЖМИТЕ НАКОНЕЧНИКИ ВАЛА ПЕРЕД ТЕМ КАК ЗАКРЕПЛЯТЬ ВИНТЫ

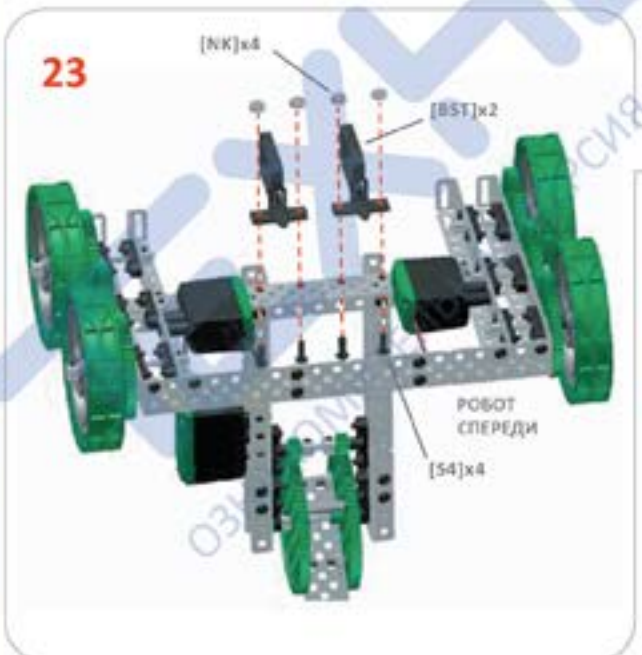

17
ПРИЖМИТЕ НАКОНЕЧНИКИ ВАЛА ПЕРЕД ТЕМ КАК ЗАКРЕПЛЯТЬ ВИНТЫ
[SH-3]x2
[COL]x2
[W4]x2
[SP4.8]x2
[G60]x2
[COL]x2
ПРИЖМИТЕ НАКОНЕЧНИКИ ВАЛА ПЕРЕД ТЕМ КАК ЗАКРЕПЛЯТЬ ВИНТЫ


18
[S2]
[G84]
[ST-1]
[S2]
[G84]



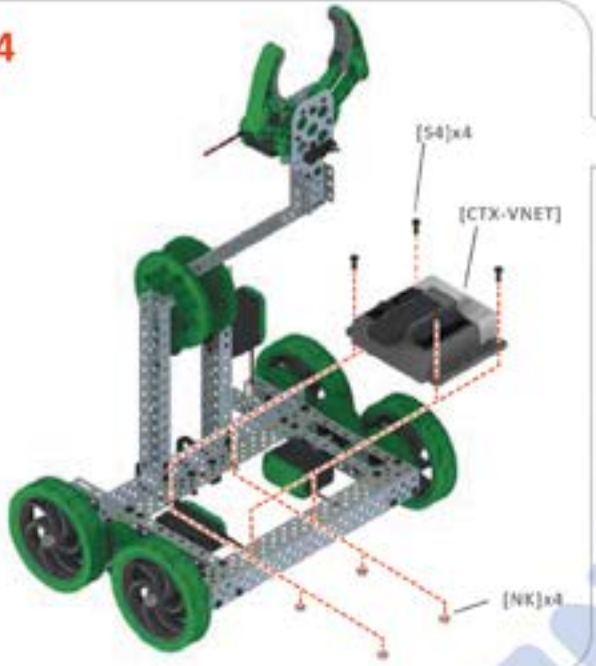


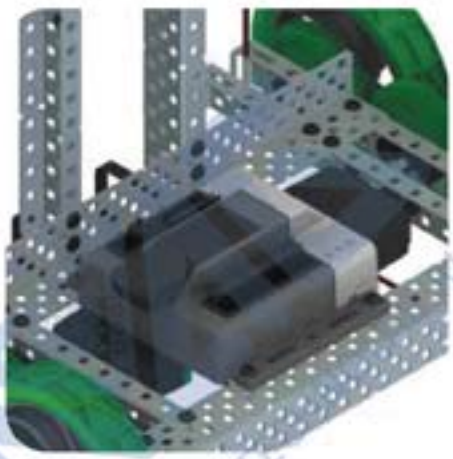
• ПРОВЕРЬТЕ, ЧТО "РУКА" ПОДНИМАЕТСЯ И ОПУСКАЕТСЯ ПЛАВНО, С НЕБОЛЬШИМ СОПРОТИВЛЕНИЕМ ОТ МОТОРА



УСТАНОВИТЕ КРЕПЛЕНИЕ БАТАРЕИ ПОД РАМОЙ

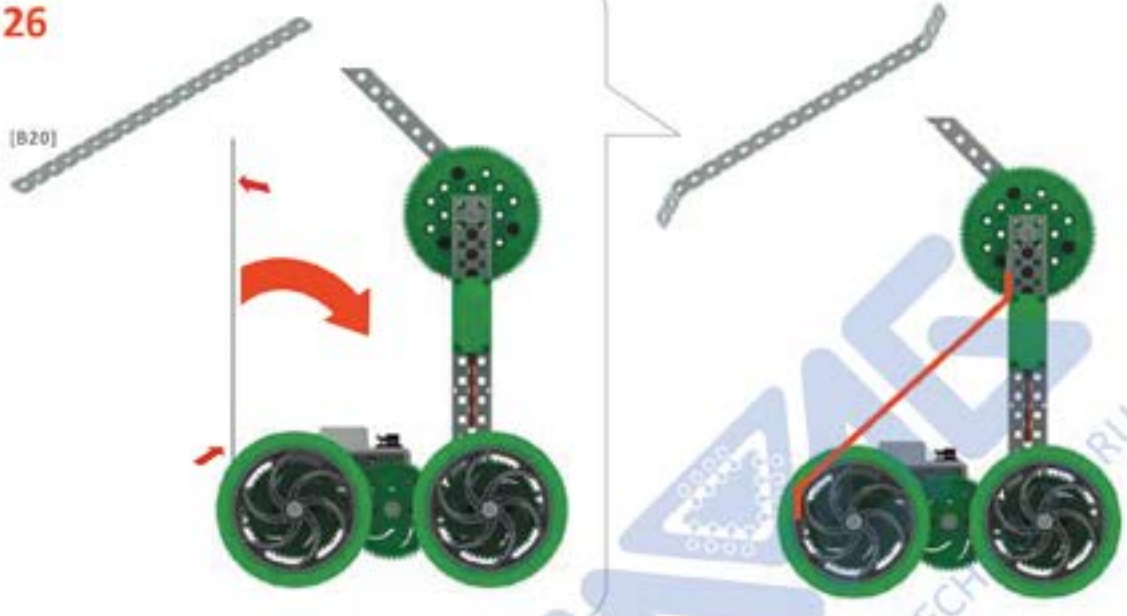
24



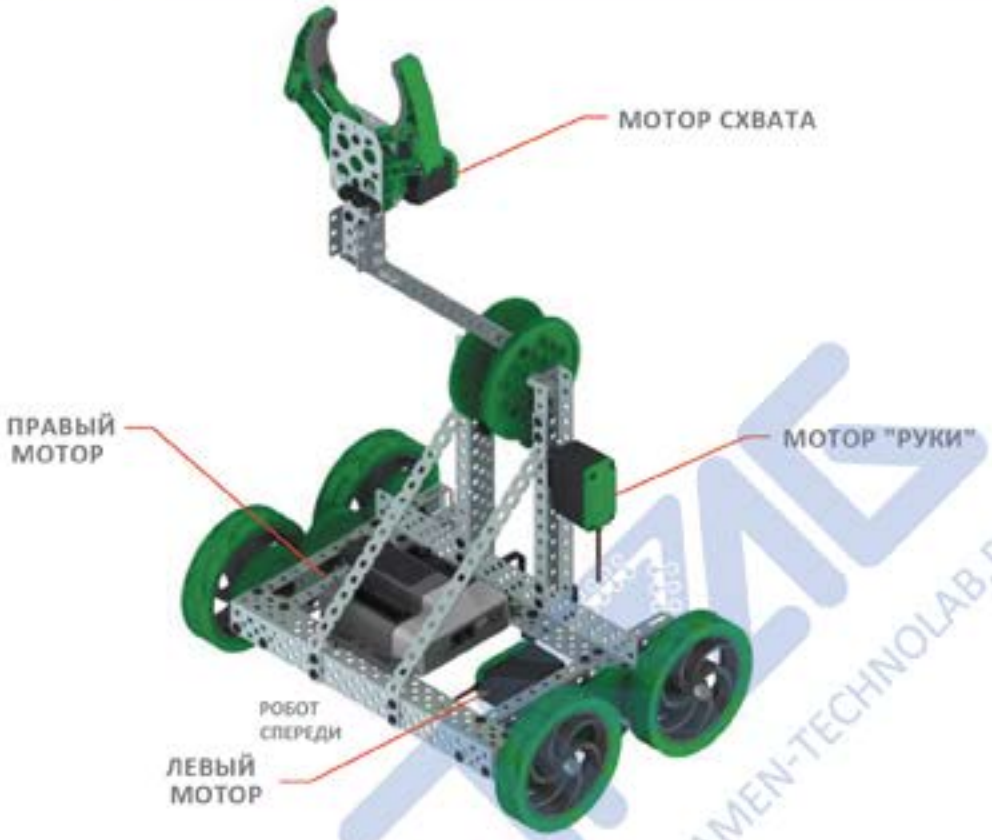


25

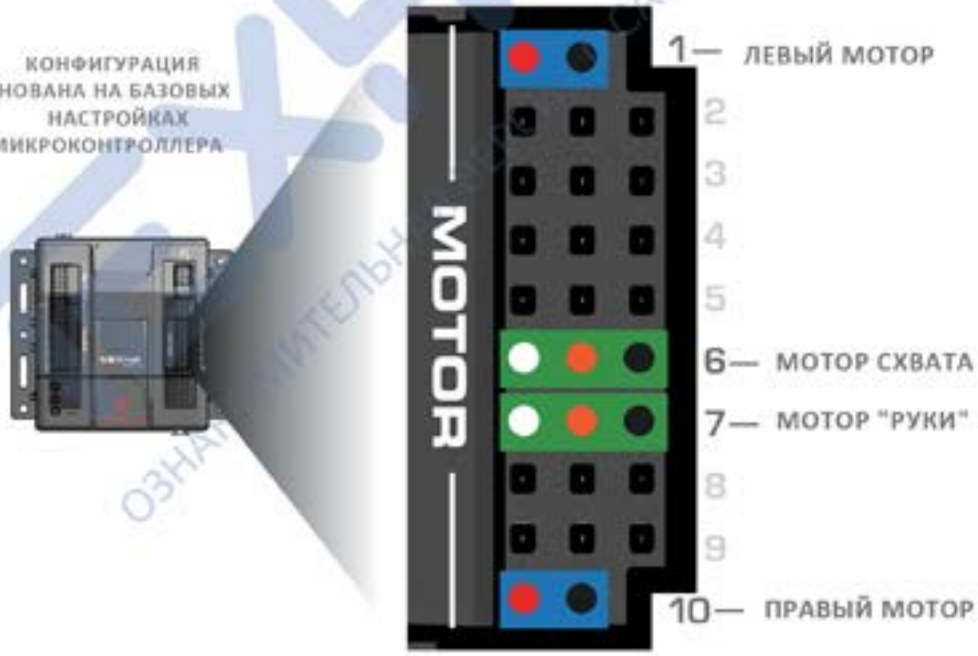
26

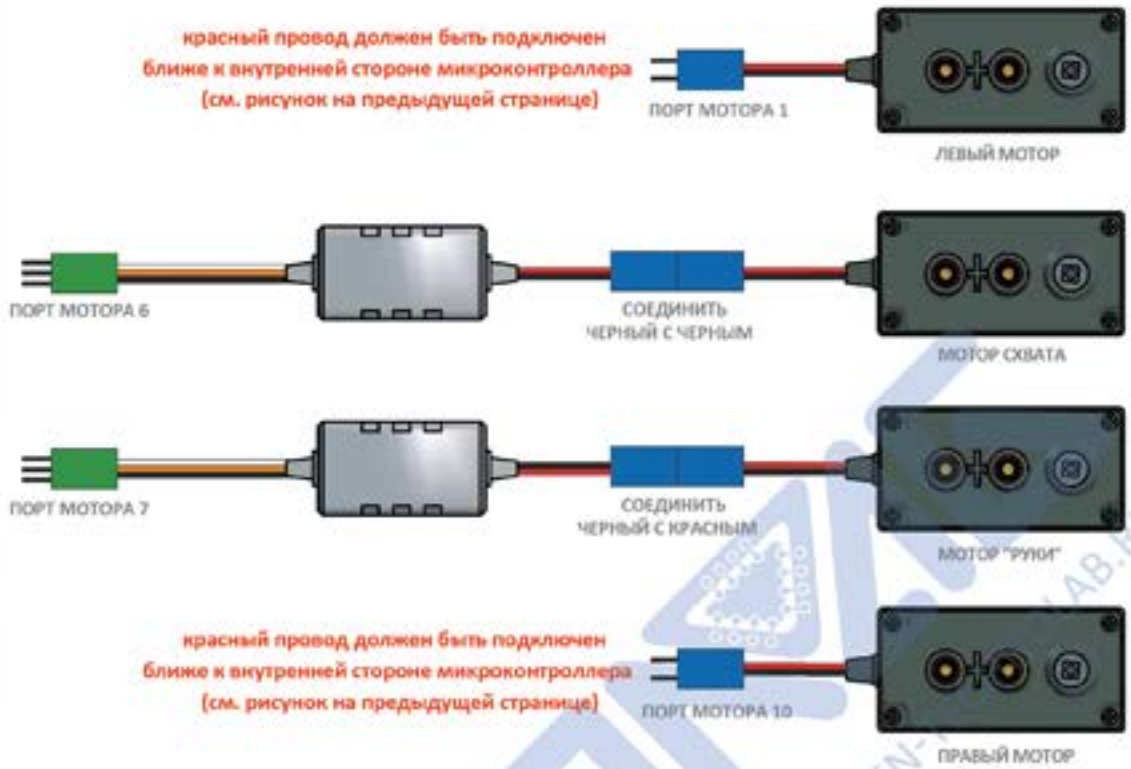


27



КОНФИГУРАЦИЯ ОСНОВАНА НА БАЗОВЫХ НАСТРОЙКАХ МИКРОКОНТРОЛЛЕРА





Более подробную информацию о микроконтроллере и системе управления VEXNet можно найти на сайте www.VEXRobotics.com/cortex (на английском языке)

ПРИЛОЖЕНИЕ №2 РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА С МАНИПУЛЯТОРОМ

ЭКЗАМЕН ТЕХНОЛАБ

РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА С МАНИПУЛЯТОРОМ

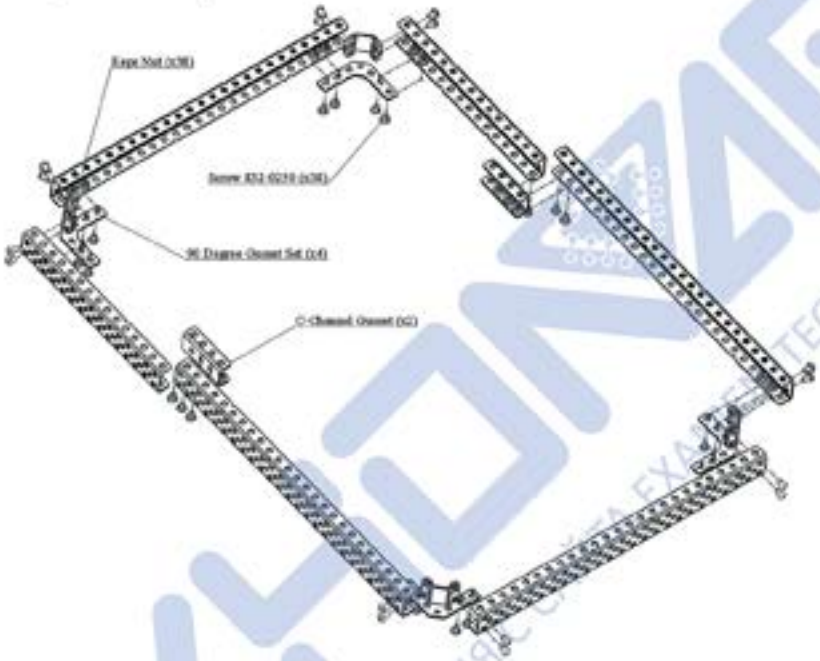
№2

Приложение № 2. Руководство по сборке мобильного робота с манипулятором

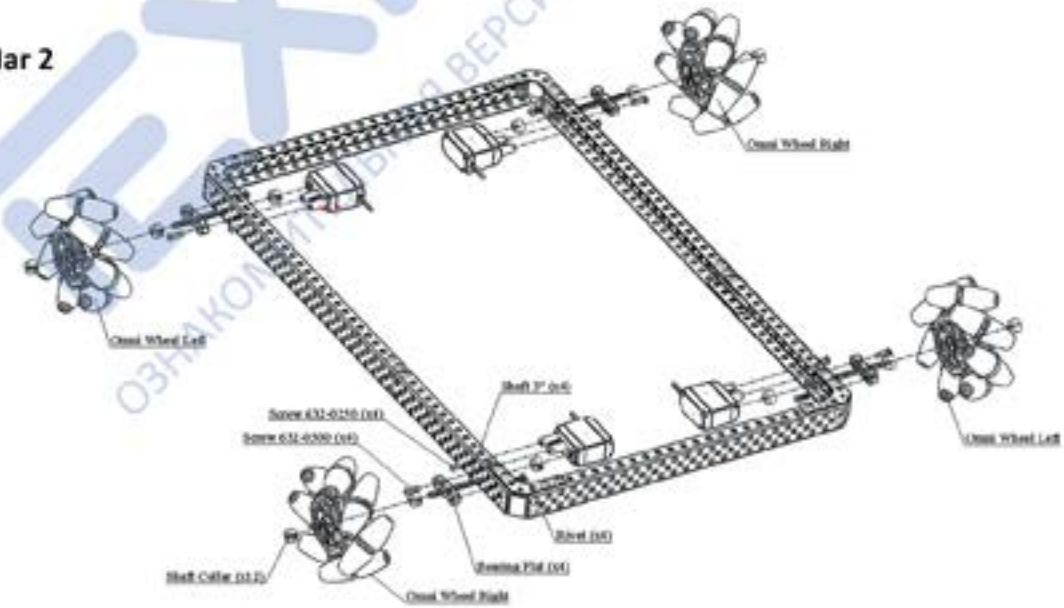
В данном руководстве используются дополнительные детали, не входящие в базовый набор.

Руководство носит рекомендательный характер и может быть изменено пользователем в соответствии с собственным техническим решением. В том числе данная инструкция содержит этапы сборки конструкции, на которых применяются металлические комплектующие, подвергнутые изменению формы, а именно – сгибанию или резке на части различной длины.

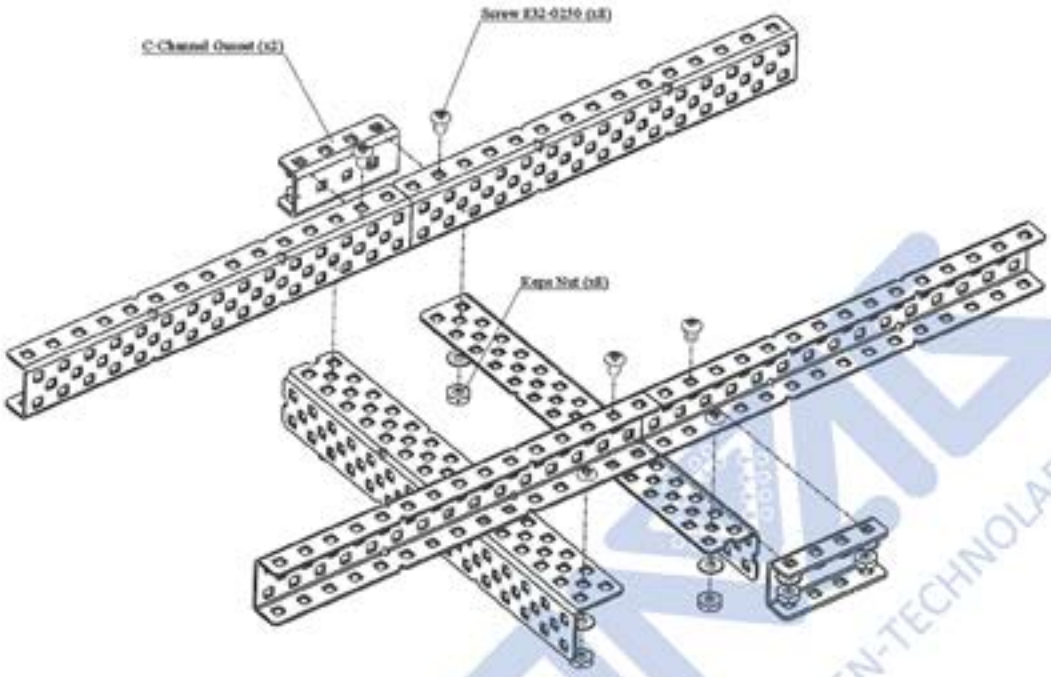
Шаг 1



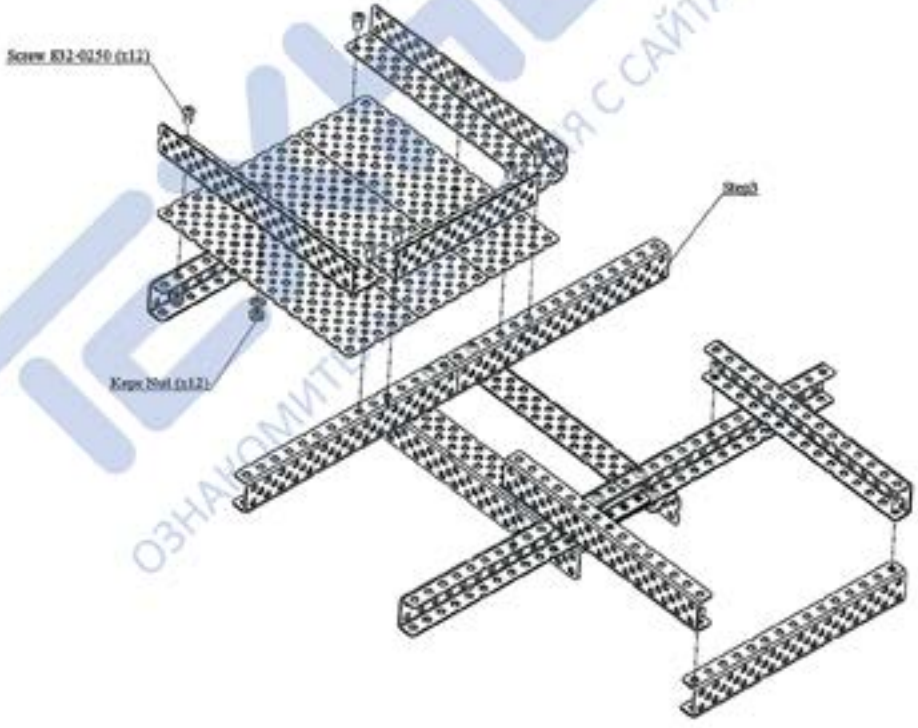
Шаг 2



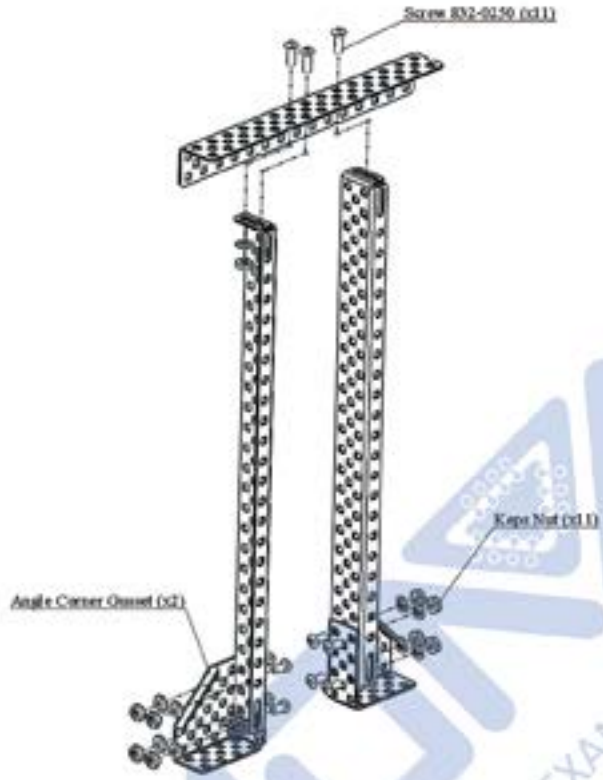
Шаг 3



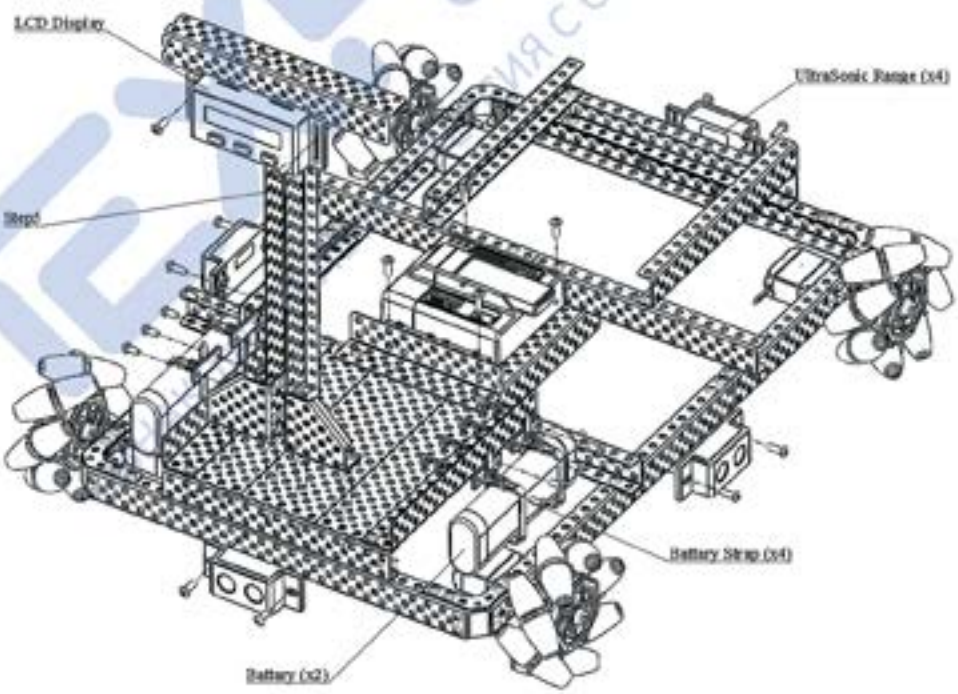
Шаг 4



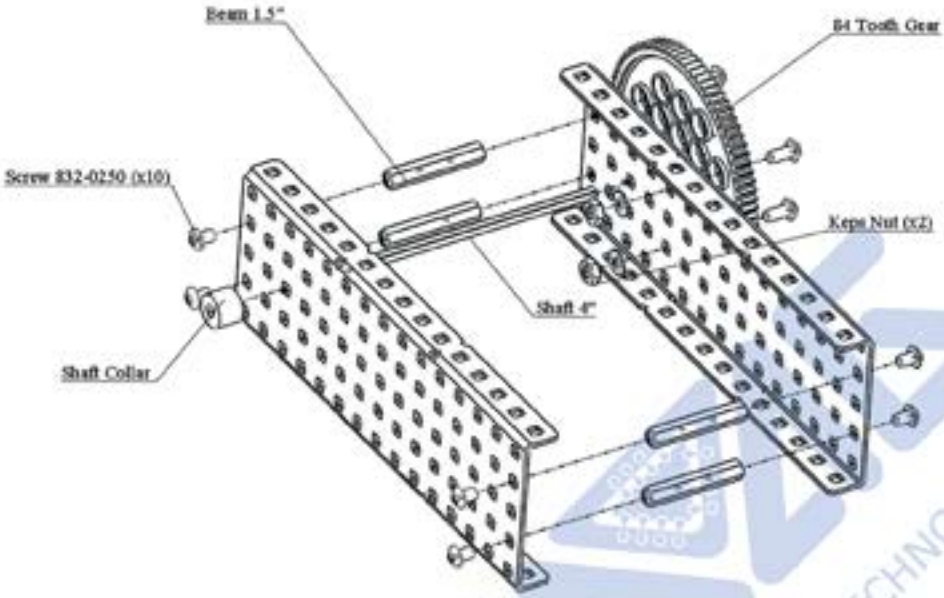
Шаг 5



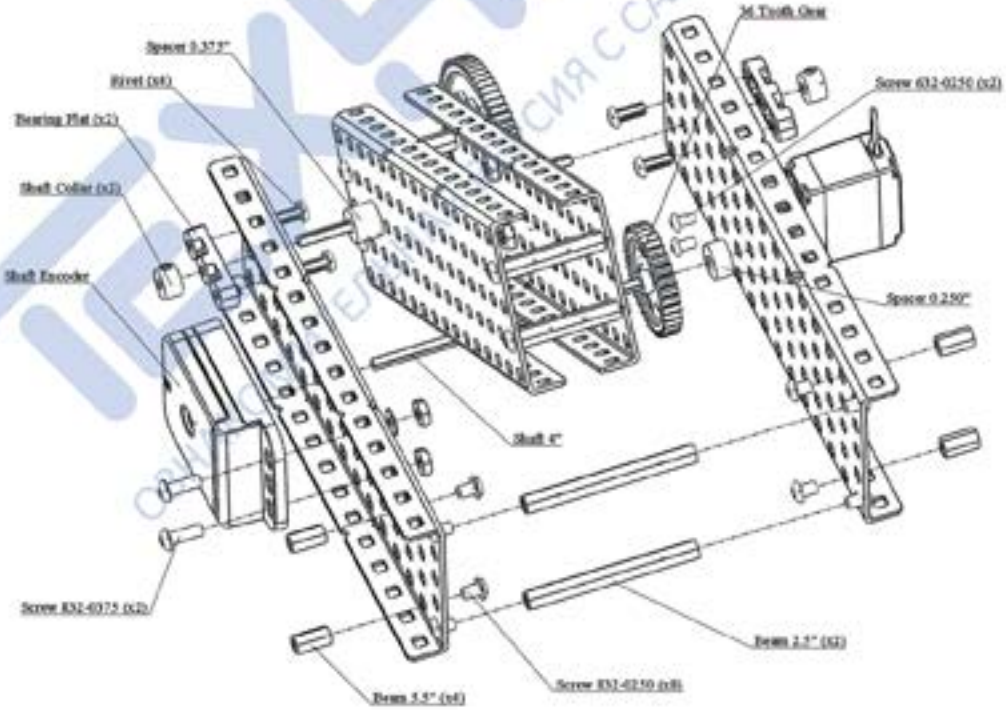
Шаг 6



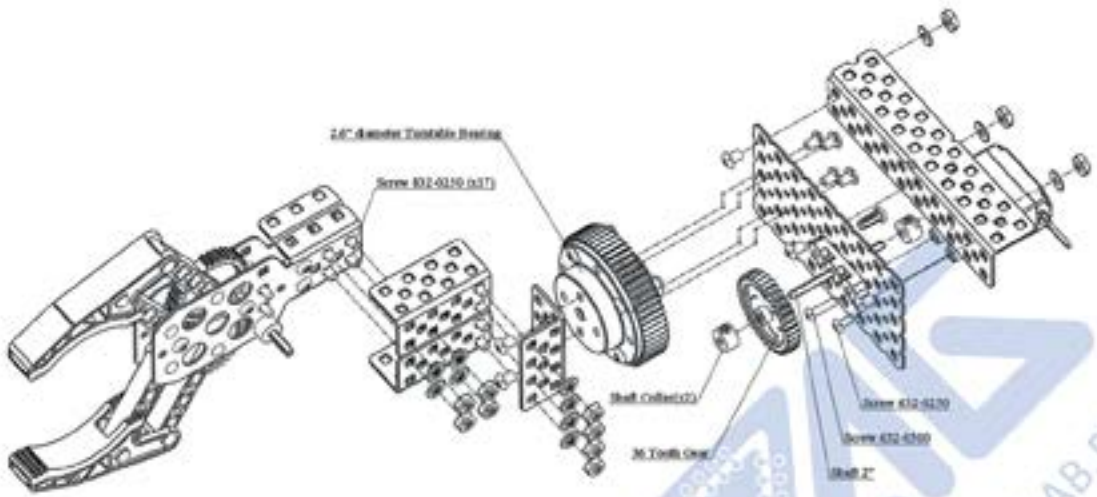
Шаг 7



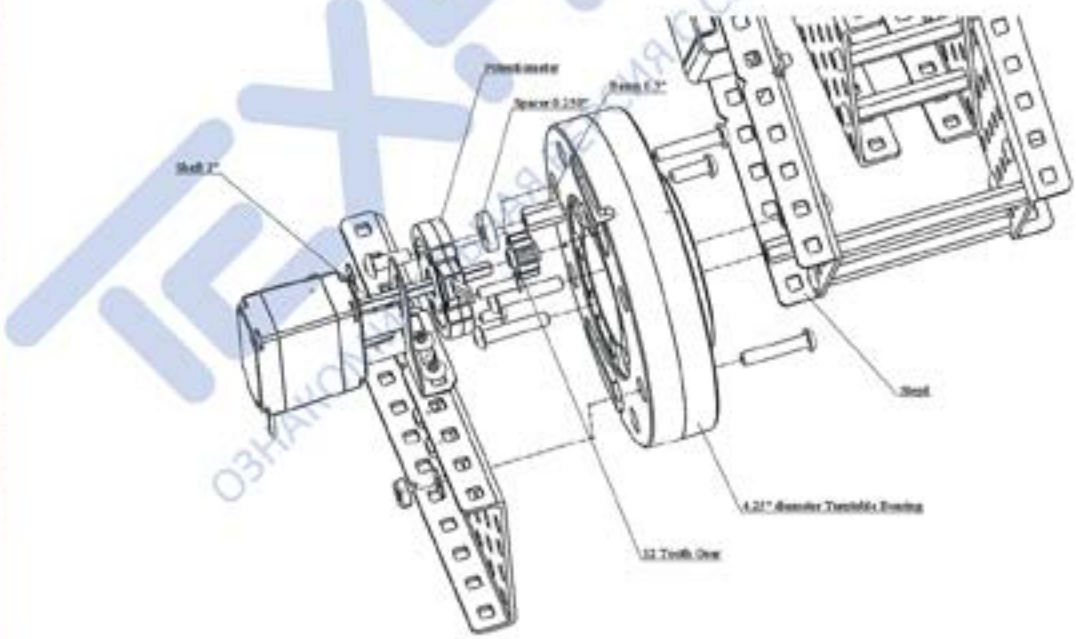
Шаг 8



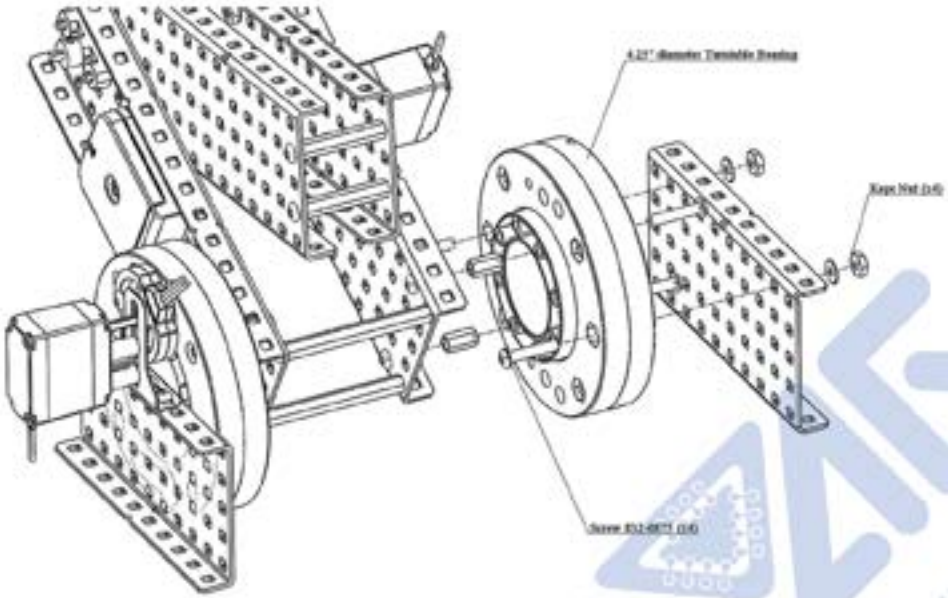
Шаг 9



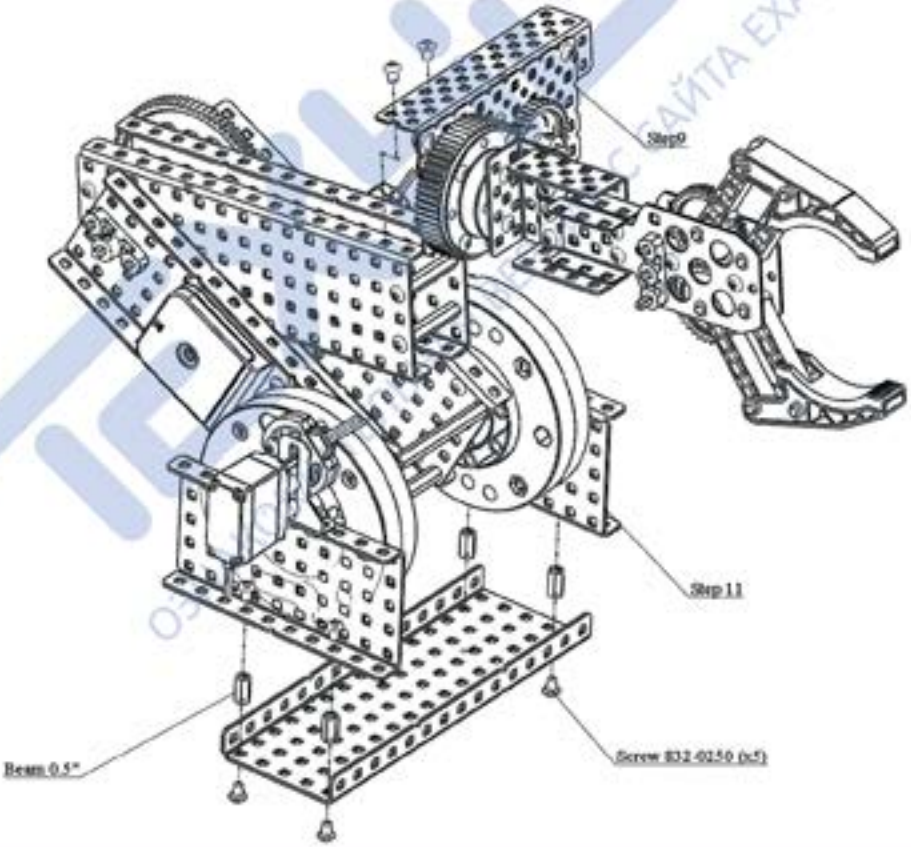
Шаг 10



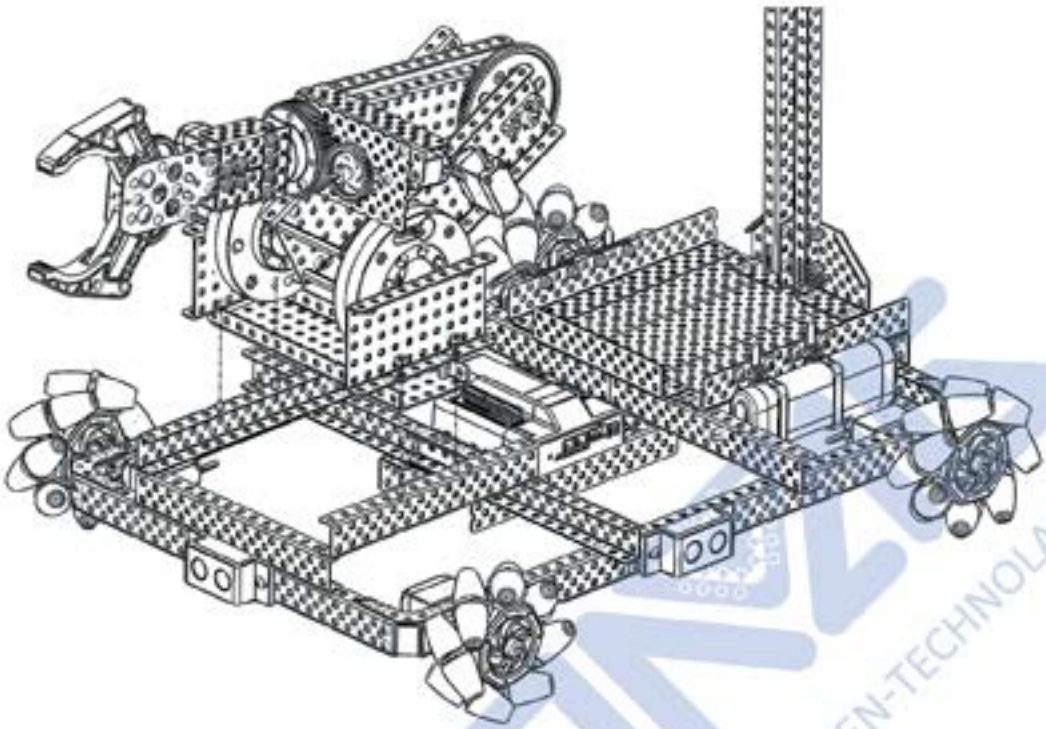
Шаг 11



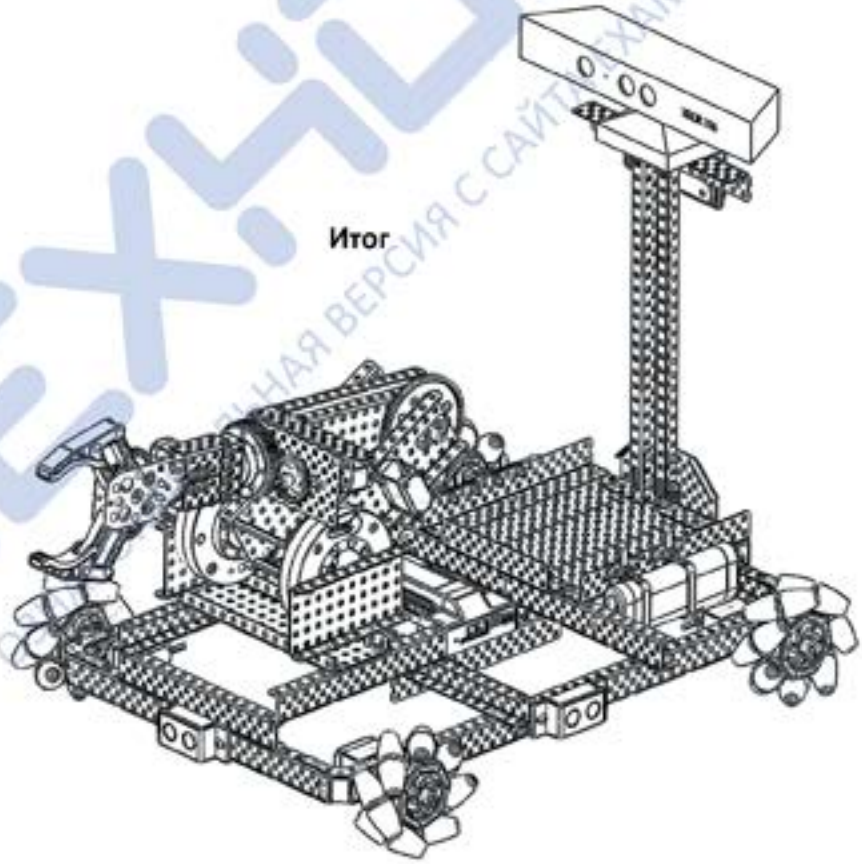
Шаг 12



Шаг 13



Итог



ПРИЛОЖЕНИЕ №3 РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА ПОВЫШЕННОЙ ПРОХОДИМОСТИ

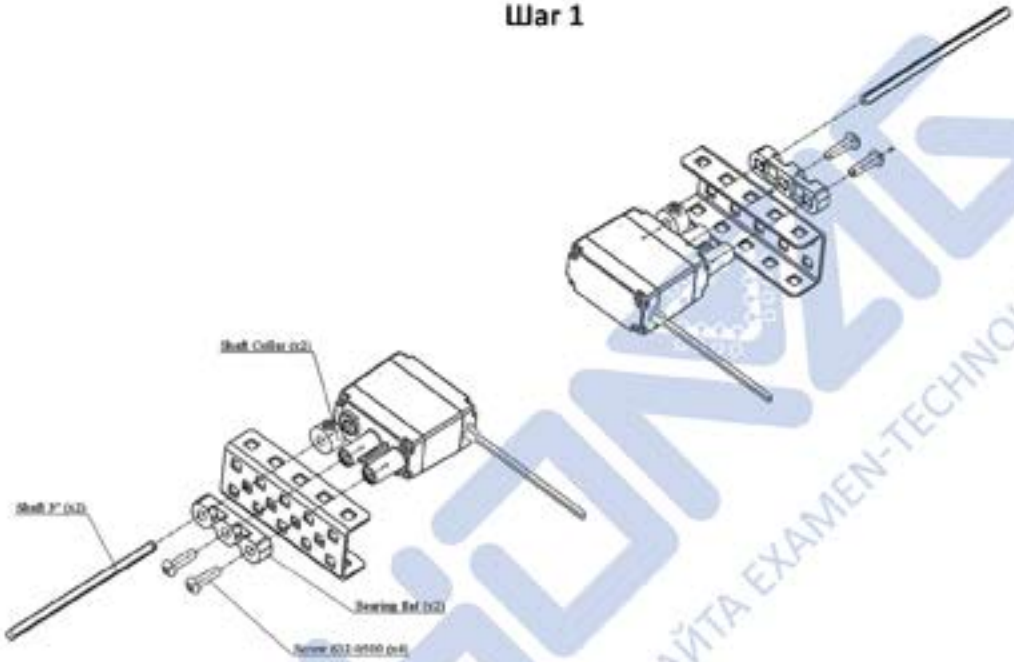
ЭКЗАМЕН ТЕХНОЛАБ

РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА ПОВЫШЕННОЙ ПРОХОДИМОСТИ

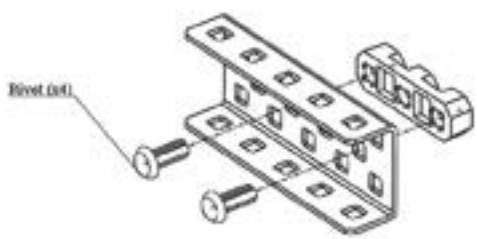
Приложение № 3. Руководство по сборке мобильного робота повышенной проходимости

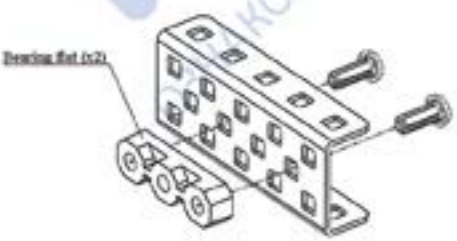
В данном руководстве используются дополнительные детали, не входящие в базовый набор.

Шаг 1

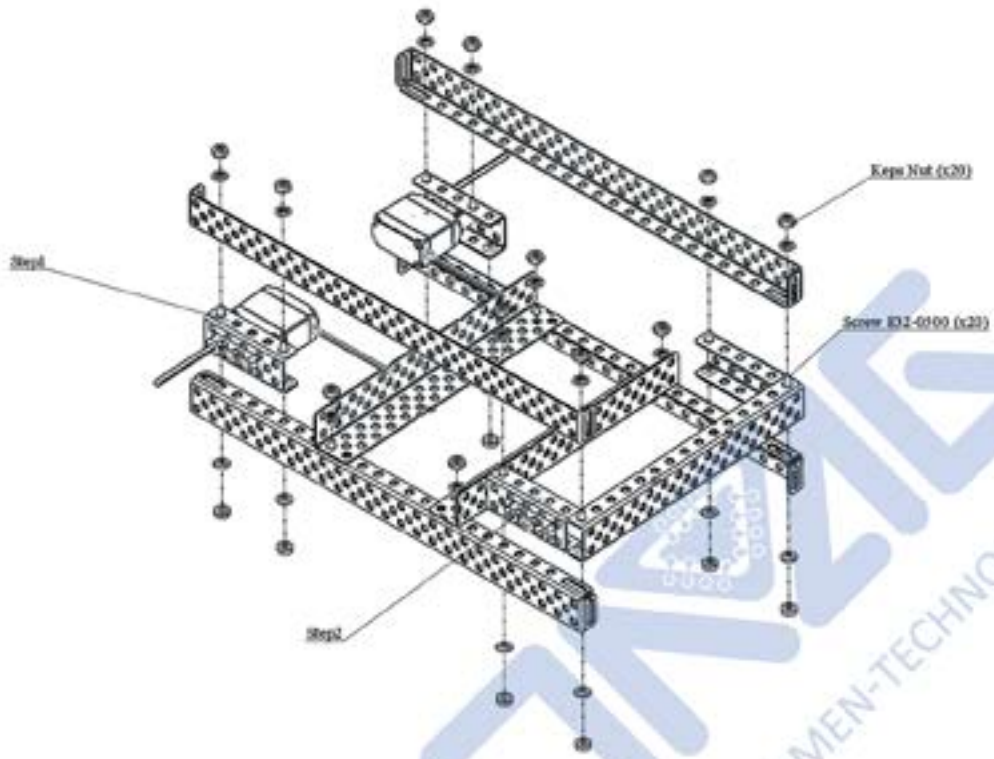


Шаг 2

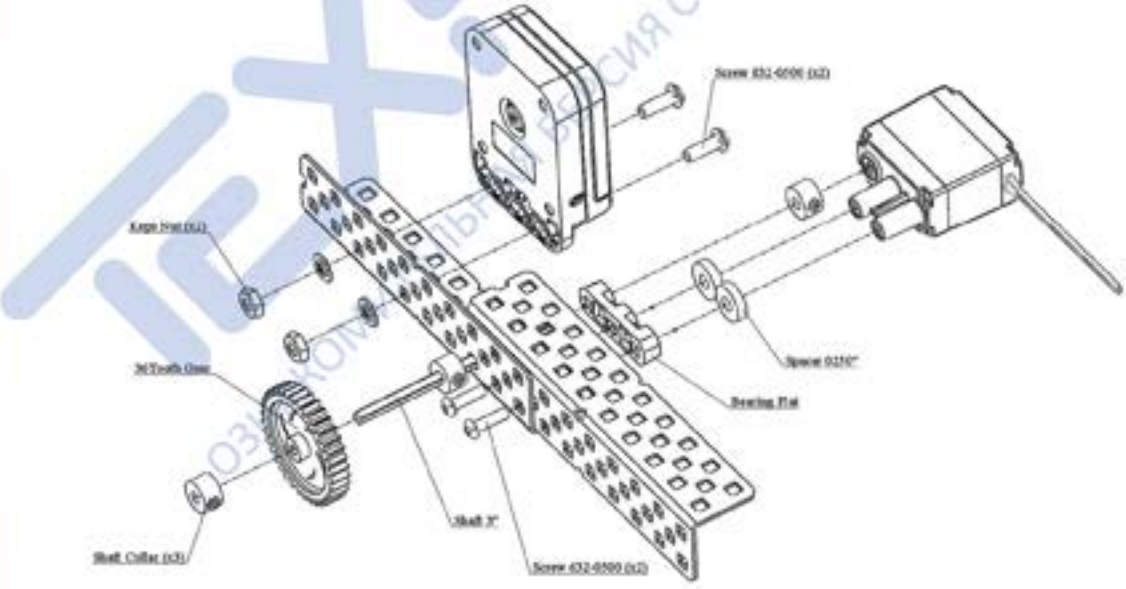




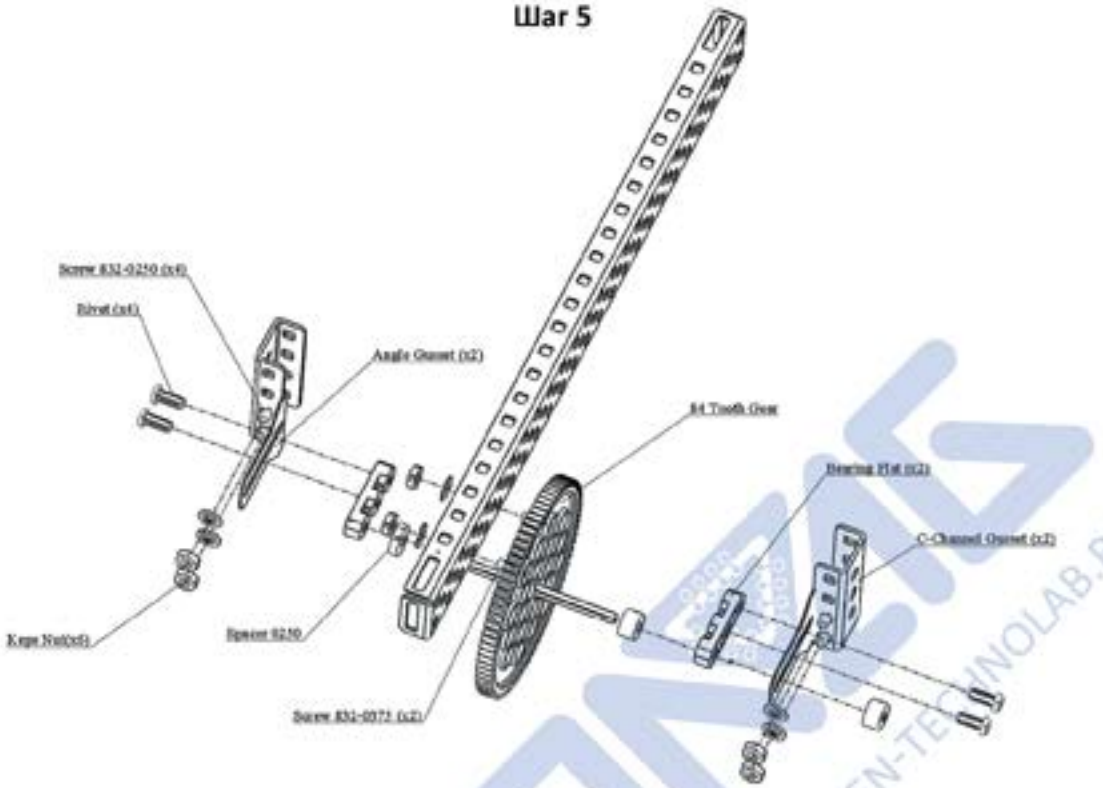
Шаг 3



Шаг 4

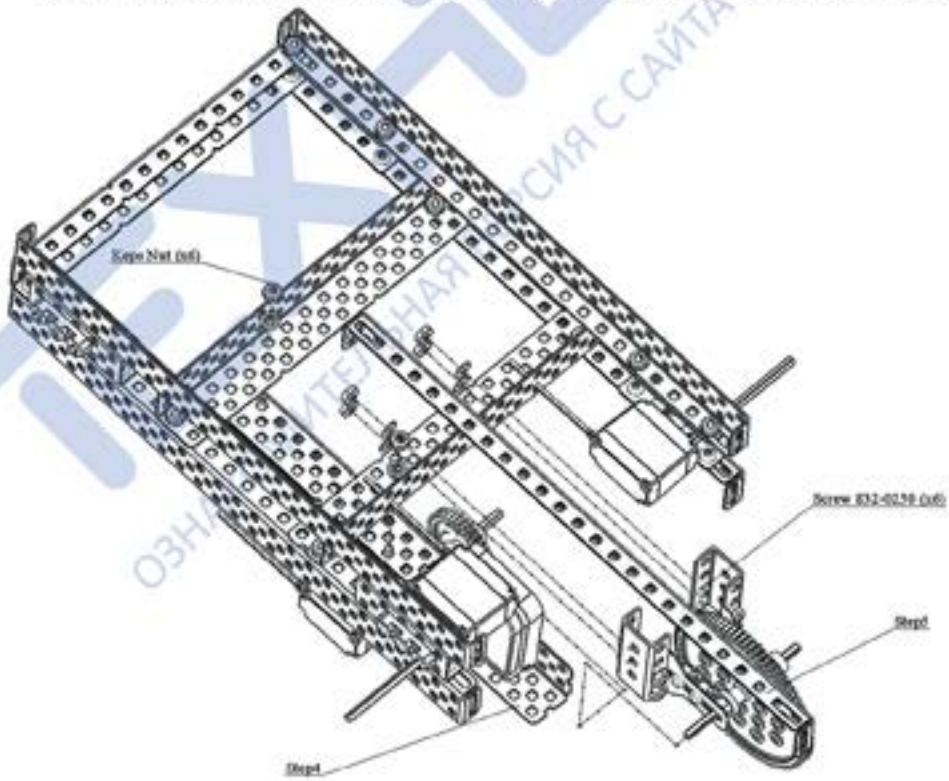


Шаг 5

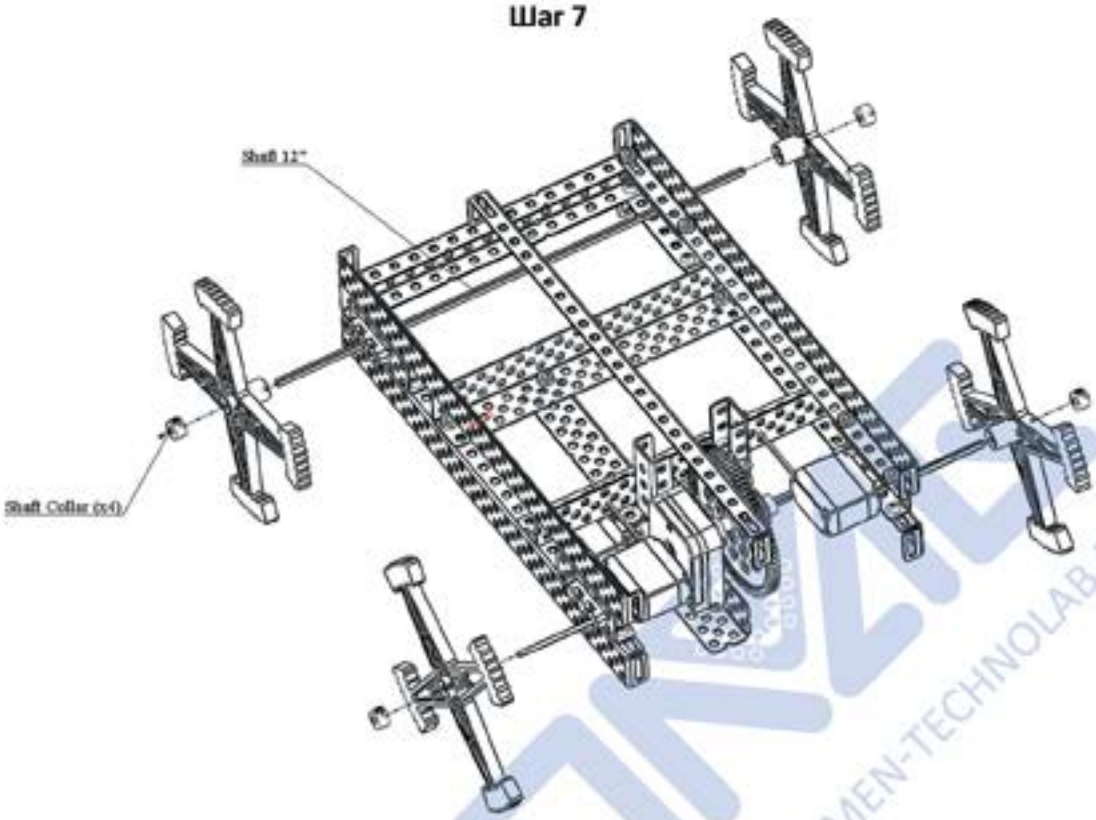


Шаг 6

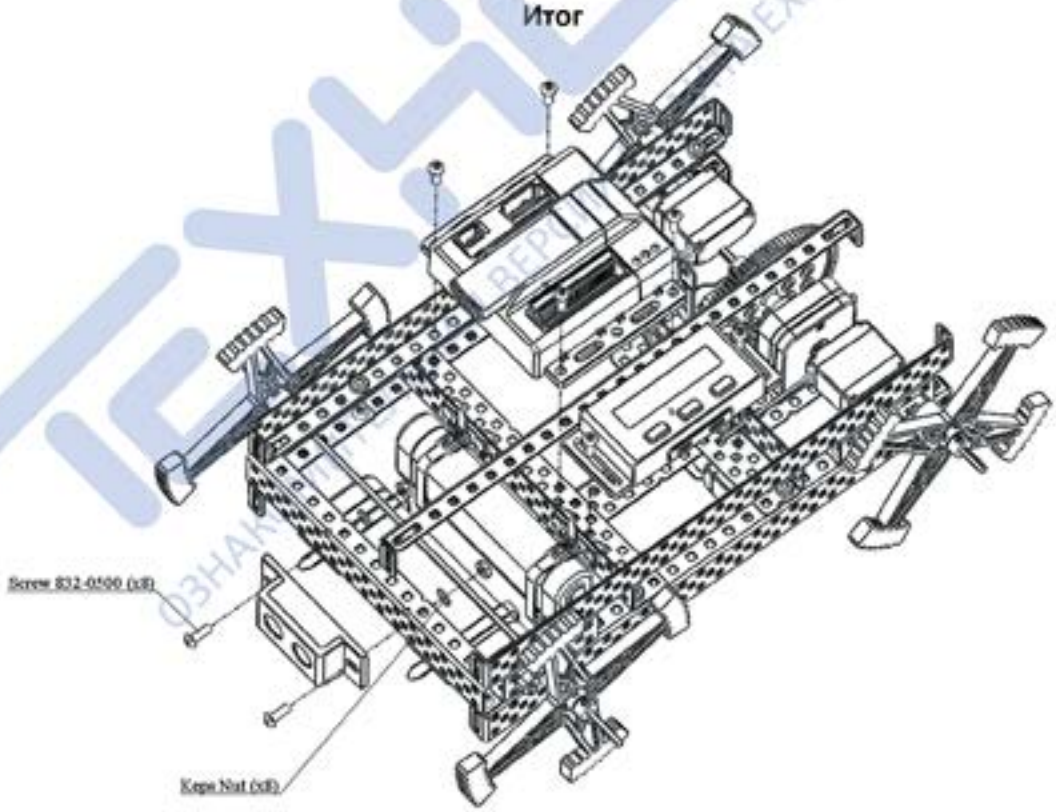
Вал из Шага 5 вставить в соответствующее отверстие в Shaft Encoder.



Шаг 7



Итог

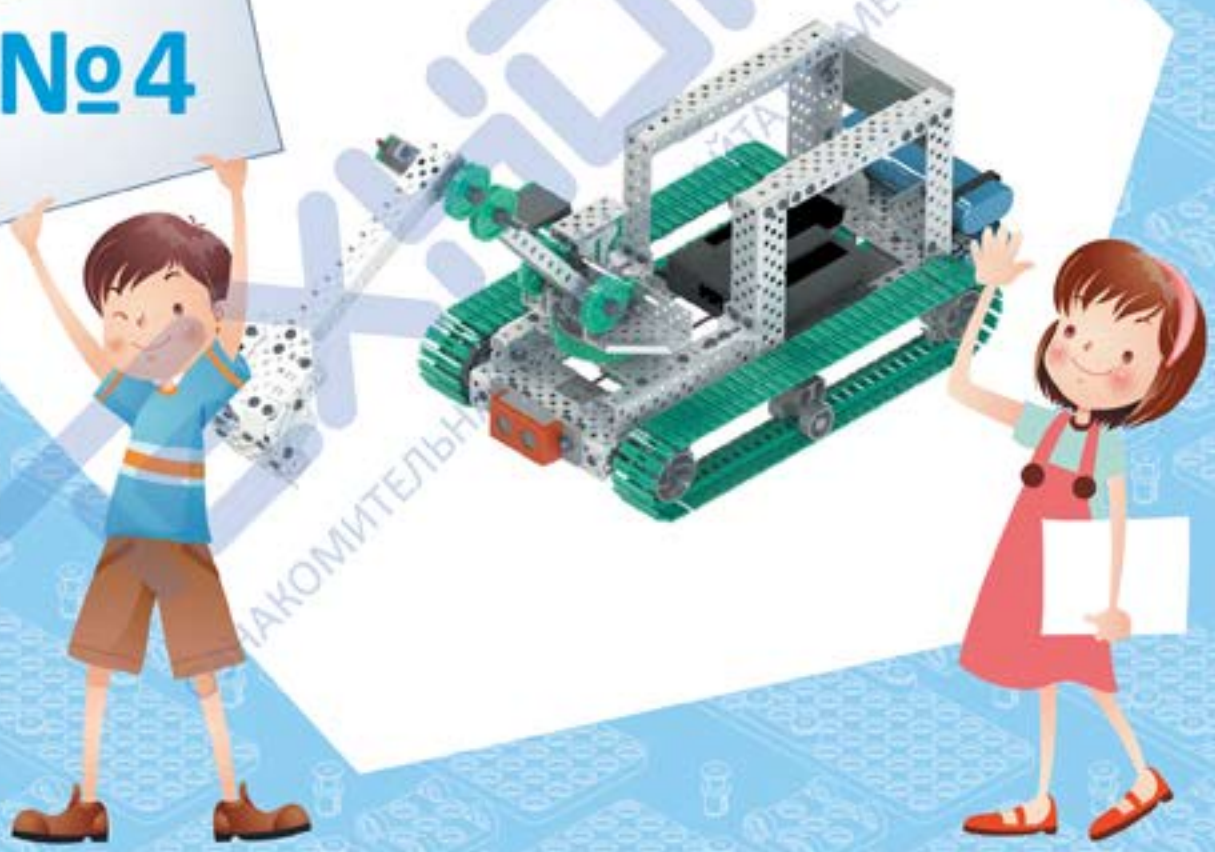


ПРИДОЖЕНИЕ №4 РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА НА БАЗЕ ГУСЕНИЦ

ЭКЗАМЕН ТЕХНОЛАБ

РУКОВОДСТВО ПО СБОРКЕ МОБИЛЬНОГО РОБОТА НА БАЗЕ ГУСЕНИЦ

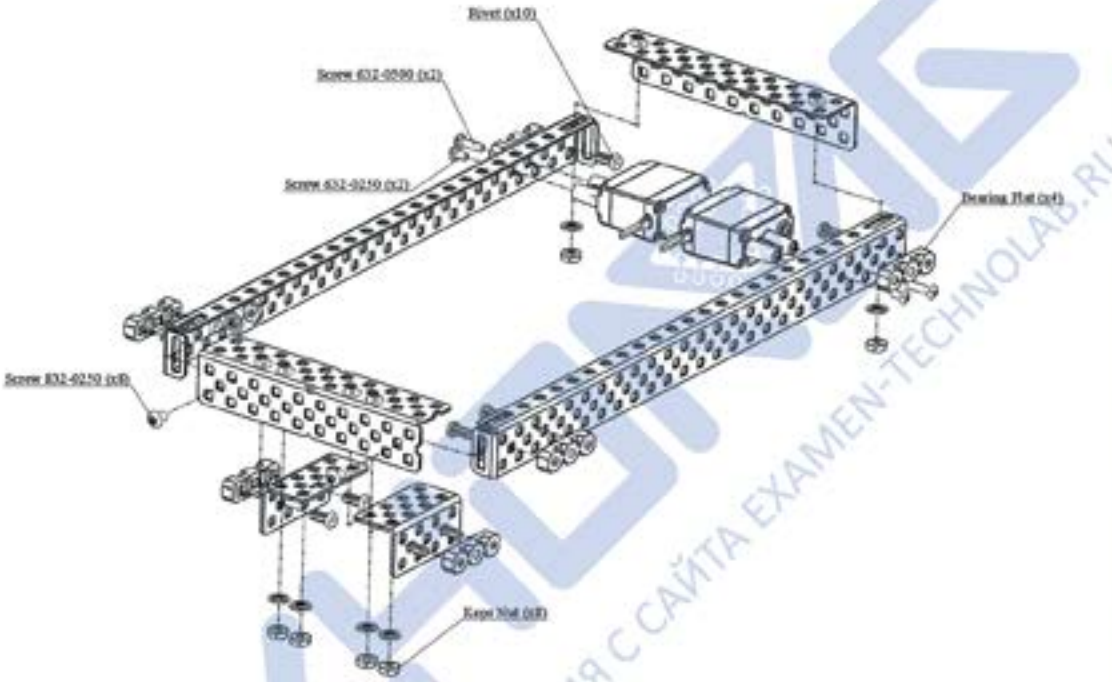
№4



Приложение № 4. Руководство по сборке мобильного робота на базе гусениц

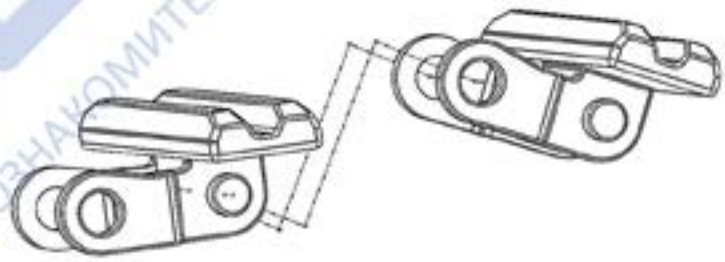
В данном руководстве используются дополнительные детали, не входящие в базовый набор.

Шаг 1

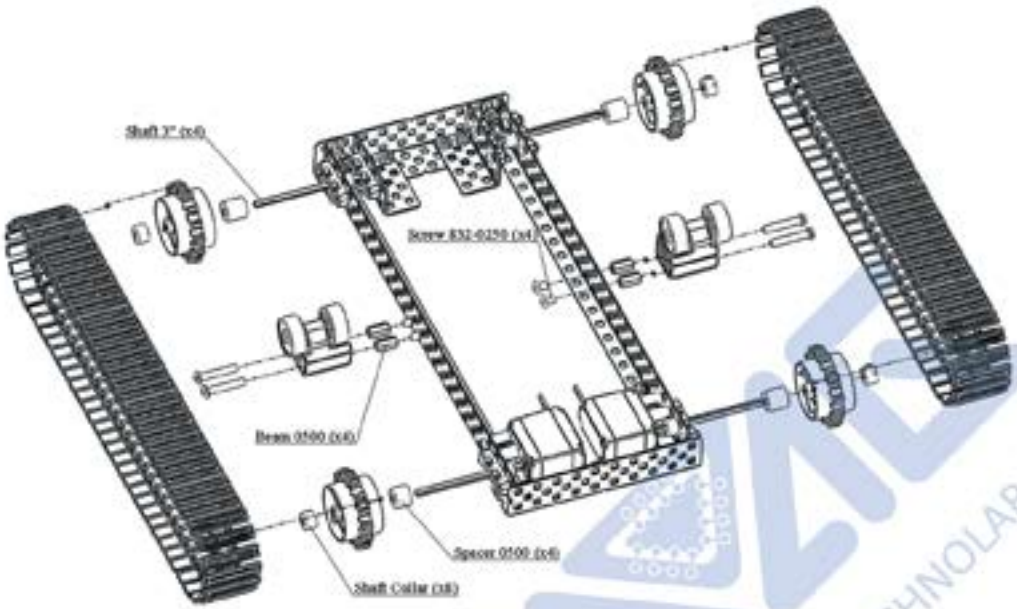


Шаг 2

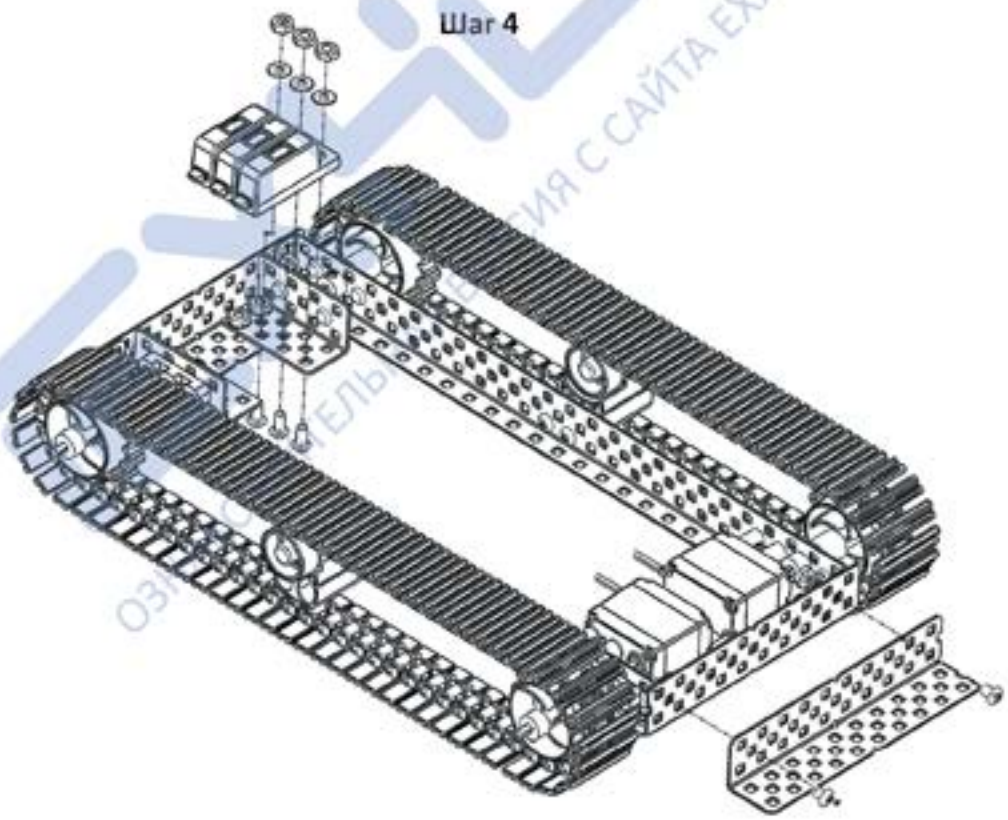
Собрать 2 гусеницы по 71 звену каждая.



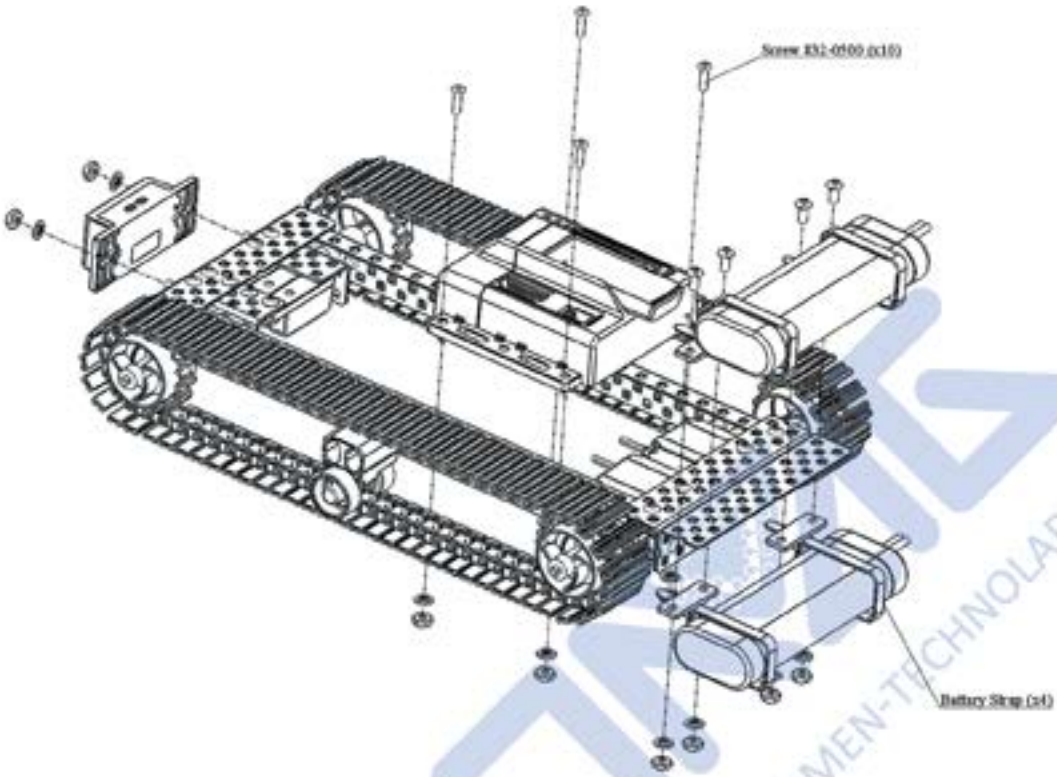
Шаг 3



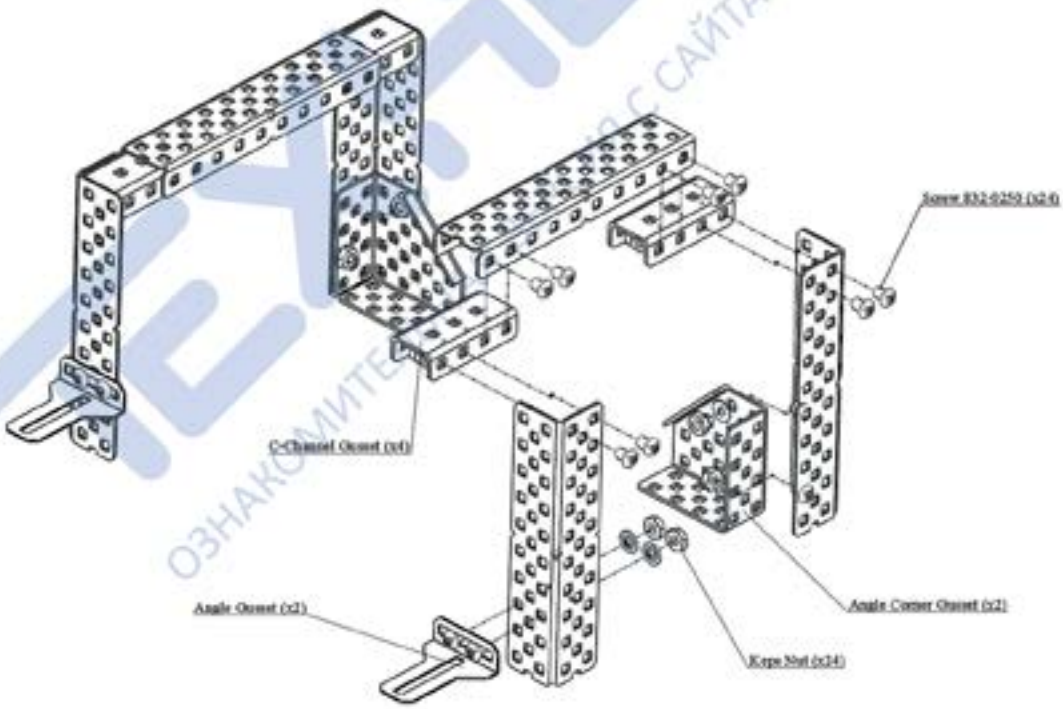
Шаг 4



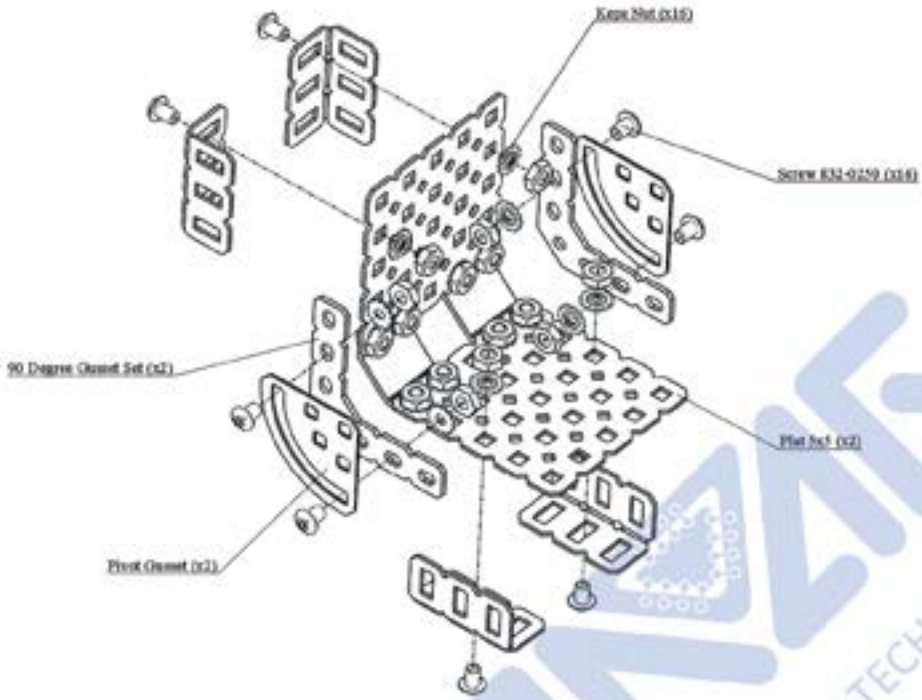
Шаг 5



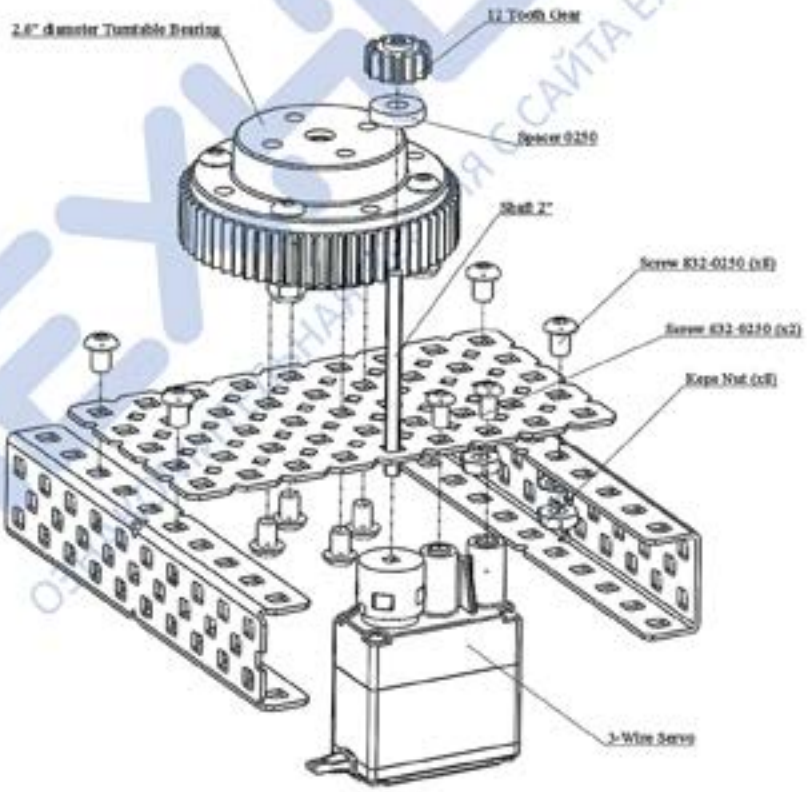
Шаг 6



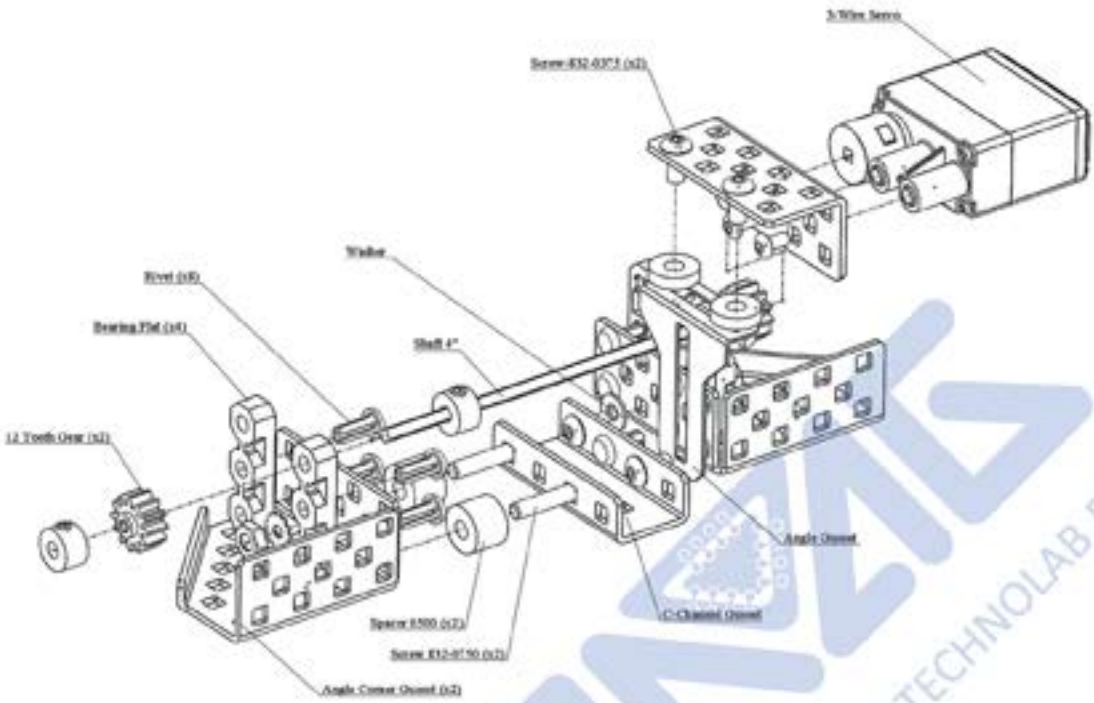
Шаг 7



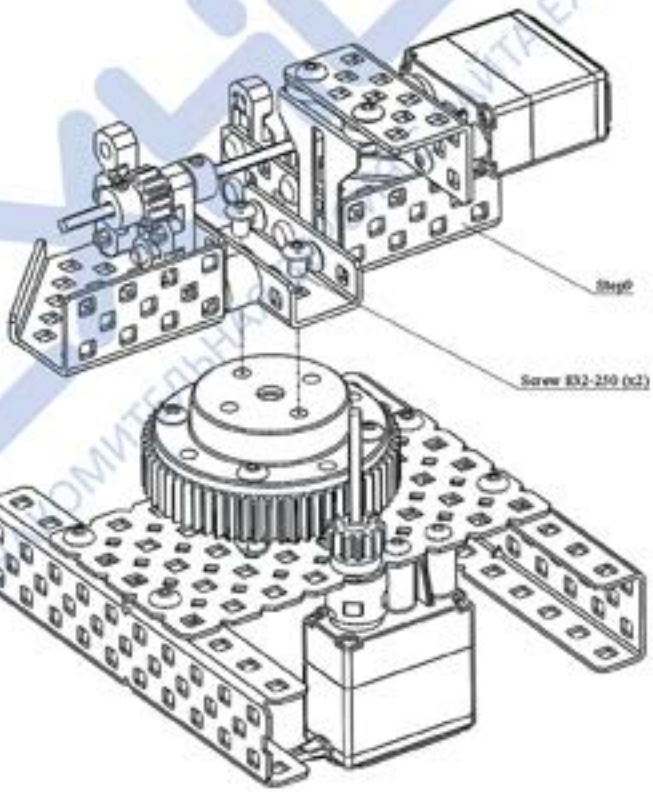
Шаг 8



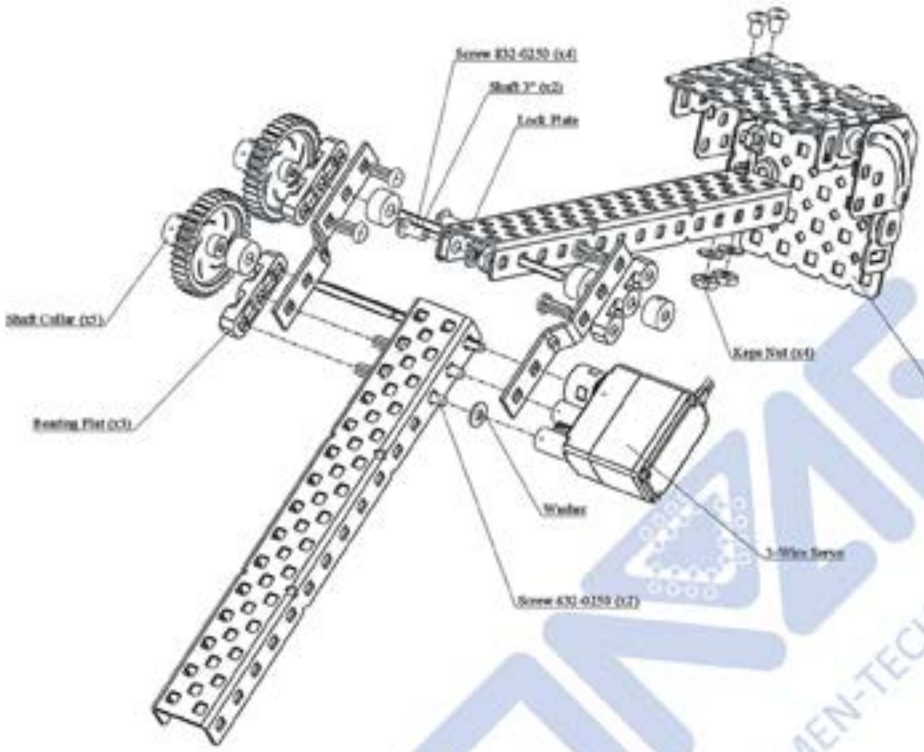
Шаг 9



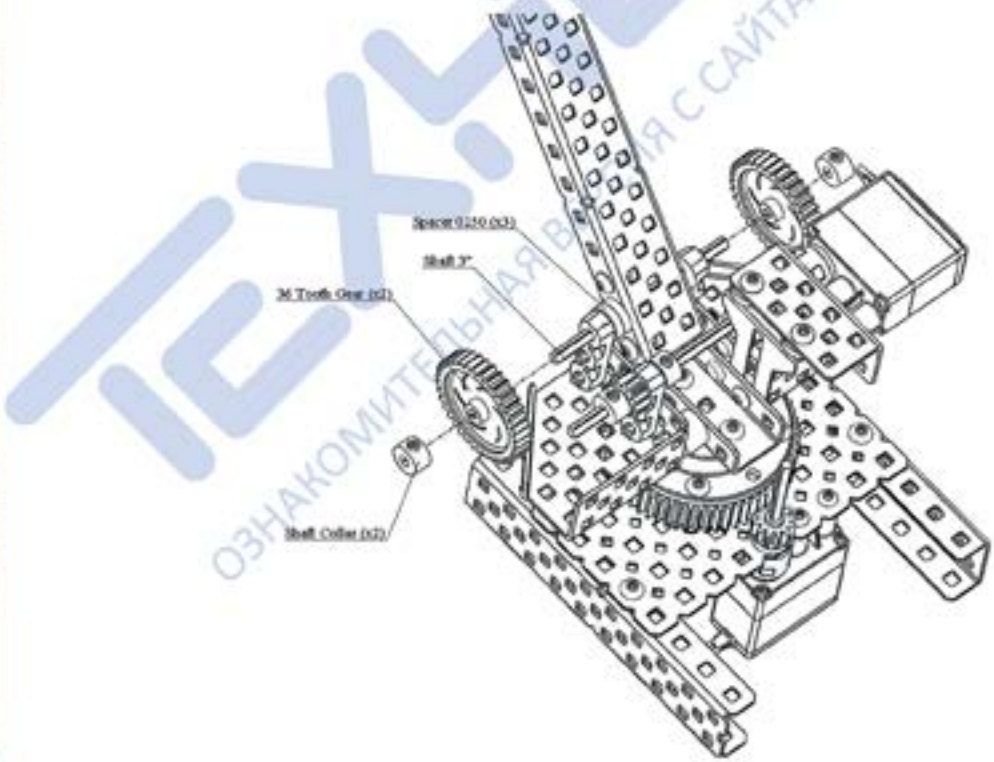
Шаг 10



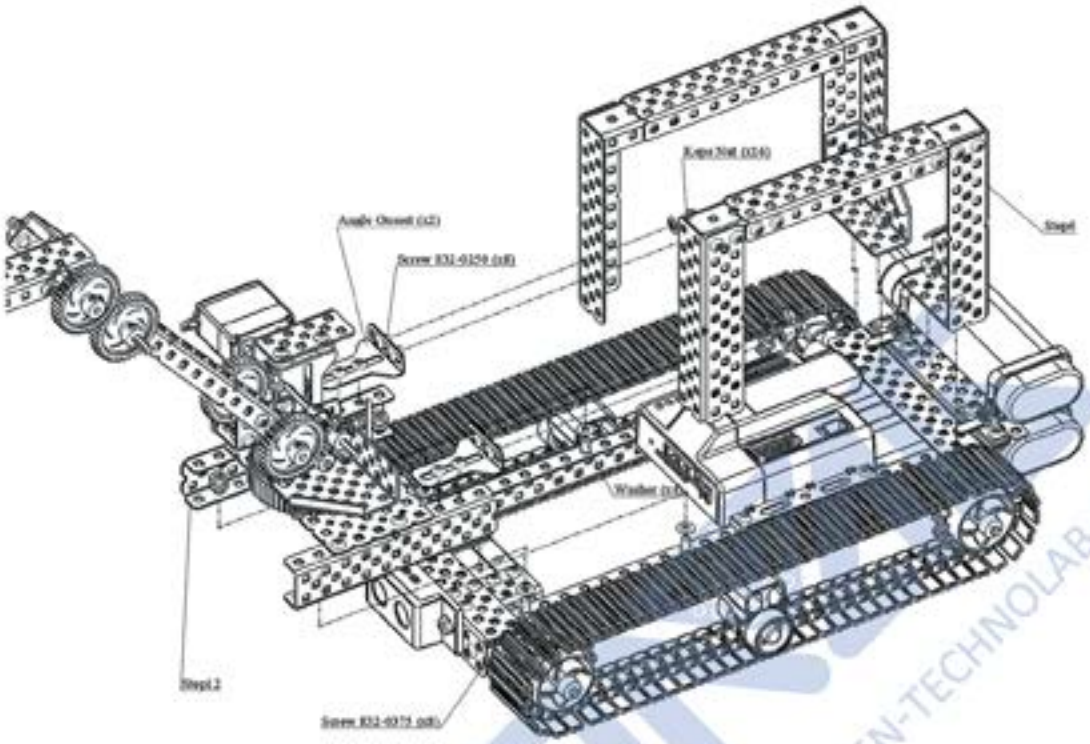
Шаг 11



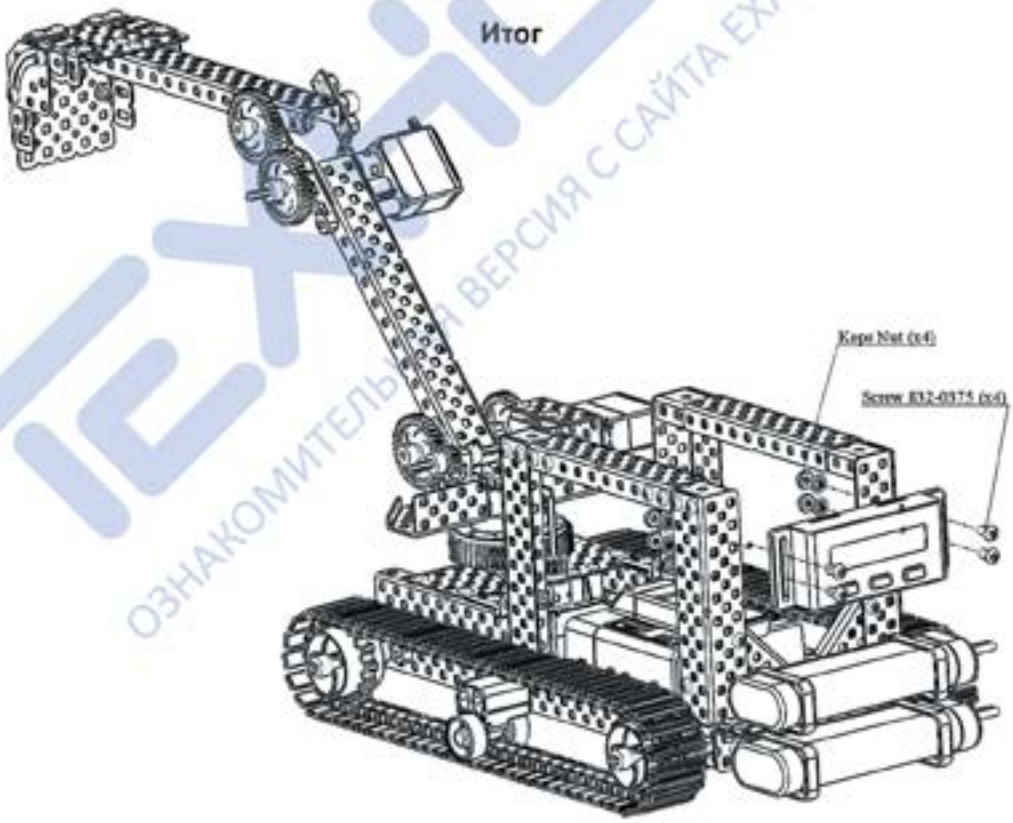
Шаг 12



Шаг 13



Итог



Учебно-методическое издание

Константин Ермишин
Алексей Панфилов
Сергей Косаченко

ОСНОВЫ РОБОТОТЕХНИКИ

Учебно-методическое пособие
к образовательному набору
по робототехнике
«Технолаб»

ОБРАЗОВАТЕЛЬНЫЙ
РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)
12 – 15 ЛЕТ

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

Гигиенический сертификат
№ РОСС RU. АЕ51. Н 16678 от 20.05.2015 г.

Главный редактор *Л. Д. Лаппо*
Корректор Баринская И. Д.
Дизайн обложки
и компьютерная верстка *А. А. Винокуров*

107045, 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./факс +7 (495) 641-00-19 (многоканальный)

ТЕХНОЛАБ

ОБРАЗОВАТЕЛЬНЫЙ РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)

www.examen-technolab.ru

Артикул ТВ-0441-М-2

ISBN 978-5-377-10297-7

12-15 ЛЕТ