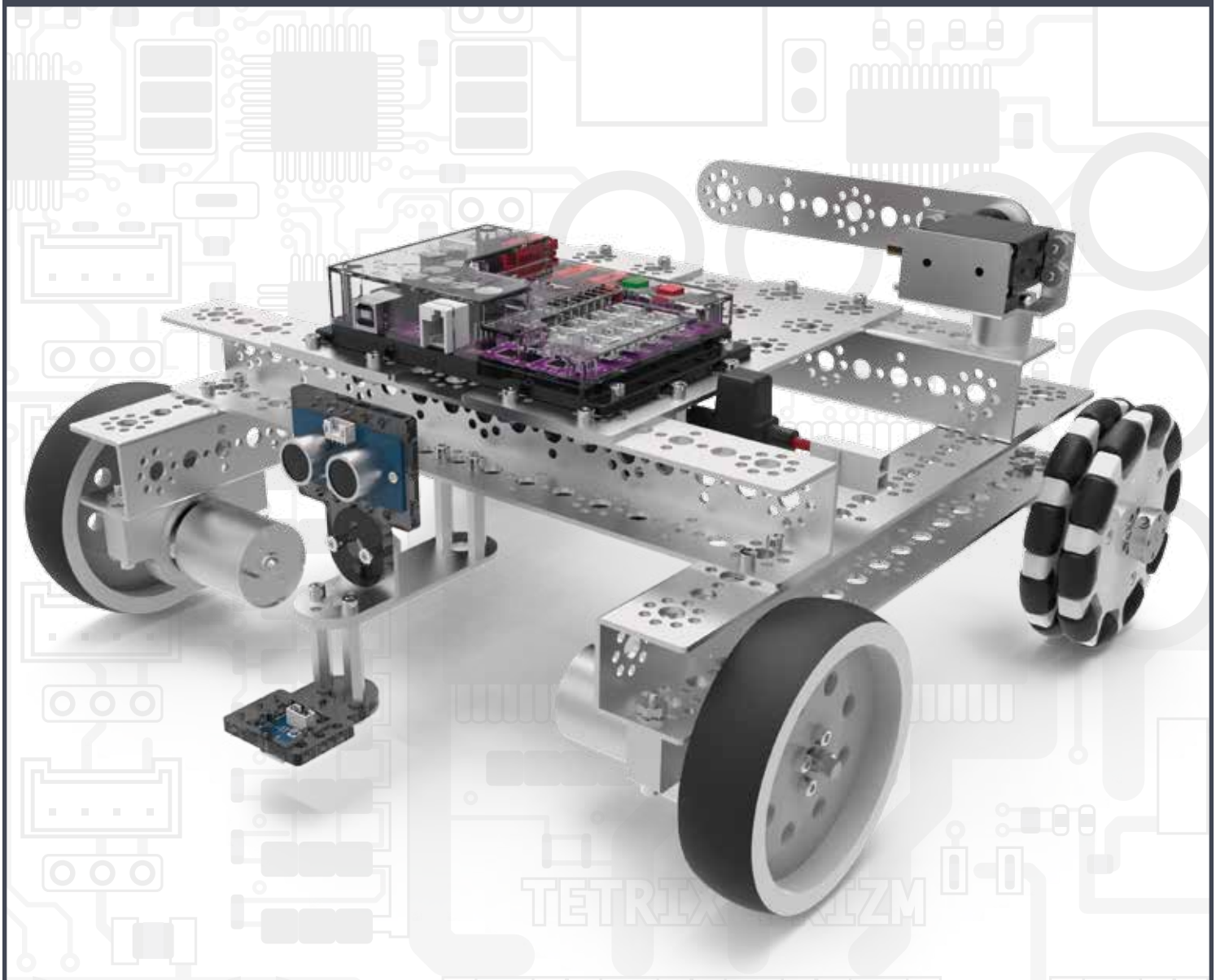
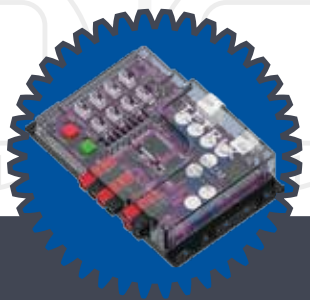
PITSCO

TETRIX® MAX





Контроллер к робототехническим моделям серии TETRIX® PRIZM™ Руководство по программированию

PITSCO EDUCATION

43212

Советники по содержимому: Пол Уттли, Памела Сциферс, Тим Лэнкфорд, Невин Джоунз и Билл Холден.

Создатели моделей и рендеринга*SolidWorks*® *Composer*™ и *KeyShot*®: Лэнкфорд, Брайан Эккелберри и Джейсон Редд. Подготовка публикации с помощью настольных издательских средств: Тодд Макджордж.

V1.0
10/16

Обновления этого руководства в формате PDF см. на TETRIXrobotics.com.

Оглавление

"Добро пожаловать, программисты, проектировщики и конструкторы роботов, инженеры, учащиеся, учителя, воспитатели и наставники!"

Компания Pitsco Education с удовольствием предлагает вашему вниманию *Руководство по программированию контроллера к робототехническим моделям серии TETRIX® PRIZM™*, которое содержит ряд увлекательных и постепенно усложняющихся упражнений для обучения основам программирования моделей TETRIX MAX, создаваемых с использованием контроллера PRIZM и *ПО Arduino (IDE)*.

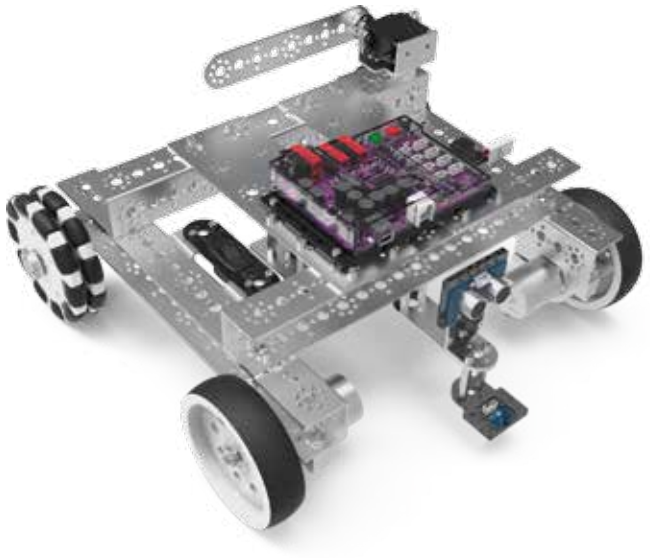
Руководство по программированию представляет собой ценное пособие, объясняющее учащимся и преподавателям, как с помощью контроллера PRIZM и конструктора серии TETRIX MAX собирать и программировать умные, точные робототехнические модели, ничем не отличающиеся от настоящих роботов. В Руководстве пять начальных упражнений, пошаговые инструкции по сборке робота-исполнителя, 10 полноценных уроков по применению робота-исполнителя, а также дополнительные упражнения по творческому изменению начального программного кода, что превращает руководство в великолепное пособие по изучению возможностей контроллера PRIZM, деталей конструктора TETRIX и *ПО Arduino (IDE)*, которое даёт учащимся отличный фундамент для последующего развития.

Это великолепное решение для введения преподавателей и учащихся в робототехнику, объединяющее в себе уже готовый к применению контроллер PRIZM с интуитивно понятным конструктором серии MAX и удобной текстовой средой программирования. Постепенное и последовательное усложнение упражнений помогает учащимся быстро и легко перейти к конструированию действующих робототехнических моделей, немедленно пожать плоды своих усилий и уделять на уроке больше времени решению задач и применению полученных научно-технических знаний.

К тому же PRIZM не просто великолепный инструмент для преподавания программирования. С его помощью можно вдохнуть жизнь в уроки о датчиках, мощности, энергии, передаточных отношениях и прочем. Даже прозрачная поликарбонатная оболочка контроллера имеет учебное назначение, позволяя пользователям видеть, как он устроен внутри.

Мы также включили кое-какие дополнительные задания по научно-техническим дисциплинам (затрагивающие понятия, не охваченные в этом руководстве), которые при желании можно проходить на каждом уроке. Эти дополнительные задания можно включить в учебный план, если вы знаете содержание этих понятий или можете привлечь к их преподаванию других учителей.

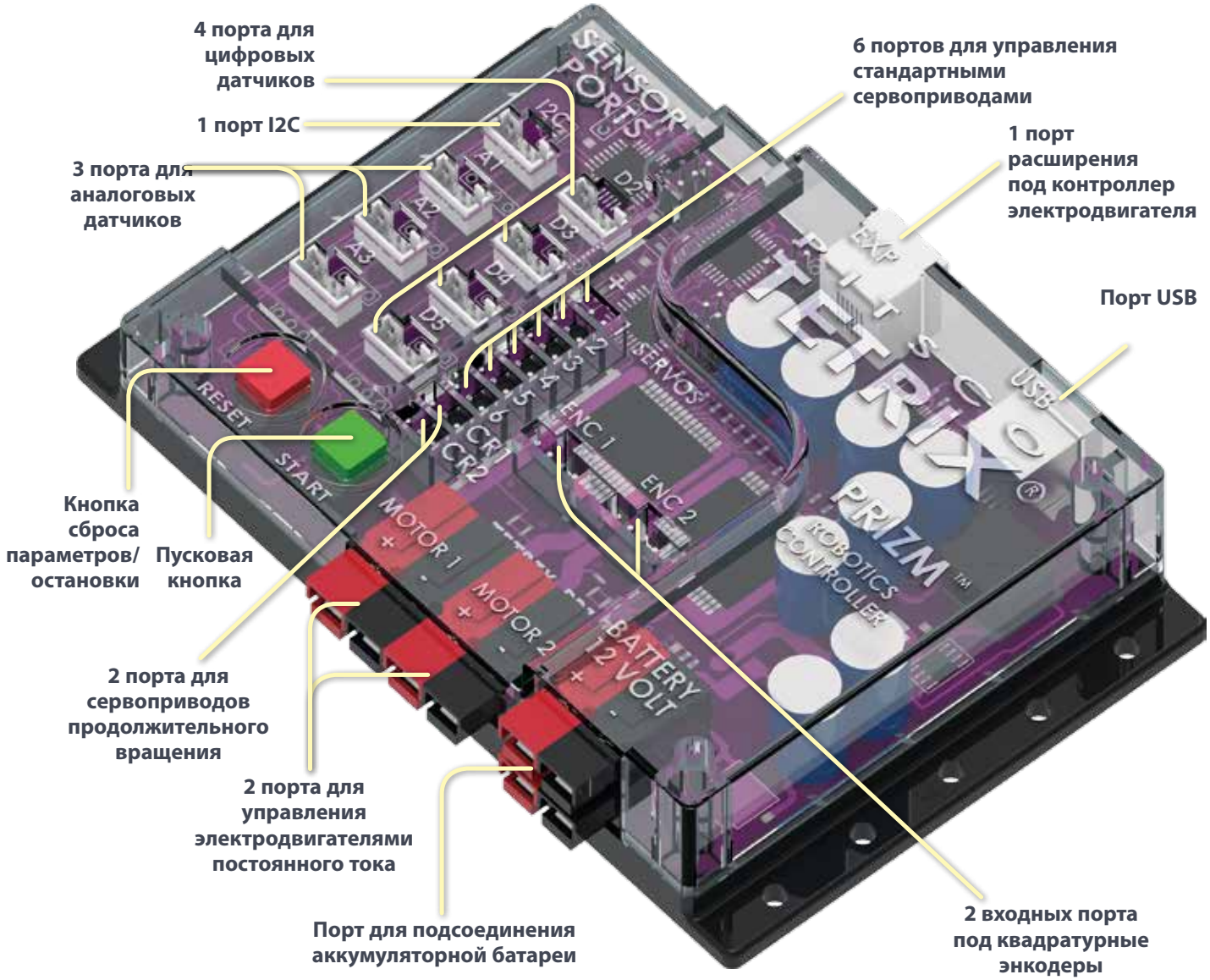
Надеемся, что это руководство станет отличной отправной точкой в учёбе с применением контроллера PRIZM. С нетерпением ждём результатов в виде новаторских проектов и моделей роботов.



Пусть кодирование, конструирование, решение задач, проектирование и сотрудничество доставят вам удовольствие!

Обзор технической части контроллера PRIZM

Контроллер к робототехническим моделям серии PRIZM:
Программируемое устройство, которое управляет робототехнической моделью серии TETRIX MAX.



Исчерпывающие и подробные технические характеристики см. в разделе "Технические характеристики контроллера к робототехническим моделям серии TETRIX® PRIZM™" в приложении на с. 124.

Датчик

Устройство, которое обнаруживает изменения в окружающей среде и реагирует на них

Ультразвуковой датчик
Позволяет робототехнической модели измерять расстояние до некоего объекта и реагировать на движение

Датчик линии
Позволяет робототехнической модели двигаться вдоль чёрной линии на белом фоне или наоборот

Электродвигатель

Устройство, которое производит движение или энергию для совершения работы

Электродвигатель постоянного тока
Электродвигатель постоянного тока работает от напряжения 12 В, вращается с частотой 152 об/мин и развивает крутящий момент 2,26 Нм

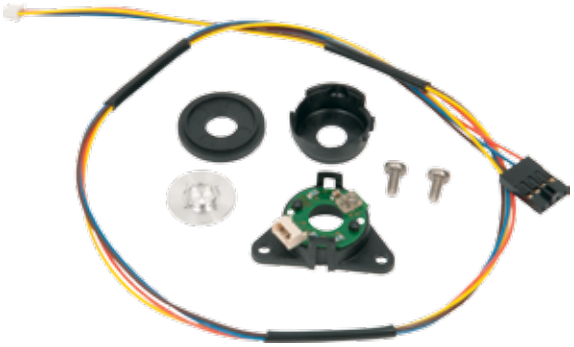
Кабель к электродвигателю, снабжённый разъёмами Powerpole
Для подключения электродвигателя постоянного тока к источнику питания

Стандартный сервопривод
Позволяет выбрать точное положение в пределах угла перемещения, равного 180 градусам

Сервопривод продолжительного вращения
Работает в режиме продолжительного вращения, как по часовой стрелке, так и против часовой стрелки

Энкодер (датчик углового положения вала):

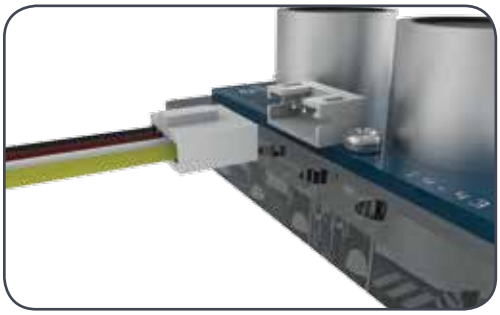
Датчик, преобразующий движение в электрический сигнал, считывая который контроллер PRIZM определяет угловое положение, число оборотов, угловую скорость или направление вращения электродвигателей постоянного тока TETRIX

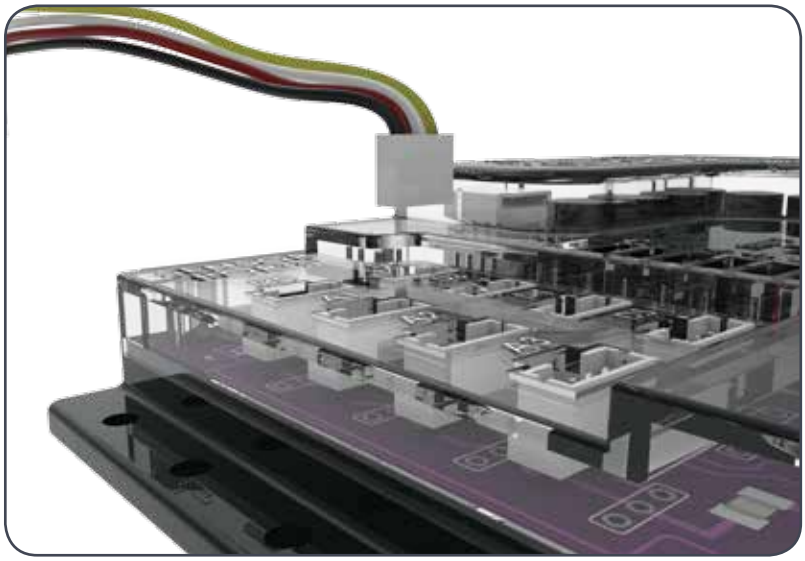


Энкодер для электродвигателя:
Позволяет программировать поведение электродвигателей, используя обороты и градусы.
Обратите внимание! Комплект энкодера для электродвигателя TETRIX не входит в робототехнический набор и покупается отдельно.

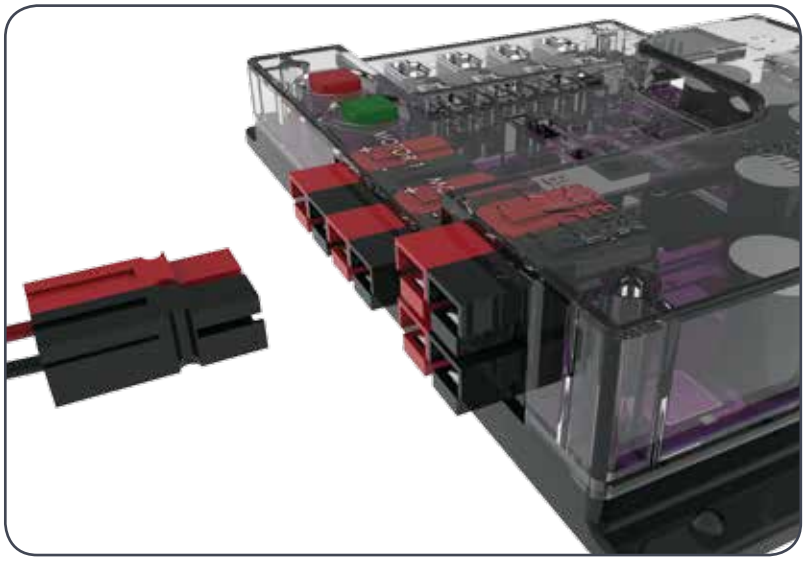
Настройка контроллера PRIZM™

Подсоединение датчиков
Для подсоединения датчика к контроллеру PRIZM вставьте в датчик один конец соединительного кабеля. Другой конец вставьте в порты с маркировкой D2-D5 или A1-A3.

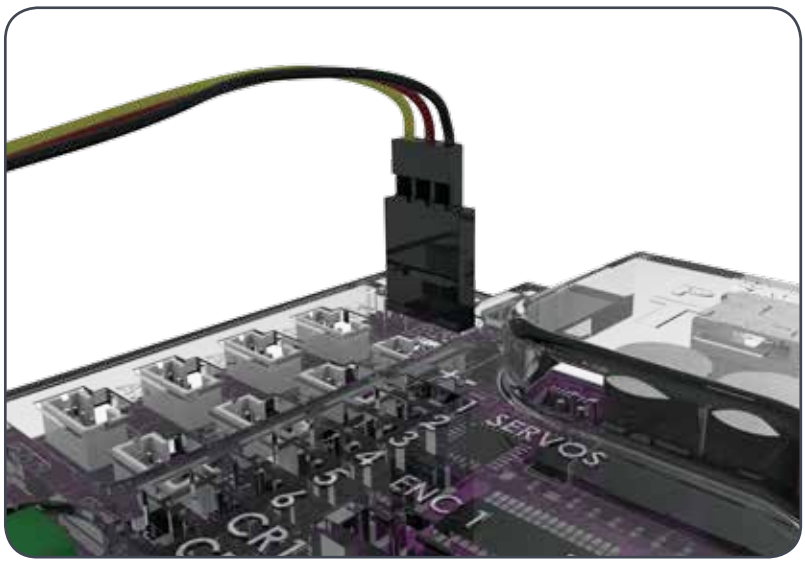




Подсоединение электродвигателей постоянного тока
Для подсоединения электродвигателя постоянного тока к контроллеру PRIZM подсоедините белый разъём Powerpole к электродвигателю, а парные цветные разъёмы — к одному из двух портов для электродвигателей. Для правильной работы электродвигателя подсоедините красный разъём соединительного кабеля к красному порту, а чёрный разъём — к чёрному порту.



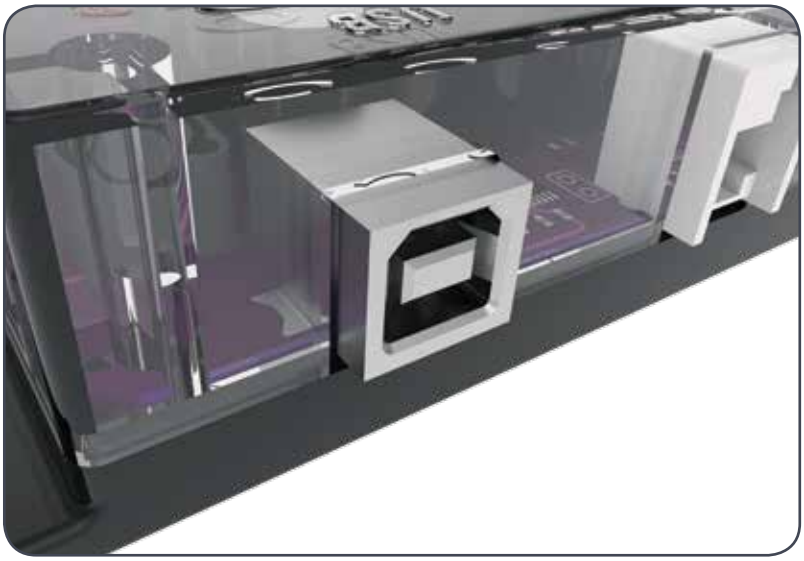
Подсоединение сервоприводов
Для подсоединения стандартного сервопривода к контроллеру PRIZM вставьте один конец провода сервопривода в один из портов (1-6) для сервоприводов. Для подсоединения сервопривода продолжительного вращения к контроллеру PRIZM вставьте один конец провода сервопривода в один из портов для сервоприводов с маркировкой CR1 и CR2.



Скачивание и подгружение данных
Порт USB на контроллере PRIZM служит для связи между самим контроллером и устройством на платформе Windows, Mac или Linux.

Порт позволяет пользователям скачивать и подгружать данные с компьютера в контроллер PRIZM.

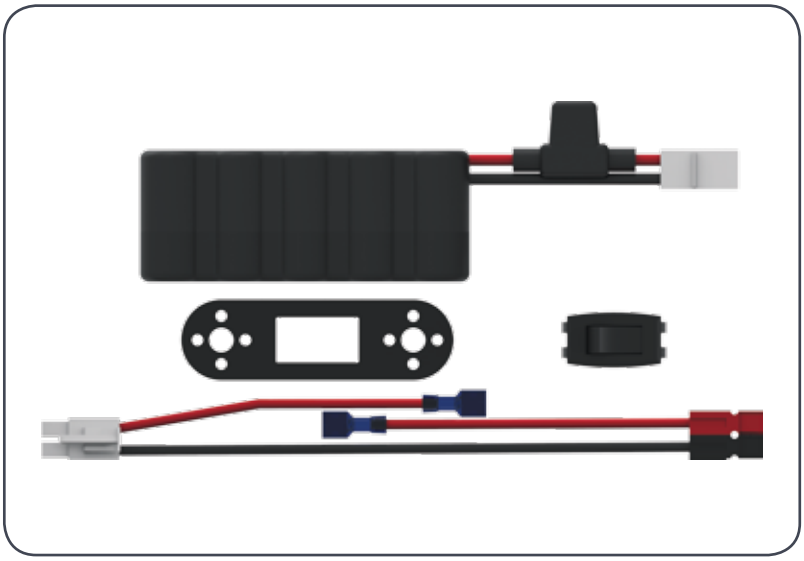
Чтобы подгрузить какую-нибудь программу в PRIZM, вставьте разъём B кабеля USB в порт USB на контроллере, а разъём A — в свой компьютер.



Электропитание для контроллера PRIZM
Для питания контроллера PRIZM служит перезаряжаемая никель-металлогидридная аккумуляторная батарея на 12 В серии TETRIX.

К контроллеру PRIZM прилагается выключатель, предназначенный для подсоединения аккумуляторной батареи к соответствующему порту, выполненному в стилистике PRIZM и снабжённому разъёмами Powerpole.

Указания по сборке выключателя и соединению его с аккумуляторной батареей см. в упражнении "Сборка робота-исполнителя" в этом руководстве.



Предупреждение! Запрещается использовать с контроллером PRIZM аккумуляторные батареи сторонних изготовителей. Аккумуляторные батареи TETRIX снабжены плавкими предохранителями, допущенными к применению с данной системой. Повреждение изделия вследствие нарушения запрета ведёт к отмене гарантии.

Подсоединение аккумуляторной батареи к контроллеру PRIZM
Для подсоединения аккумуляторной батареи к PRIZM вставьте цветные соединители аккумуляторной батареи в один из двух портов на контроллере, предназначенных для аккумуляторной батареи. Для правильной работы электродвигателя подсоедините красный разъём соединительного кабеля к красному порту, а чёрный разъём — к чёрному порту. Нельзя соединять разъёмы с разной полярностью.

Аккумуляторную батарею можно подсоединить либо к верхнему, либо к нижнему ряду разъёмов.

Предупреждение! Запрещается подсоединять к одному контроллеру PRIZM две аккумуляторные батареи. Попытка сделать это приведёт к повреждению и отказу изделия. Используйте для питания электроцепи контроллера PRIZM и любых контроллеров расширения, соединённых с ним в гирляндную цепь, только одну аккумуляторную батарею.

Обзор программного обеспечения

Единая среда разработки (IDE) Arduino представляет собой программное обеспечение с открытыми исходными кодами, позволяющее составлять управляющие программы и отправлять их в исполнительные устройства. Удобный для разработчиков инструмент создания программ в виде текста, *ПО Arduino (IDE)* можно использовать на различных устройствах под управлением операционных систем Windows, Macintosh и Linux. Благодаря этому *ПО Arduino (IDE)* универсально в применении, удобно для работы в классе, годится для выполнения конкурсных заданий, и позволяет любому пользователю, даже неискушенному любителю, начав с простого, достичь высокого уровня программирования. Все указанное делает его в равной мере идеальным как для новичка, так и для многоопытного программиста.

Для диалога с PRIZM в *ПО Arduino (IDE)* используется язык программирования на основе языка Си. В *ПО Arduino (IDE)* отдельная программа называется *скетчем*. В ходе выполнения каждого упражнения в этом руководстве предполагается создание скетча, который отдаёт команды роботу.

В рамках этого руководства мы сосредоточимся на основах использования *ПО Arduino (IDE)* применительно к контроллеру PRIZM. Практический разбор примеров и наглядное применение комманд для небольшой модели робота-исполнителя, собранной из робототехнического конструктора серии TETRIX MAX, покажет вам, насколько легко использовать Arduino с PRIZM.

Примечание: Руководство нельзя рассматривать в качестве исчерпывающего учебного пособия по программированию на языке Arduino, в основе которого лежит язык Си. В интернете множество отличных ресурсов для тех, кто хочет узнать о навыках программирования больше. Если вам интересны такие ресурсы, полезно будет начать с сайта Arduino: **www.arduino.cc**.

Настройка программного обеспечения

Первым делом необходимо установить *программное обеспечение (ПО) Arduino (в виде единой среды разработки — IDE)*. На сайте Arduino (**www.arduino.cc**) есть ПО для операционных систем Windows, Macintosh, и Linux. Зайдя на главную страницу сайта Arduino, щёлкните по закладке Download. На странице Download выберите установочный файл (дистрибутив) для своей операционной системы и следуйте дополнительным указаниям.

Установка библиотеки Arduino для контроллера PRIZM

Добавляя специальные библиотеки можно расширить сферу использования *ПО Arduino (IDE)*. Библиотеки представляют собой набор программ, с помощью которых легче создать некую программу или, как называют это в Arduino, — скетч. После успешной установки *ПО Arduino (IDE)* можно добавить библиотеку Arduino для контроллера PRIZM. В библиотеке для контроллера PRIZM хранятся особые функции, написанные для контроллера TETRIX PRIZM. В ходе занятий мы подробно рассмотрим не все, но многие из этих функций.

Тем, кто захочет подробнее изучить все функции, имеющиеся в библиотеке для контроллера PRIZM, предлагаем заглянуть в раздел приложения, озаглавленный "Библиотечные функции на языке Arduino для контроллеров серии TETRIX PRIZM".

Пользователям библиотека PRIZM поставляется в виде файла .zip: TETRIX_PRIZM.zip. Для начала необходимо скачать библиотеку для контроллера PRIZM с сайта TETRIX. Библиотеку вы найдёте на **www.TETRIXrobotics.com/PRIZMdownloads**.

Библиотеку для контроллера PRIZM после скачивания можно установить в *ПО Arduino (IDE)* двумя способами.

Импортирование библиотеки в виде файла .zip

Сделать это в *ПО Arduino (IDE)* можно с помощью команды Add .ZIP Library [Добавить библиотеку .ZIP] в меню.

Находясь в *ПО Arduino (IDE)*, последовательно пройдите по пунктам **Sketch > Include Library**. В выпадающем меню выберите команду **Add .ZIP Library** [Добавить библиотеку .ZIP] (рисунок 1).

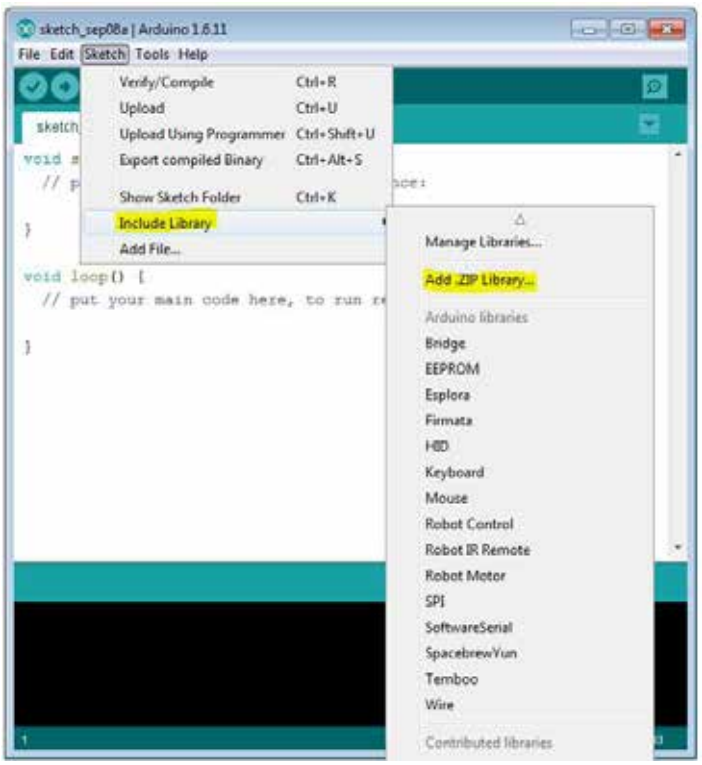


Рисунок 1

Примечание: Все указания и снимки экрана, используемые в данном руководстве, имеют отношение к версии 1.6.11 *ПО Arduino (IDE)*. Возможны незначительные различия в указаниях и изображениях, относящихся к разным операционным системам и версиям ПО.

Примечания для учителя:

- При наличии в классной комнате соответствующих аппаратных и программных средств обдумайте, как скачать и установить на компьютеры, которыми вы с учащимися будете пользоваться, *ПО Arduino (IDE)* и библиотеку Arduino для контроллера TETRIX PRIZM.
- Учителям рекомендуется разбить класс на группы по 2-4 человека и выполнить каждое задание самим, прежде чем давать его учащимся. Тогда вы хорошо прочувствуете возможные вопросы учащихся и будете знать ответы на них.

Подсказка: На рисунках в этом разделе представлен вид экрана, типичный для Windows. Вид и место нахождения файлов в операционных системах Mac и Linux могут не совпадать.

Вам будет предложено выбрать ту библиотеку, которую вы хотели бы добавить. Перейдите в то место, куда сохранили библиотеку TETRIX_PRIZM.zip, выберите его и откройте (рисунок 2).

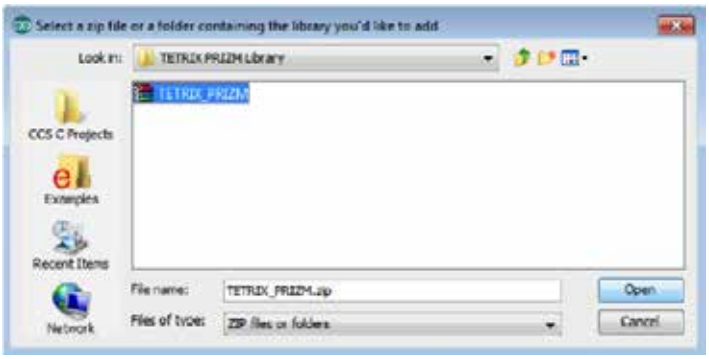


Рисунок 2

Вернитесь в меню **Sketch > Include Library**. Теперь внизу выпадающего меню должна быть видна библиотека. Теперь её можно использовать в наших скетчах; но образцы скетчей для библиотеки появятся в меню **File > Examples** только после перезапуска *ПО Arduino (IDE)*.

Установка вручную

Чтобы установить библиотеку для контроллера PRIZM вручную, сначала закройте *ПО Arduino (IDE)*. Затем извлеките содержимое из файла TETRIX_PRIZM.zip с библиотекой. Перенесите или скопируйте появившуюся папку TETRIX_PRIZM в папку с библиотеками на языке Arduino.

Если вы пользуетесь Windows, последовательность, скорее всего, будет такой: **Documents\Arduino\libraries**.

Если вы пользуетесь Mac, последовательность, скорее всего, будет такой: **Documents\Arduino\libraries**.

Если вы пользуетесь Linux, папка **libraries** будет находится в книге скетчей на языке Arduino.

Перезапустите *ПО Arduino (IDE)*. Убедитесь, что библиотека TETRIX_PRIZM появилась в меню **Sketch > Include Library**.

Кроме того, теперь в выпадающем меню **File > Examples > TETRIX_PRIZM** [Файл > Примеры > TETRIX_PRIZM] должно появиться несколько примеров скетчей для контроллера PRIZM.

Вот и всё! Мы благополучно установили библиотеку Arduino для контроллера PRIZM.

Настройка соединения по USB

Контроллер PRIZM и *ПО Arduino (IDE)* будут поддерживать связь между собой через порт USB компьютера.

Поэтому перед программированием надо убедиться, что контроллер PRIZM правильно настроен в ПО Arduino (IDE) для работы по USB.

Легче всего сделать это так: сначала запустить *ПО Arduino (IDE),* пройти в меню **Tools > Board** и выбрать пункт **Arduino/Genuino Uno** (рисунок 3). И в контроллере PRIZM, и в оригинальной Arduino UNO используется одинаковый микроконтроллер, поэтому выбрать необходимо именно эту плату.

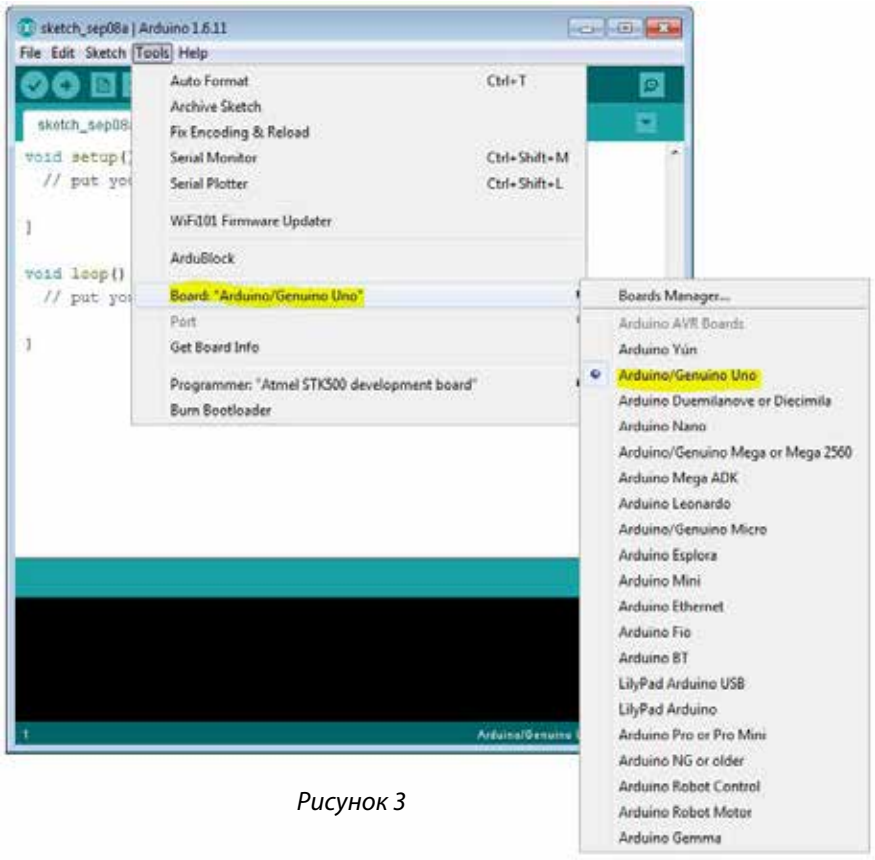


Рисунок 3

Далее, **не подсоединяя контроллер PRIZM**, пройдите в меню **Tools > Port** (рисунок 4) и проверьте, какие есть соединения. Если не обнаружится никаких соединений, слово *Port* будет серым. Если соединения обнаружатся, обратите внимание на перечисляемые порты COM.

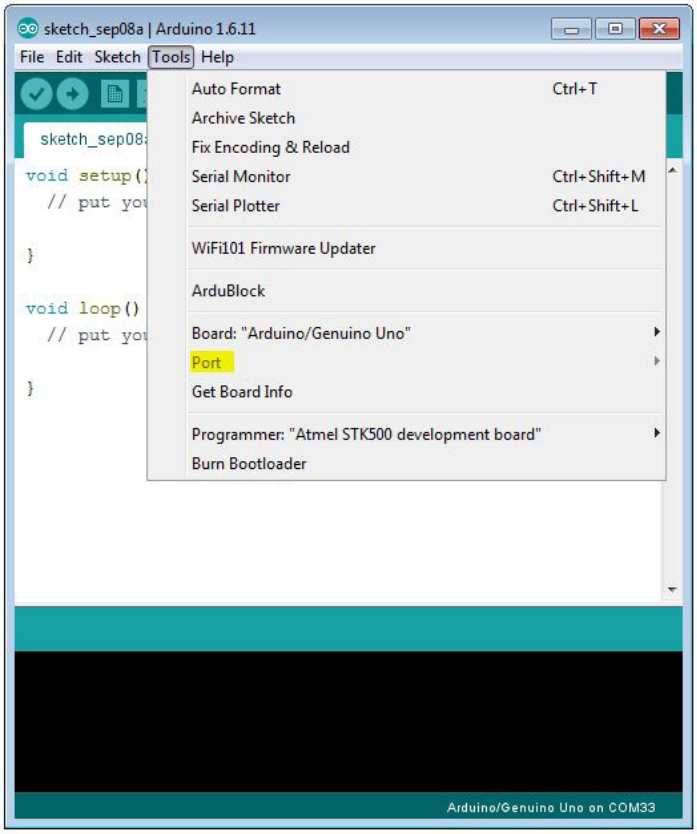
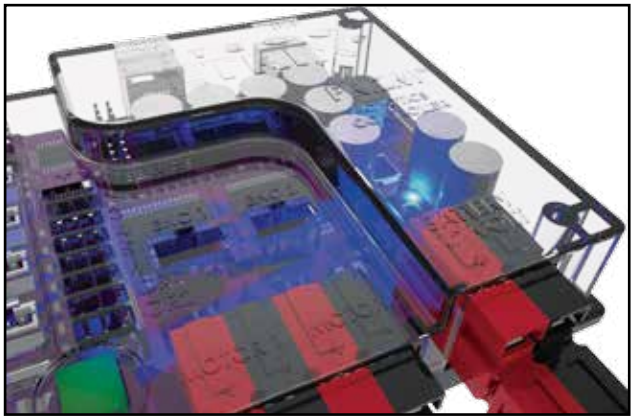


Рисунок 4: Пункт выпадающего меню "Port" при отключенном контроллере PRIZM. Учтите, что содержимое меню может быть разным.

Далее подсоедините контроллер PRIZM к порту USB и подайте на него электропитание, подсоединив аккумуляторную батарею TETRIX и нажав на кнопку включения.



После включения должен загореться синий светодиод. Самое первое подключение у контроллера PRIZM настраивается дольше других. На это может уйти от 5 до 10 секунд. После подсоединения и установки контроллер PRIZM получит в компьютерной системе свой порт связи (порт COM).

Пройдите в меню **Tools > Port** и выберите вновь установленный порт связи (порт COM). Новый порт связи будет закреплён за контроллером PRIZM. Выбирая новый порт COM, вы даёте *ПО Arduino (IDE)* команду использовать его для связи. Порт связи, который окажется в вашем распоряжении, наверняка будет выглядеть иначе, чем изображённый на рисунке 5.

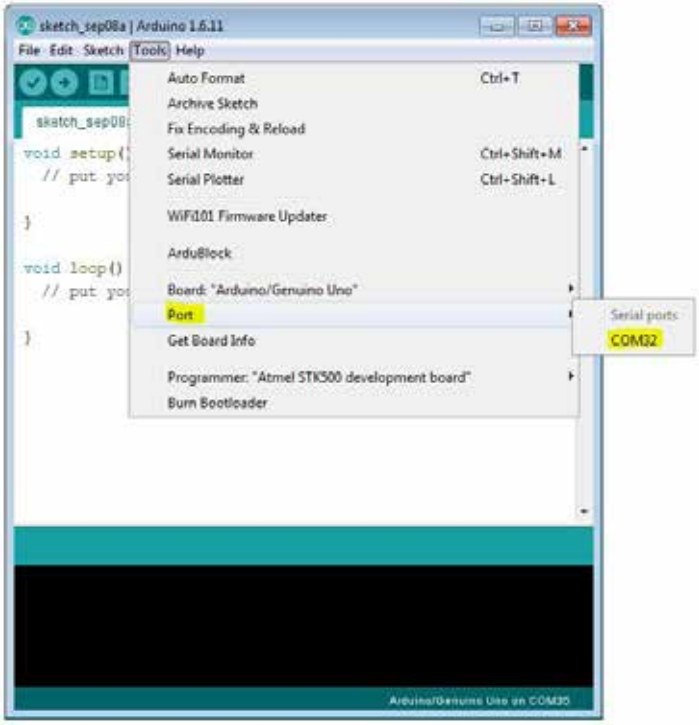


Рисунок 5: Выпадающее меню с занятым портом после подсоединения контроллера PRIZM к порту USB и подачи электропитания.

В этом примере появляется новый порт COM32.

Выберите новый порт — для *ПО Arduino (IDE)* это будет означать, что порт надо использовать для связи. Наш порт скорее всего будет выглядеть иначе, но это не повод для беспокойства. Выбор порта связи означает, что связь с контроллером PRIZM налажена, так что можно браться за составление управляющих кодов.

По завершении этого шага операционная система компьютера автоматически будет обращаться к этому выбранному порту при каждом подсоединении контроллера PRIZM и запуске *ПО Arduino (IDE)*.

Подсказка: На одном и том же компьютере каждое устройство PRIZM будет использовать отдельный порт связи. Всякий раз, подключая новый контроллер PRIZM, соблюдайте правила нумерации и привязки к компьютеру, изложенные выше. Для простоты можно пронумеровать каждый контроллер PRIZM и закрепить его за соответствующим компьютером. Так компьютеру будет проще выбирать правильный порт для контроллера PRIZM при всяком подсоединении и включении.

Начальные упражнения

Пора приступить к упражнениям. Каждое из пяти начальных упражнений призвано познакомить вас с *программным обеспечением (ПО) Arduino (IDE)* и тем, как оно взаимодействует с контроллером PRIZM и специально отобранной аппаратурой начального уровня. Успешно выполнив эти первые пять упражнений, вы увидите, насколько легко использовать *ПО Arduino (IDE)* с контроллером PRIZM и подготовитесь к программированию модели Робота-исполнителя серии PRIZM.

Скетч на языке Arduino

Приступая к созданию первого скетча, важно усвоить несколько основных правил, касающихся скетчей. Лучше всего представить себе скетч в виде перечня команд, подлежащих выполнению в порядке их записи. В каждом скетче будет несколько команд, а каждая команда обычно будет состоять из одной строки текста внутри скетча. Текстовые строки внутри скетча ещё называют кодом, именно поэтому программирование иногда называют написанием управляющих кодов или кодированием.

Большинство правил программирования на основе текста называют синтаксисом, и в этом есть сходство с грамматикой и пунктуацией обычного языка.

Так, каждая кодовая строка должна заканчиваться точкой с запятой, подобно тому, как точка обозначает конец предложения. По мере появления в упражнениях новых правил кодирования мы их обязательно объясним.

Зачастую лучше учиться на примерах. В следующих разделах мы разберём несколько примеров кодирования, чтобы лучше понять, как создавать скетчи и подгружать их в контроллер PRIZM.

Пример скетча каждой функции можно найти в выпадающем меню **File > Examples > TETRIX_PRIZM**.

Примечание: Если вы уже освоились с созданием скетчей в ПО Arduino и хотите двинуться дальше, то начать всё же полезно с просмотра каждой библиотечной функции. Мы сформулировали и включили в Руководство определения функций, наряду с описаниями, которые показывают, как каждая из них будет выглядеть Arduino скетче. Ищите их в приложении к этому руководству на сс. 133-145. Функций в библиотеке со временем может стать больше, поскольку библиотека обновляется с появлением более современных версий.

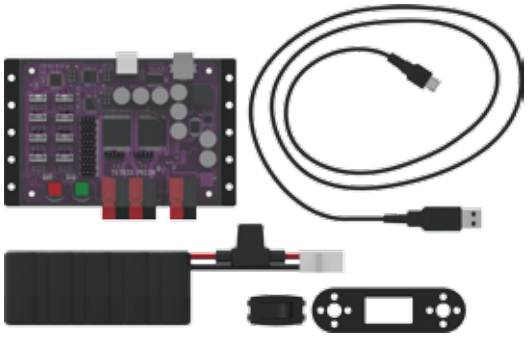
Примечание: Кроме библиотеки функций контроллера PRIZM есть полное собрание команд для Arduino, в которых необходимо разобраться, прежде чем мы сможем создавать функциональные программы. Справочник по языку Arduino, а также многочисленные учебные пособия по этому языку есть на домашней странице компании-разработчика Arduino: **www.arduino.cc**.

Упражнение 1: Привет, мир!

Начнём с очень простого скетча, который заставит мигать встроенный в контроллер PRIZM красный светодиод. Упражнение сводится к составлению программы "Привет, мир!" для контроллера PRIZM и служит вводным упражнением для любого начинающего программиста. Скетч, который мы создадим, относится к простейшим и самым базовым функциям PRIZM. Нам понадобится только PRIZM, источник питания и кабель USB для соединения с компьютером.

Необходимые детали и принадлежности

- Контроллер PRIZM
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Переходник для выключателя аккумуляторной батареи контроллера PRIZM
- Компьютер



Открывание скетча

Для начала посмотрим на первый пример нашего скетча. Откройте скетч, последовательно выбирая в меню пункты **File > Examples > TETRIX_PRIZM > GettingStarted_Act1_Blink_RedLED**. Откроется окно нового скетча под названием GettingStarted_Act1_Blink_RedLED (рисунок 6).

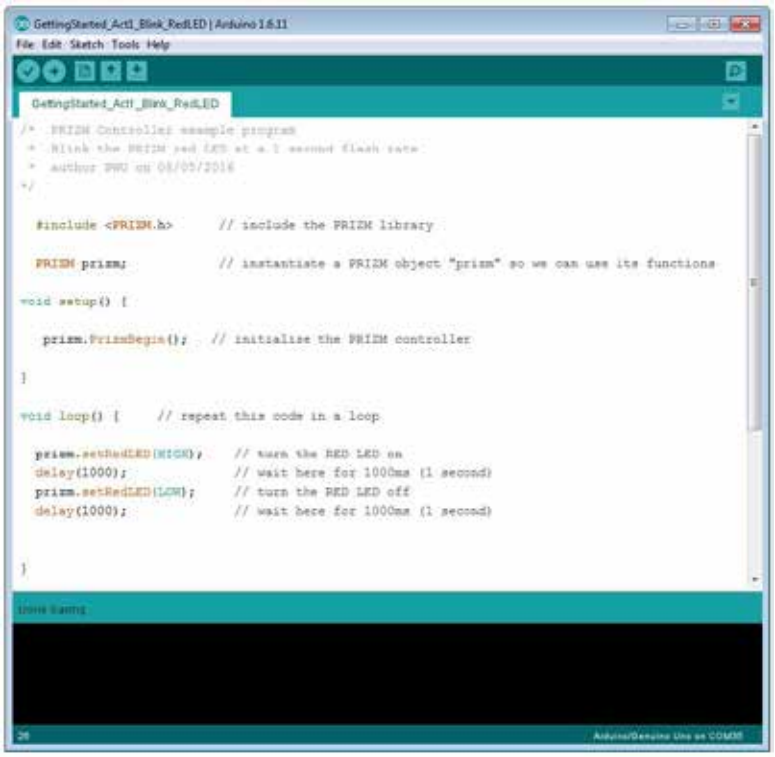


Рисунок 6

Построение базы знаний

Прежде чем подгрузить скетч в PRIZM, нужно убедиться, что контроллер включён, подсоединён к компьютеру и обнаружен компьютером (рисунок 7). Подсоединив PRIZM, как показано на иллюстрации, включите его с помощью выключателя. Узнать, что PRIZM включён, можно по загоревшемуся синему светодиоду. Чтобы понять, обнаружил ли компьютер контроллер PRIZM, проверьте порт так же, как мы делали это в разделе про настройку связи через USB.

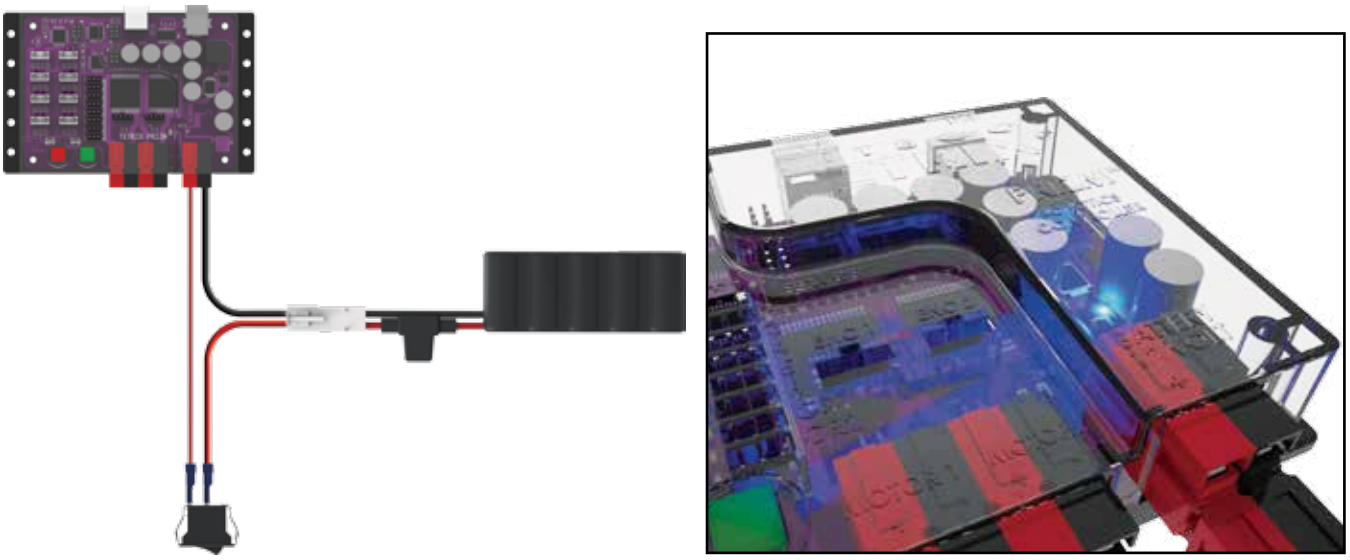


Рисунок 7

Выполнение управляющего кода

Чтобы подгрузить скетч в контроллер PRIZM, щёлкните по кнопке "Загрузить" (**Upload**) (рисунок 8).

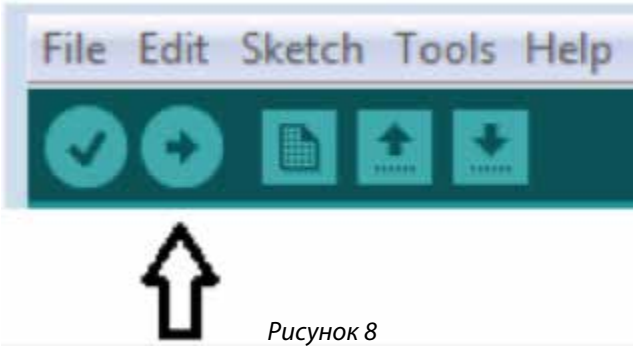


Рисунок 8

Пока загружаются данные, сборку от порта USB будут мигать желтые светодиоды. Когда подгрузка закончится, рядом с кнопкой возврата в исходное состояние (Reset) будет ровно светиться зелёный светодиод. Зелёный светодиод означает готовность программы к выполнению. Чтобы выполнить программу, нажмите кнопку пуска (Start). Красный светодиод возле кнопки возврата в исходное состояние (Reset) будет включаться-выключаться с секундной периодичностью. Чтобы остановить программу, нажмите кнопку возврата в исходное состояние (Reset).

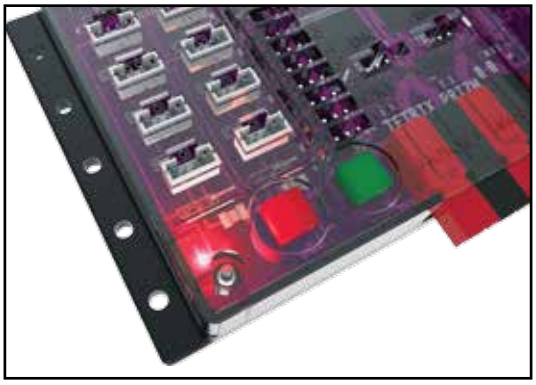
Поздравляем! Вы успешно загрузили в контроллер PRIZM свой первый скетч и показали результаты всему человечеству.

Последующие действия

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, которые заставляют светодиод мигать, читайте следующие разделы в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страница 133: операторы включения
- Страница 134: функции инициализации
- Страница 136: кнопка пуска
- Страница 137: встроенные светодиоды

Давайте на время изменим некоторые параметры в нашем коде и посмотрим, как это отразится на поведении красного светодиода. Согласно примечаниям в примере, длительность включённого или выключенного состояния светодиода определяется функцией задержки. Это один из параметров, которые мы можем поменять в нашем коде. Поэкспериментируйте, меняя эти значения, чтобы добиться от светодиода новых вариантов мигания. Попробуйте заставить светодиод мигать быстрее или медленнее.



Связь с настоящей жизнью

В окружающем нас мире электроники много мигающих устройств: сигнальные фонари в местах ведения дорожных работ, вафельницы (мигание сменяется ровным свечением, когда вафли готовы), светодиоды в наших телефонах, извещающие о входящих звонках. Скорость их мигания — или время их мигания — регулируется электроникой. При этом сигналы управления могут поступать от простых схем синхронизации или от компьютеров или иных электронных устройств.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Электротехнические термины (напряжение, ток и сопротивление)
- Как работает светодиод
- От чего зависит цвет светодиода

Технология

- Как работают компьютеры
- Как программируются компьютеры

Проектирование

- Решение задач

Математика

- Частота
- Продолжительность
- Последовательность

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототроителя, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Упражнение по программированию

Опираясь на этот пример, попытайтесь добиться мигания светодиода с помощью нового скетча. Не ограничиваясь миганием красного светодиода, постарайтесь добиться мигания и зелёного светодиода. Вспыхивающие или мигающие фонари издавна используются в качестве сигналов или для передачи сообщений на большие расстояния. Можете поставить себе задачу составить программу для передачи сообщения "Привет, мир!" мигающим кодом Морзе.

Чтобы создать новый скетч, последовательно выберите в меню пункты **File > New**. Обязательно ознакомьтесь с приложением "Библиотечные функции Arduino для контроллера TETRIX PRIZM" в конце руководства, чтобы изучить структуру и синтаксис команд. Те, кому понадобится дополнительная помощь, могут также найти пример библиотеки управляющих кодов в приложении. Не забывайте пользоваться при работе с программным обеспечением простейшими действиями, вроде "скопировать" и "вставить".

В программном обеспечении есть инструмент, который должен помочь вам избежать синтаксических ошибок в создаваемом коде. Проверить синтаксис можно, щёлкнув по кнопке "Проверить" (**Verify**) (рисунок 9). В результате код скомпилируется, но подгружаться не будет. Перед проверкой вам будет предложено сохранить скетч.

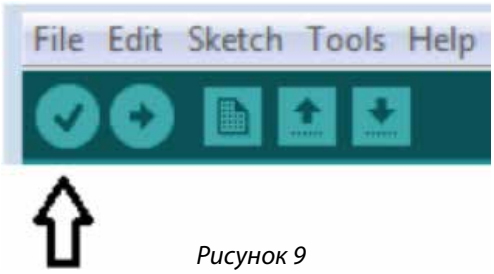


Рисунок 9

Если в коде есть ошибки, компилятор покажет их в окне ошибок в нижней части окна скетча (рисунок 10).
Сначала необходимо исправить ошибки и только потом подгружать код в контроллер PRIZM. Если ошибок нет, компилятор закончит компиляцию и покажет, что закончил её, после чего можно будет подгружать код.

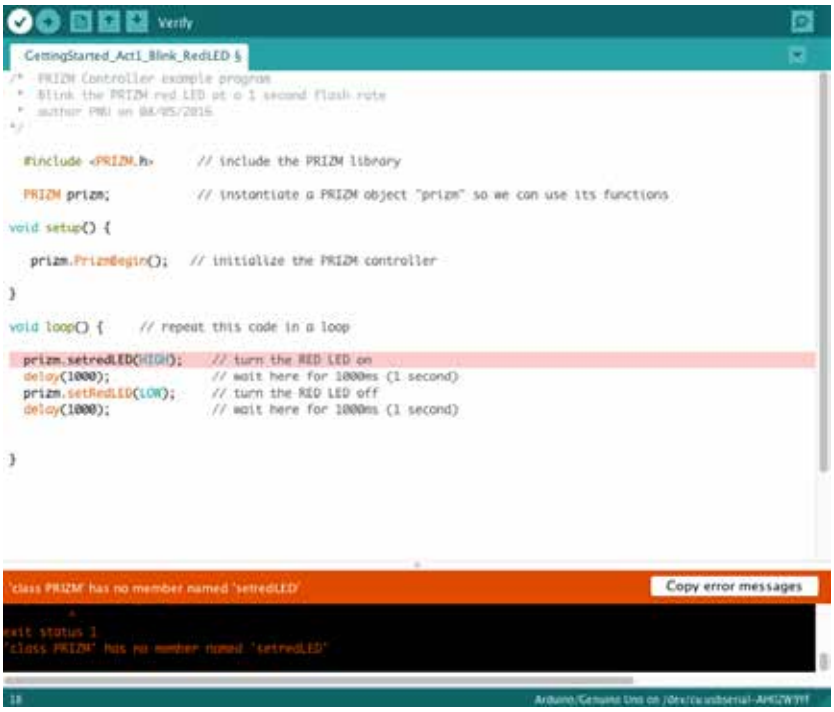


Рисунок 10

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

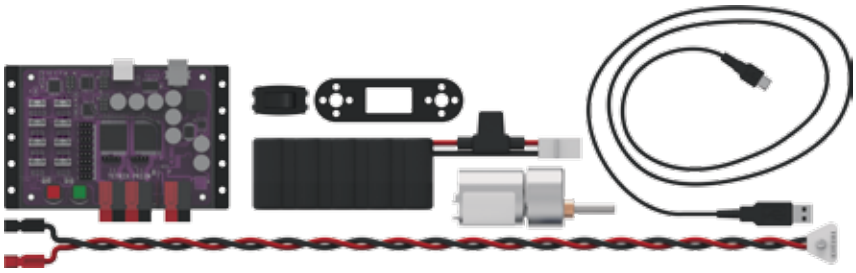
Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 2: Приведение в движение электродвигателей постоянного тока

Для второго упражнения не будем ничего усложнять, но добавим движение. Создадим скетч, который будет вращать вал электродвигателя постоянного тока из конструктора TETRIX.

Необходимые детали и принадлежности

- Электродвигатель постоянного тока из конструктора TETRIX
- Кабель с разъёмами Powerpole для электродвигателя из конструктора TETRIX
- Контроллер PRIZM
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Кнопка включения и выключения питания от батареи PRIZM
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца. Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > GettingStarted_Act2_Move_DCMotor.** Откроется окно нового скетча под названием GettingStarted_Act2_Move_DCMotor (рисунок 11).

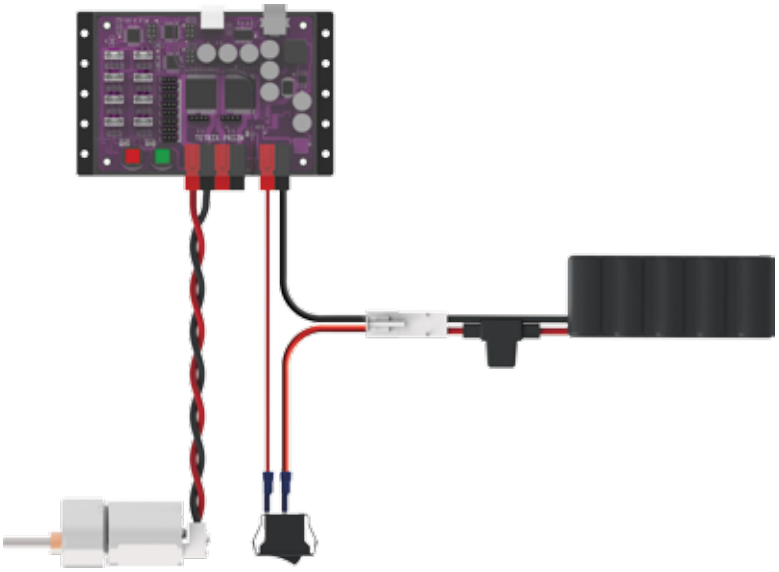
Рисунок 11

Построение базы знаний

Во втором скетче стоит внимательнее рассмотреть синтаксис скетча, а именно комментарии. Комментарии представляют собой строки в программе, которые сообщают вам или кому-то ещё о том, как работает программа. Они служат только для уведомления и не влияют на управляющий код.

Есть два вида заметок: однострочные и многострочные. Перед однострочными заметками стоят символы // и всё, что находится между // и концом строки — это заметка. Перед многострочными заметками ставится значок /* и заметка может состоять из нескольких строк, но должна завершаться значком */.

При взгляде на второй скетч видно, что комментарии объясняют, как автор скетча представлял себе работу этой программы. Назначение этого скетча — заставить вращаться электродвигатель постоянного тока на канале № 1 в течение пяти секунд, а потом плавно остановить. Через две секунды после плавной остановки, электродвигатель начнёт вращаться в противоположном направлении. Так будет продолжаться, пока вы не нажмёте красную кнопку возврата в исходное состояние (Reset).



Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Помните, что мы добавили новое соединение с электродвигателем.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, нажмите зелёную кнопку пуска на контроллере PRIZM.

Понаблюдайте за направлением и продолжительностью вращения электродвигателя. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Когда будете готовы остановить электродвигатель, нажмите красную кнопку возврата в исходное состояние (Reset).

Последующие действия

В этом скетче используется функция prizm.setMotorPower. У неё два параметра: канал управления электродвигателем и мощность электродвигателя. В примере prizm.setMotorPower(1,25) означает, что электродвигатель № 1 будет вращаться с мощностью 25 % по часовой стрелке. Первое значение в круглых скобках определяет канал управления электродвигателем, а второе значение в круглых скобках определяет мощность в процентах и направление вращения.

Можно изменить направление вращения и порядок остановки, если заменить второе значение в круглых скобках. Если второе значение отрицательно, то электродвигатель вращается против часовой стрелки, как видно в скетче. Если вместо 0 вставить 125, то остановка произойдёт не в результате плавной остановки с выбегом, а в результате торможения.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в приведении электродвигателя в движение, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 137-142: функции электродвигателя постоянного тока

Поупражняйтесь в изменении параметров в скетче. Мы можем изменить мощность электродвигателя, направление вращения вала электродвигателя, порядок остановки и задержку между функциями. Понаблюдайте, как эти изменения сказываются на электродвигателе.

Связь с настоящей жизнью

В управлении работой электродвигателей нет ничего нового. А вот управление их работой посредством бортового компьютера в легковом электромобиле (например, марки Tesla) на скорости 112 км/ч в движении по дороге или во время резкого поворота на извилистом шоссе — это нечто новое! Чтобы такие электроприводные автомобили выполнили поворот на скоростном шоссе, когда водитель вращает руль, необходима координация угловой скорости и мощности всех приводных электродвигателей. Всё это предусмотрено в программе, заложенной в электронную начинку таких автомобилей, чтобы водителю было просто и легко ими управлять.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Как работают электродвигатели постоянного тока
- Угловая скорость

Технология

- Зависимость между мощностью, напряжением и силой тока
- Крутящий момент

Проектирование

- Определение нагрузки и крутящего момента

Математика

- Широтно-импульсная модуляция (ШИМ)
- Обороты в минуту (об/мин)

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит вращаться ваш электродвигатель постоянного тока. Вспомните, чему мы научились, выполняя первое упражнение, и придумайте необычные способы использовать мигающие светодиоды с вращающимся электродвигателем.

Обязательно ознакомьтесь с приложением Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM на странице 133 в помощь при проверке структуры и синтаксиса скетча.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототроителя, дополняющую Руководство по программированию контроллера PRIZM. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

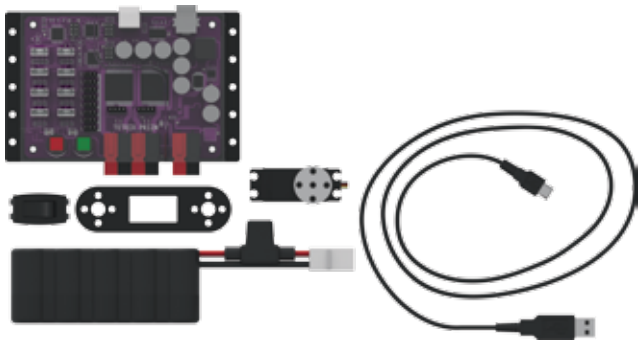
Упражнение 3: Приведение в движение сервоприводов

Третье упражнение поможет нам изучить на примере сервопривода ещё один элемент движения. Создадим скетч, который будет вращать сервопривод.

Достоинство сервоприводов заключается в способности занимать заданное положение вне зависимости от исходного положения в пределах ограниченного угла перемещения. У сервоприводов вал поворачивается на определённый угол в пределах от 0 до 180 градусов. Например, можно дать сервоприводу команду занять положение 45 градусов вне зависимости от исходной точки поворота. Если он начнет поворот с 0 градусов, то сдвинется по часовой стрелке в положение, соответствующее 45 градусам. Если он начнет поворот со 120 градусов, то сдвинется против часовой стрелки в положение, соответствующее 45 градусам.

Необходимые детали и принадлежности

- Стандартный сервопривод HS-485HB с поворотом вала на 180 градусов
- Контроллер PRIZM
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Кнопка включения и выключения питания от батареи PRIZM
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File>Examples>TETRIX_PRIZM>GettingStarted_Act3_Move_Servo**. Откроется окно нового скетча под названием GettingStarted_Act3_Move_Servo (рисунок 12).

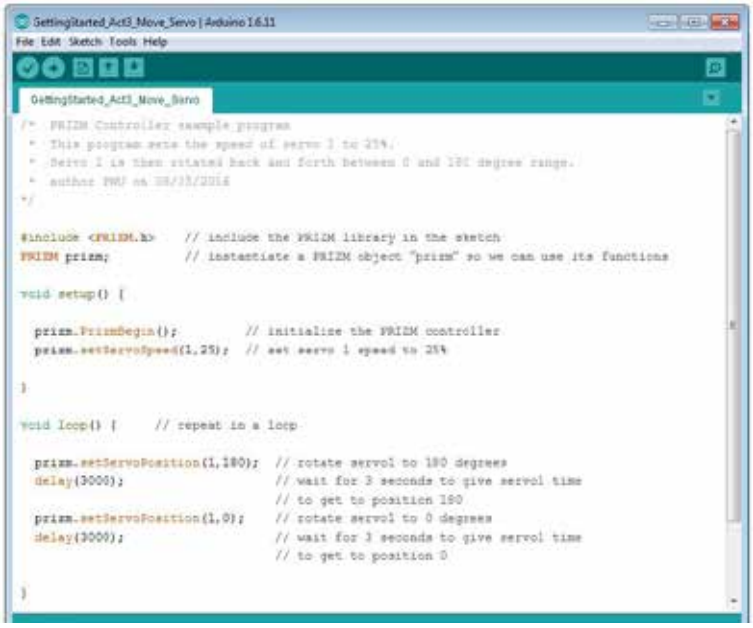


Рисунок 12

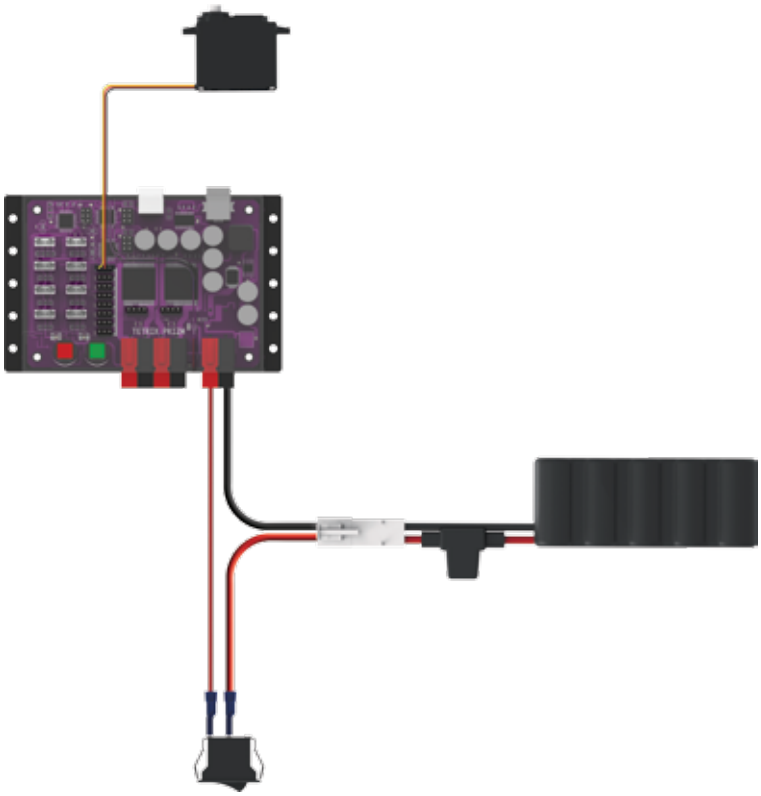
Построение базы знаний

В третьем скетче мы продолжим изучение синтаксиса скетчей и, в частности, обратим внимание на оператор включения и объявление объекта. Оператор включения служит для добавления в скетч функциональных возможностей библиотеки контроллера PRIZM. Объявление объекта — это приём, используемый при написании программ, позволяющий облегчить управление ими по мере роста их размера и сложности.

Библиотека программного обеспечения контроллера PRIZM представляет собой собрание особых мини-программ, и каждая — с собственным особым именем функции. Назначение этих мини-программ — сделать написание скетчей для контроллера PRIZM лёгким и интуитивно понятным. При помощи оператора включения мы добавляем функциональные возможности библиотеки в наш скетч. Оператор включения для контроллера PRIZM: #include <PRIZM.h>.

Объявление объекта — важный оператор при использовании контроллера PRIZM наряду с библиотекой котроллера PRIZM. Чтобы воспользоваться функциями из библиотеки программ PRIZM, сначала нам необходимо объявить имя объекта библиотеки, который после этого вставляется в виде "приставки" перед каждой библиотечной функцией. Объявление объекта, которое мы используем, таково: PRIZM prizm;. Этот оператор мы используем сразу после оператора включения.

Во всех наших примерах скетчей мы используем оператор включения и объявление объекта с целью добавить в скетчи функции из библиотеки программного обеспечения PRIZM. Операторы включения и объявления объекта типичны для большинства видов языков, построенных на основе языка Си.



Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Помните, что мы добавили новое соединение с сервоприводом.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, нажмите зелёную кнопку пуска на контроллере PRIZM.

Понаблюдайте за направлением и продолжительностью вращения сервопривода. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Когда будете готовы остановить электродвигатель, нажмите красную кнопку возврата в исходное состояние (Reset).

Последующие действия

В этом скетче мы впервые знакомимся с двумя новыми функциями PRIZM, а именно: prizm.setServoSpeed и prizm.setServoPosition. У обеих функций по два параметра, но разных.

Два параметра функции prizm.setServoSpeed — это канал управления сервоприводом и скорость поворота вала сервопривода. В примере prizm.setServoSpeed(1,25) означает, что сервопривод № 1 повернётся с мощностью 25 % в положение, указанное функцией prizm.setServoPosition. Вызвать эту функцию придётся только один раз, в начале программы.

Два параметра функции prizm.setServoPosition — это канал сервопривода и заданное конечное положение. В примере prizm.setServoPosition(1,180) означает, что сервопривод № 1 повернётся в заданное конечное положение, соответствующее 180 градусам.

В скетче обе эти функции вместе сообщают сервоприводу не только заданное конечное положение, но и скорость, с которой следует двигаться к заданному конечному положению. Мы можем изменить положение и скорость поворота сервопривода, заменив значения обеих функций.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в приведении сервопривода в движение, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 144-145: функции сервопривода

Поупражняйтесь в изменении параметров в скетче. Понаблюдайте, как эти изменения сказываются на сервоприводе.

Связь с настоящей жизнью

В прошлом сервоприводы главным образом использовались в сочетании с передатчиками радиосигналов управления для радиоуправляемых моделей автомобилей. При помощи этих передатчиков, иначе называемых пультами дистанционного управления (ПДУ), управляли закрылками и хвостовым оперением радиоуправляемых моделей самолётов. В роботах можно использовать сервоприводы под управлением пультов ДУ — но можно использовать и сервоприводы под управлением контроллера PRIZM, чтобы приводить в действие руки роботов, их захватные устройства, наклонять и поворачивать монтажные опоры видеокамер, или выполнять многие другие перемещения, требующие точности.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Рычаги
- Центростремительная сила

Технология

- Рычажные механизмы
- Передача силы

Проектирование

- Приложение момента силы

Математика

- Радианные измерения в сравнении с измерениями в декартовой системе координат
- Длина дуги

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит поворачиваться вал вашего сервопривода. Вспомните, чему мы научились, выполняя предыдущие упражнения, и придумайте необычные способы объединить пройденные функции.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

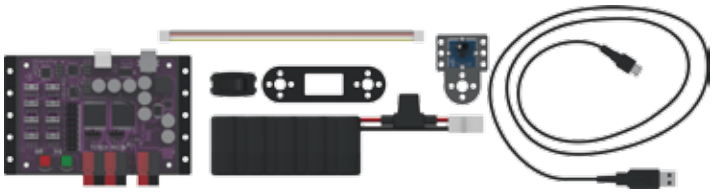
Упражнение 4: Первое знакомство с датчиком линии

Для четвёртого упражнения мы переключим внимание с выходных параметров электродвигателей на поступающие от датчиков сигналы, для чего ознакомимся с датчиками и изучим их. В данном примере мы подсоединим датчик линии к порту D3 для цифровых датчиков и создадим скетч для считывания цифрового сигнала, поступающего от датчика линии.

Датчики позволяют нам собирать информацию об окружающем мире. Каждый датчик собирает информацию определённого вида. Датчик линии при помощи отражённых инфракрасных лучей различает светлые и тёмные поверхности.

Необходимые детали и принадлежности

- Сравнение светлой и тёмной поверхности
- Датчик линии и кабель серии Grove
- Контроллер PRIZM
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Кнопка включения и выключения питания от батареи PRIZM
- Компьютер



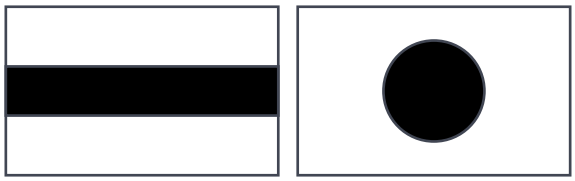


Рисунок 13: Светлая поверхность с контрастными темными элементами

Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > GettingStarted_Act4_Intro_LineFinder**. Откроется окно нового скетча под названием GettingStarted_Act4_Intro_LineFinder (рисунок 14).

Рисунок 14

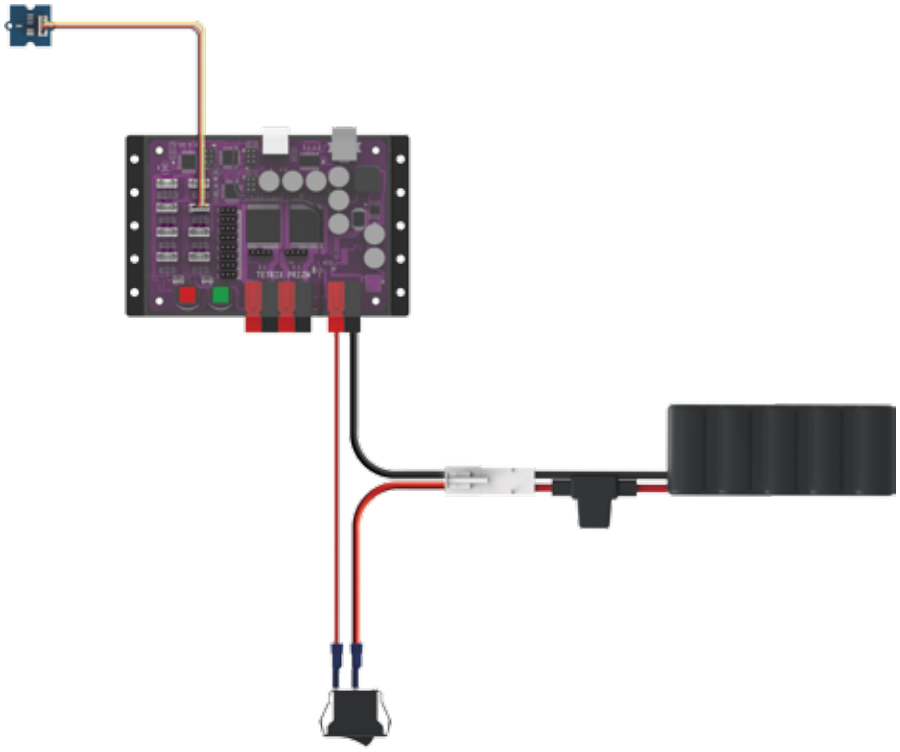
Построение базы знаний

Для четвёртого скетча понадобится внимательнее изучить два основополагающих структурных элемента в составе любого скетча. В каждом из скетчей, которые вы пишете, будет 2 функции - setup и loop.

Функция setup следует сразу за оператором включения и объявлением объекта, будучи составной частью начала нашего кода. В настроечной() функции есть данные, которые понадобятся в составе скетча только один раз. Здесь могут оказаться многие функции, но мы всегда используем по крайней мере оператор инициализации контроллера PRIZM: prizm.PrizmBegin(). Главное назначение этой функции — настройка кнопки пуска.

Функция loop занимается именно тем, о чём можно догадаться по её названию. Всё, что находится в функции loop, будет непрерывно повторяться, пока скетч не закончится по команде или по нажатию кнопки возврата в исходное состояние (Reset). В петле() содержится основная часть нашего кода.

Содержательная часть настроечной() и петлевой() функций заключена в фигурные скобки. Открывающая (левая) фигурная скобка { стоит в начале группы операторов, а закрывающая (правая) фигурная скобка } стоит в конце группы операторов. Можно мысленно представить их себе в виде ограничителей, удерживающих стоящие на полке книги, а код, заключённый в скобки, называют блоком кода или блоком операторов.



Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Помните, что мы добавили порту 3 для цифровых датчиков новое соединение с датчиком линии.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, нажмите зелёную кнопку пуска на контроллере PRIZM.

Подержите датчик над контрастной поверхностью. Сдвигая датчик со светлой области на тёмную, наблюдайте за красным светодиодом на контроллере PRIZM. Когда датчик окажется над линией, над поверхностью, не отражающей свет, или над тёмной поверхностью, красный светодиод погаснет. Когда датчик окажется над белой поверхностью или над поверхностью, отражающей свет, красный светодиод загорится.

Когда будете готовы остановить датчик, нажмите красную кнопку возврата в исходное состояние (Reset).

Последующие действия

При помощи этого скетча мы получаем начальное представление об управляющей структуре, о новой функции и об операторе сравнения. Структура программы представляет собой оператор "если", новая функция выглядит так: prizm.readLineSensor, а оператор сравнения так: == (равно).

Простейший оператор "если" позволяет нам проверить соблюдение некоторого условия. Если условие соблюдено, тогда программа может выполнить некое действие. Если оператор в виде утверждения в круглых скобках является истинным, тогда код в скобках выполняется, если оператор не является истинным, тогда программа пропустит этот код.

Функция prizm.readLineSensor считывает состояние датчика линии и отправляет обратно значение "1" (HIGH) или "0" (LOW). Когда датчик линии обнаруживает тёмную линию или неотражающую поверхность, он посылает значение "1", а значение "0" он посылает, когда обнаруживает белую или отражающую поверхность.

Оператор сравнения == (равно) определяет вид проверки.

Когда в скетче объединяются эти три элемента, мы создаём проверяемое условие на основе сигнала, поступающего от датчика линии, который заставляет красный светодиод загораться или гаснуть.. Проще говоря, если датчик линии обнаруживает линию или неотражающую поверхность, он выключает красный светодиод. Если датчик линии обнаруживает белую или отражающую поверхность, он включает этот светодиод.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в использовании датчика линии, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков

Опробуйте датчик линии на разных поверхностях и на разной высоте над ними, чтобы посмотреть, как реагирует этот датчик.

Примечание: У датчика линии диапазон обнаружения можно подстраивать, вращая небольшой регулировочный винт на тыльной стороне платы с датчиком.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Связь с настоящей жизнью

В этом мире бывает непросто отыскать нужный нам путь. Объяснить роботу, как ему найти заданный путь, тоже может оказаться непростым делом. Один из способов указать складским роботам дорогу к нужному месту — заставить их двигаться по разметочным линиям. Но чтобы двигаться по линиям, роботы должны эти линии обнаружить. Этого можно добиться, используя датчик, различающий тёмные и светлые поверхности. Он, при помощи компьютерного кода, сообщает роботу, где находится искомая линия. Другой код, управляя электродвигателями постоянного тока, подправляет направление движения, если робот отклоняется от линии.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Свет — отражение и поглощение
- Электромагнитный спектр

Технология

- Цифровые и аналоговые сигналы
- Калибровка (тарировка)

Проектирование

- Определение места нахождения края

Математика

- Анализ данных

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит работать ваш датчик линии. Вспомните, чему мы научились, выполняя предыдущие упражнения, и придумайте дополнительные необычные способы работы, основанные на условии для датчика линии.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

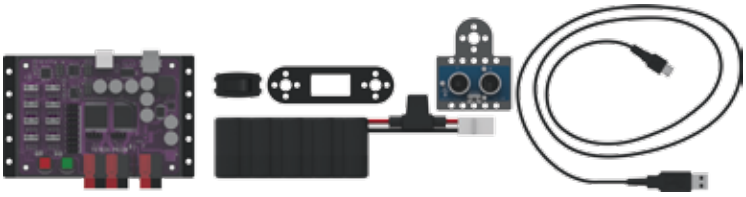
Упражнение 5: Знакомство с ультразвуковым датчиком

В последнем из начальных упражнений мы завершим изучение датчиков, создав скетч для использования ультразвукового датчика. В этом упражнении мы подсоединим ультразвуковой датчик к порту D3 для цифровых датчиков и отобразим расстояние до объекта, который поместим перед ним, в окне прибора контроля последовательной передачи данных.

Как и прочие датчики, ультразвуковой датчик помогает нам собирать информацию. Собранная ультразвуковым датчиком информация — это информация о расстоянии. Датчик испускает короткую серию звуковых импульсов и ждёт, когда они вернутся, отразившись от некоего объекта в пространстве. По времени, прошедшему между испусканием и возвращением серии звуковых импульсов, определяется расстояние до объекта. Датчик может измерять расстояние в диапазоне от 3 до 400 сантиметров.

Необходимые детали и принадлежности

- Ультразвуковой датчик и кабель серии Grove
- Контроллер PRIZM
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Кнопка включения и выключения питания от батареи PRIZM
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

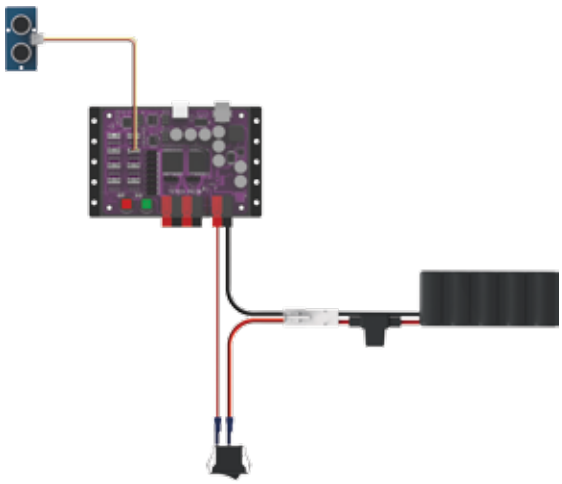
Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > GettingStarted_Act5_Intro_UltraSonic**. Откроется окно нового скетча под названием GettingStarted_Act5_Intro_UltraSonic (рисунок 15).

Рисунок 15

Построение базы знаний

Для нашего пятого скетча, создаваемого в рамках начальных упражнений, нам нужно рассмотреть полезное средство обзора данных, выводимых в виде текста. Монитор последовательного порта отображает данные, отправляемые контроллером PRIZM через последовательный порт связи по USB соединению.

Достоинство такого инструмента, как прибор контроля последовательной передачи данных, заключается в его способности отображать данные в реальном времени, — ведь это позволяет вам принимать более обоснованные решения, касающиеся конструкций и программирования создаваемых роботов. В нём можно показывать данные, поступающие от датчиков, подключённых к контроллеру PRIZM, или проверять любые данные, собранные контроллером PRIZM, например показания счётчика энкодера, данные о токе, потребляемом электродвигателем постоянного тока, или данные о положении вала сервопривода.



Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Помните, что мы добавили порту 3 для цифровых датчиков новое соединение с ультразвуковым датчиком.

Подгрузите скетч. Перед прогоном скетча надо из окна скетча открыть прибор контроля последовательной передачи данных. Чтобы открыть прибор контроля последовательной передачи данных, щёлкните по значку увеличительного стекла в правом верхнем углу окна скетча (рисунок 16).

Рисунок 16

Прибор контроля последовательной передачи данных откроется в отдельном окне (рисунок 17).

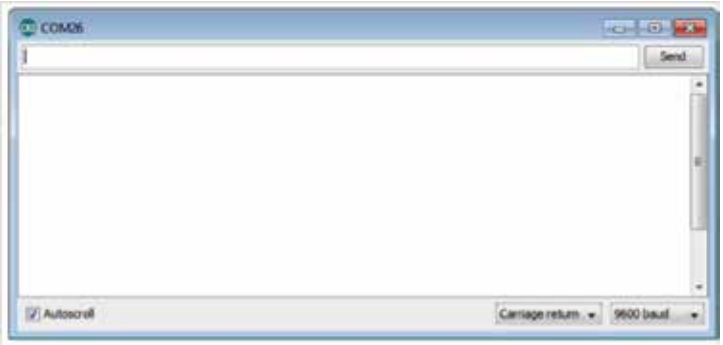


Рисунок 17

Положив датчик плашмя на стол рабочей частью вверх, нажмите зелёную кнопку пуска, чтобы выполнить код.

Подержите над датчиком на разных расстояниях какой-нибудь предмет и понаблюдайте за данными, которые отображаются в реальном времени в окне прибора контроля последовательной передачи данных.

Когда будете готовы остановить датчик, нажмите красную кнопку возврата в исходное состояние (Reset).

Последующие действия

В скетче впервые появляется несколько новых функций: Serial.begin(), Serial.print(),
prizm.readSonicSensorCM() и Serial.println().

Serial.begin() позволяет использовать в скетче последовательную передачу данных. Будучи функцией инициализации, которая выполняется только один раз, она относится к настроечной части скетча. Важная часть функции Serial.begin() — это скорость передачи данных, которая измеряется в битах в секунду. Стандартное значение для скорости передачи данных в окне контроля последовательной передачи данных составляет 9600, поэтому именно его мы используем в функции Serial.begin().

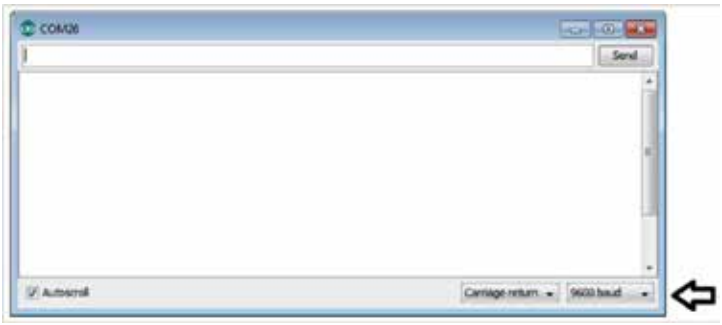


Рисунок 18

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Функция Serial.print() отправляет данные в последовательный порт в виде удобочитаемого текста. Он может подаваться в виде динамичной информации от другого устройства или в виде статичной информации от программиста.

Функция prizm.readSonicSensorCM() считывает состояние ультразвукового датчика и выдаёт цифровое значение в заданных пределах измерений. У нашего датчика дальность измерений составляет от 3 до 400 сантиметров. Это значение должно отражать расстояние, на котором ультразвуковой датчик находится от обнаруживаемого объекта.

Функция Serial.println() распечатывает данные через порт последовательной передачи данных в виде удобочитаемого текста, за которым следует встроенная команда на создание новой строки.

Рисунок 19

Эти четыре функции могут показаться сложными, но на самом деле совместно они действуют просто. В этом скетче функция Serial.begin(9600) разрешает и определяет скорость передачи данных в настройках. Функция Serial.print() указывает, какой вид данных предстоит распечатать.
Функция prizm.readSonicSensorCM() предоставляет тот вид данных, который предстоит распечатать, потому что находится в круглых скобках в функции Serial.print(). А функция Serial.println("Centimeters") поясняет, какой вид данных печатается, — в данном случае с модификатором "Сантиметры".

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе ультразвукового датчика и прибора контроля последовательной передачи данных см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков

Связь с настоящей жизнью

В самом начале жизни вас, вероятно, сканировали с помощью ультразвукового датчика. Если вы ничего такого не помните, то лишь потому, что это происходило до вашего рождения. Одна из областей, в которой ультразвуковая техника применяется с потрясающими результатами, — это медицина. Врачи используют датчик для улавливания отражённых звуковых волн, чтобы заглянуть внутрь живого человека. Врачи могут видеть размер эмбриона и степень его развития и на основе этого установить приблизительный срок родов!

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Терминология, связанная со звуковыми волнами (частота, амплитуда, гребень, впадина)
- Отражение звуковых волн

Технология

- Измерительная частота
- Оцифровка звука

Проектирование

- Области применения звуковых методов измерения

Математика

- Закон обратных квадратов

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит работать ваш ультразвуковой датчик в сочетании с прибором контроля последовательной передачи данных. Вспомните, чему мы научились, выполняя предыдущие упражнения, и попробуйте поместить разные предметы перед ультразвуковым датчиком, чтобы увидеть, обнаруживает ли он их и способен ли точно измерить расстояние до них.

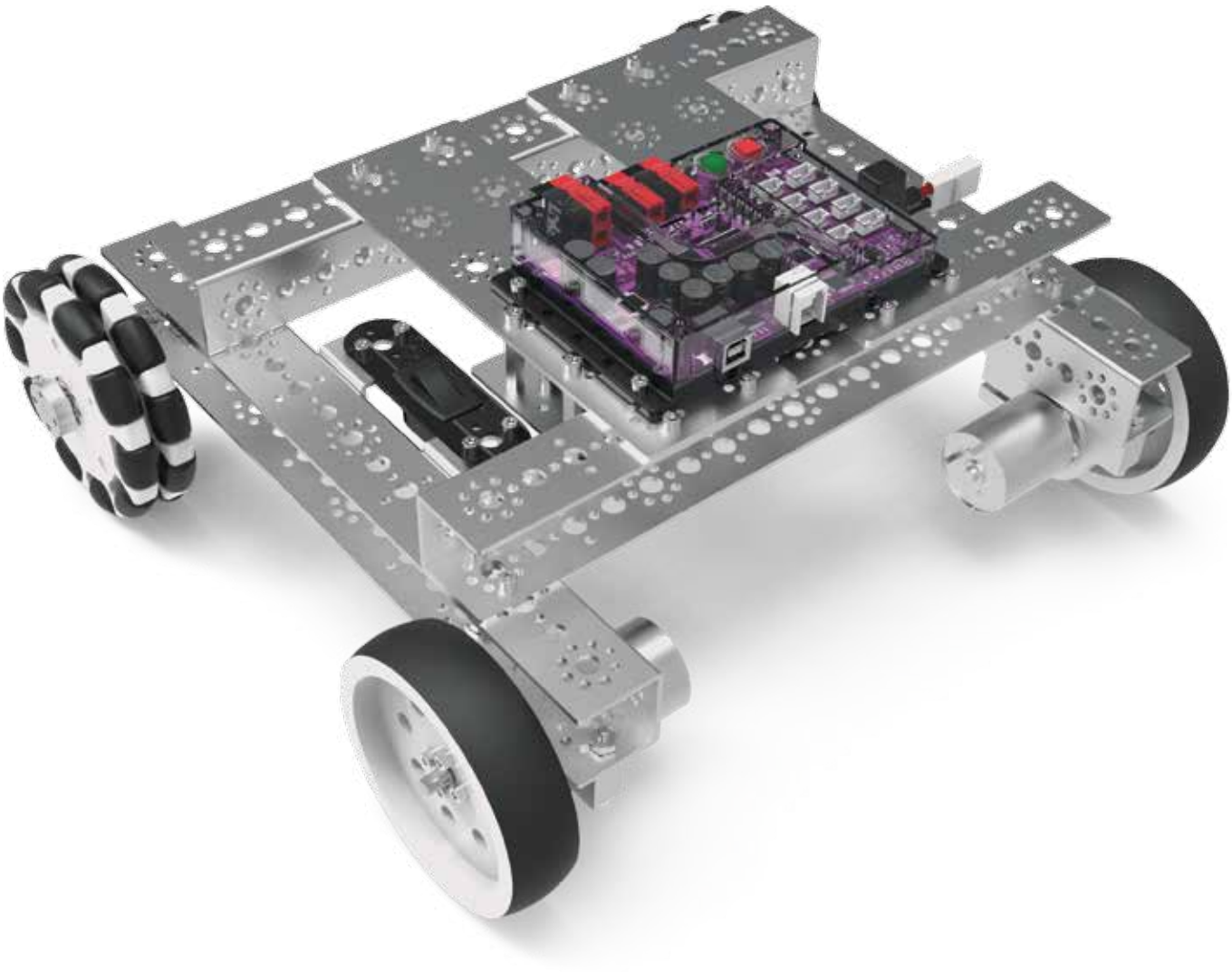
Попробуйте заменить в коде строку prizm.readSonicSensorCM() строкой prizm.readSonicSensorIN(), чтобы выразить расстояние до объекта в дюймах. И обязательно замените в Serial.println вариант "Centimeters" ("Сантиметры") вариантом "Inches" ("Дюймы"), чтобы в окне прибора контроля последовательной передачи данных была указана правильная единица измерения. Поняв, как использовать ультразвуковой датчик, вы наделите своего робота зрением, и он сможет обходить объекты и препятствия.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

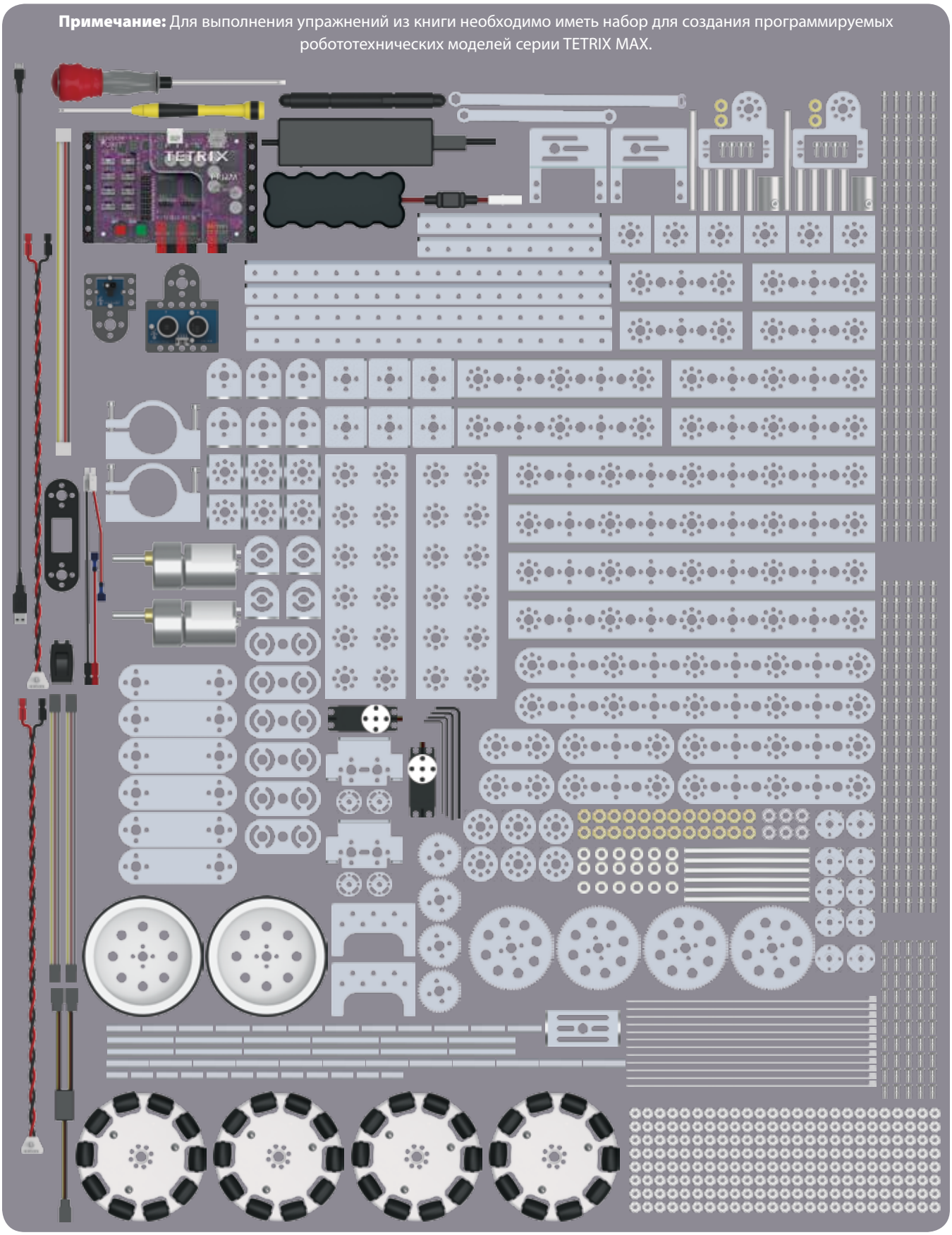
Сборка и кодирование робота-исполнителя серии PRIZM

Именно этого вы и ждали. Пора перейти на следующий уровень. Вы прилежно усваивали основы, а теперь пришло время применить усвоенное в работе с настоящим роботом. Выполняя следующие 10 упражнений, мы узнаем, как построить робота, как научить его простейшим движениям, как управлять его перемещениями, как добавлять датчики и многое другое. А завершающим станет упражнение, объединяющее весь пройденный материал. Вот-вот начнётся самое интересное.

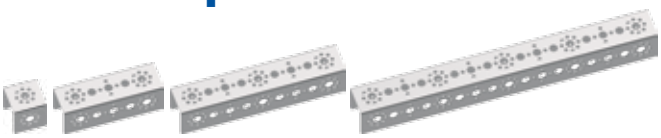


Набор для создания программируемых робототехнических моделей

Примечание: Для выполнения упражнений из книги необходимо иметь набор для создания программируемых робототехнических моделей серии TETRIX MAX.



Указатель деталей из набора для создания программируемых робототехнических моделей серии TETRIX® MAX



Профильные рейки

№ по каталогу	Название детали	Кол-во
39065	Профильная рейка 32 мм серии TETRIX® MAX	6
39066	Профильная рейка 96 мм серии TETRIX® MAX	4
39067	Профильная рейка 160 мм серии TETRIX® MAX	4
39068	Профильная рейка 288 мм серии TETRIX® MAX	4

Пластины, рейки, скобы и уголки

№ по каталогу	Название детали	Кол-во
39070	Плоские рейки 288 мм серии TETRIX® MAX	2
39072	Уголок 144 мм серии TETRIX® MAX	2
39071	Уголок 288 мм серии TETRIX® MAX	2

Пластины и скобы

№ по каталогу	Название детали	Кол-во
39073	Плоские монтажные пластины TETRIX® MAX	2
39061	Плоские скобы серии TETRIX® MAX	6
39062	Г-образные скобы серии TETRIX® MAX	6
39281	Внутренние угловые скобы серии TETRIX® MAX	6
39270	Внутренние П-образные скобы серии TETRIX® MAX	6
41790	Угловая скоба с регулируемым углом крепления серии TETRIX® MAX	4

Планки

№ по каталогу	Название детали	Кол-во
39274	Планка серии TETRIX® MAX 64 мм x 27 мм	2
39273	Планка серии TETRIX® MAX 96 мм x 27 мм	2
39272	Планка серии TETRIX® MAX 160 мм x 27 мм	2
39271	Планка серии TETRIX® MAX 288 мм x 27 мм	2
41791	Плоская скоба TETRIX® MAX с регулируемым углом крепления	6

Оси, ступицы и прокладки

№ по каталогу	Название детали	Кол-во
39079	Ступица для вала электродвигателя серии TETRIX® MAX	2
39172	Ступицы для оси серии TETRIX® MAX	6
39092	Установочные кольца на ось серии TETRIX® MAX	6
39088	Оси 100 мм серии TETRIX® MAX	6
39091	Бронзовые втулки серии TETRIX® MAX	24
39090	Прокладки для ступицы шестерни серии TETRIX® MAX	2
39100	Прокладки для оси 1/8" серии TETRIX® MAX	12
39101	Прокладки для оси 3/8" серии TETRIX® MAX	6
39387	Плоская круглая прокладка серии TETRIX® MAX	6

Шестерни и зубчатые передачи

№ по каталогу	Название детали	Кол-во
39028	Шестерня с 40 зубьями серии TETRIX® MAX	4
39086	Шестерня с 80 зубьями серии TETRIX® MAX	4

Стойки и распорки

№ по каталогу	Название детали	Кол-во
39102	Распорная стойка 6-32 x 1" TETRIX®	12
39103	Распорная стойка 6-32 x 2" TETRIX®	12
39107	Распорные стойки 6-32 x 32 мм TETRIX®	12
41253	Распорные стойки 6-32 x 16 мм TETRIX®	12

Сервоприводы и детали конструктора

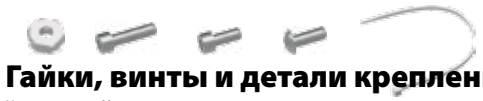
№ по каталогу	Название детали	Кол-во
39060	Одинарный кронштейн для стандартных сервоприводов серии TETRIX® MAX	1
39593	Стандартный поворотный рычаг с подшипником серии TETRIX® MAX	1
39197	Стандартный сервопривод HS-485HB с поворотом вала на 180° градусов	2
39081	Удлинитель для сервоприводов	2
39082	Разветвлённый соединительный кабель для сервоприводов	1
41789	Монтажный комплект для стандартного сервопривода серии TETRIX® MAX	2
39280	Регулируемые кронштейны для сервоприводов серии TETRIX® MAX	2

Электродвигатели пост. тока и детали конструктора

№ по каталогу	Название детали	Кол-во
39089	Монтажная опора для электродвигателя серии TETRIX® MAX	2
39530	Редукторный электродвигатель пост. тока серии TETRIX® MAX	2

Шины и колёса

№ по каталогу	Название детали	Кол-во
39055	Колесо 4" серии TETRIX® MAX	2
36466	Всенаправленное роликовое колесо 4" серии TETRIX® MAX	2

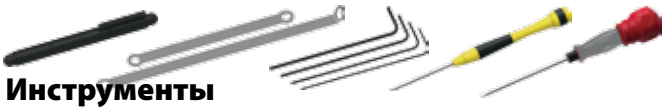


Гайки, винты и детали крепления

№ по каталогу	Название детали	Кол-во
39094	Зубчатые гайки	100
39097	Винты с головкой под торцевой ключ 6-32 x 1/2"	100
39098	Винты с головкой под торцевой ключ 6-32 x 5/16"	100
39111	Винт со сферической головкой 3/8"	50
31902	Стяжки	12

Аккумуляторная батарея и и зарядное устройство

№ по каталогу	Название детали	Кол-во
38009	Держатели для аккумуляторной батареи серии TETRIX® MAX	2
39057	Аккумуляторная батарея, 12 В, 3 000 мА серии TETRIX® MAX	1
39830	Зарядное устройство для аккумуляторной батареи на 12 В серии TETRIX® MAX	1



Инструменты

№ по каталогу	Название детали	Кол-во
36404	Отвёртка 4-в-1	1
38001	Набор гаечных ключей TETRIX®	2
39104	Комплект шестигранных (торцевых) ключей серии TETRIX®	4
40341	Миниатюрная отвёртка с шестигранным шариковым жалом	1
42991	Отвёртка 2-в-1	1

Электроника и органы управления

№ по каталогу	Название детали	Кол-во
43000	Контроллер PRIZM	1
41352	Кабель с разъёмами Powerpoles для электродвигателя TETRIX	2
43169	Переходник для выключателя аккумуляторной батареи к контроллеру PRIZM	1
43056	Комплект датчика линии	1
43055	Комплект ультразвукового датчика	1
40967	3-футовый кабель USB с разъёмами типа A-B	1

Обзор деталей конструктора серии TETRIX MAX

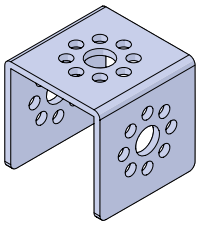
В следующем обзоре механических деталей рассматриваются комплектующие из набора для создания программируемых робототехнических моделей серии TETRIX MAX.

Профильные рейки

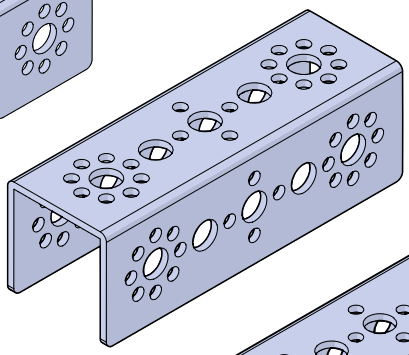
Конструктивные элементы серии TETRIX MAX обозначаются и различаются по длине. Например, профильная рейка 32 мм или плоская рейка 288 мм. Ниже есть дюймовая линейка, предназначенная для измерения распорных стоек и колёс. Для измерения длины деталей используйте сантиметровую линейку, которая есть внизу этого разворота.

Конструктивные элементы

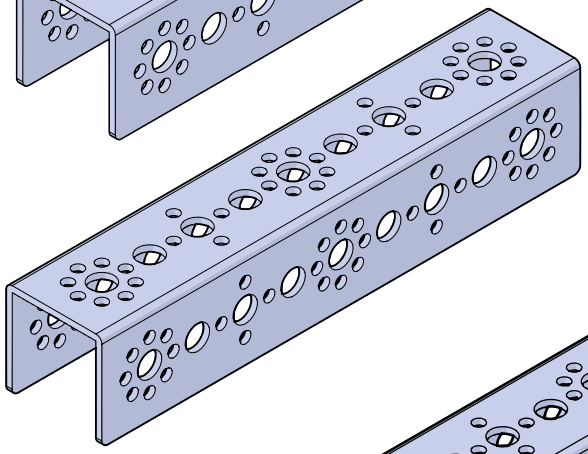
Профильная рейка 32 мм 39065



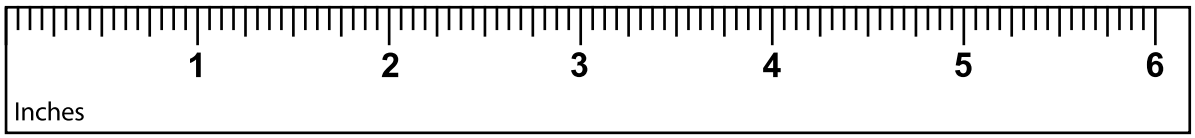
Профильная рейка 96 мм 39066

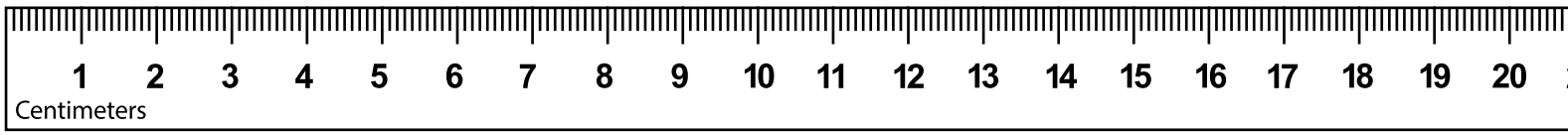


Профильная рейка 160 мм 39067

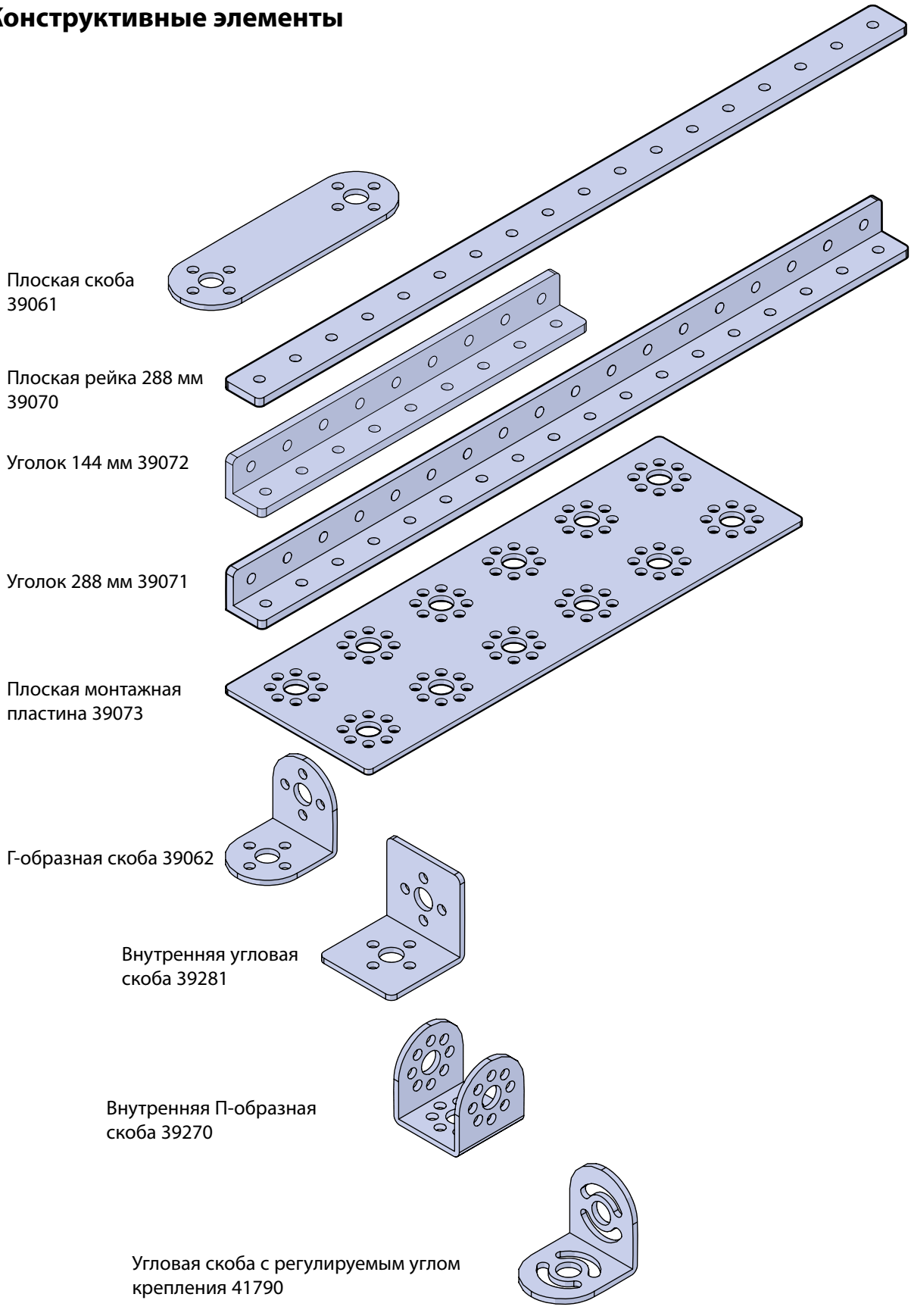


Профильная рейка 288 мм 39068





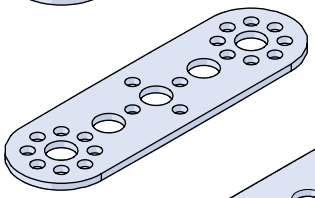
Конструктивные элементы



Конструктивные элементы

Планка 64 мм x 27 мм 39274

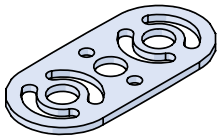
Планка 96 мм x 27 мм 39273



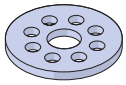
Планка 160 мм x 27 мм 39272

Планка 288 мм x 27 мм 39271

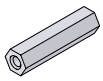
Плоская скоба с регулируемым углом крепления 41791



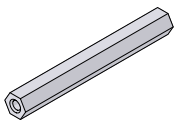
Плоская круглая прокладка 39387



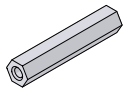
Распорная стойка 6-32 x 1" 39102



Распорная стойка 6-32 x 2" 39103



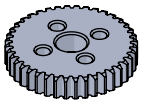
Распорная стойка 6-32 x 32 мм 39107



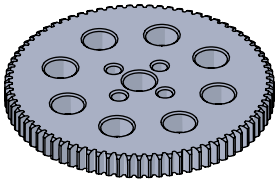
Распорная стойка 6-32 x 16 мм 41253

Элементы механизмов движения

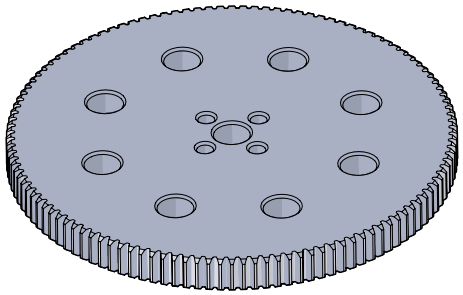
Шестерня с 40 зубьями 39028



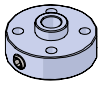
Шестерня с 80 зубьями 39086



Шестерня с 120 зубьями 39085



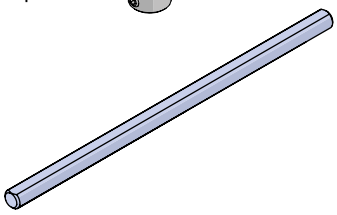
Ступица на ось 39172



Ступица для вала электродвигателя 39079

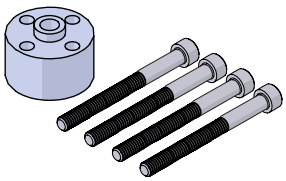
Установочное кольцо на ось 39092

Оси 100 мм 39088



Бронзовая втулка 39091

Прокладка для ступицы шестерни 39090

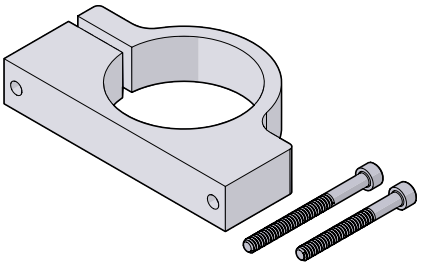


Прокладка для оси 1/8" 39100

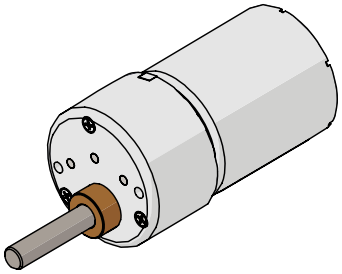
Прокладка для оси 3/8" 39101

Элементы механизмов движения

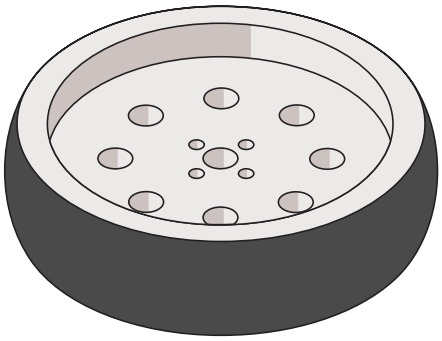
Монтажная опора электродвигателя 39089



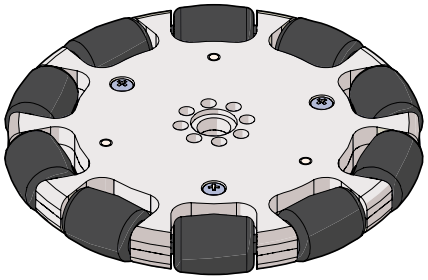
Редукторный электродвигатель постоянного тока 39530



Колесо 4" 39055



Всенаправленное роликовое колесо 4" в комплекте/в сборе 36466



Зубчатая гайка 39094

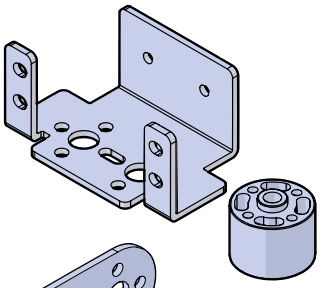
Винт с головкой под торцевой ключ 6-32 x 1/2" 39097

Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

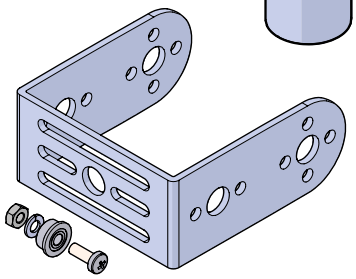
Винт со сферической головкой 3/8" 39111

Элементы механизмов движения

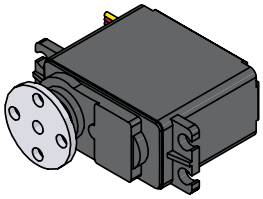
Одинарный кронштейн для стандартных сервоприводов 39060



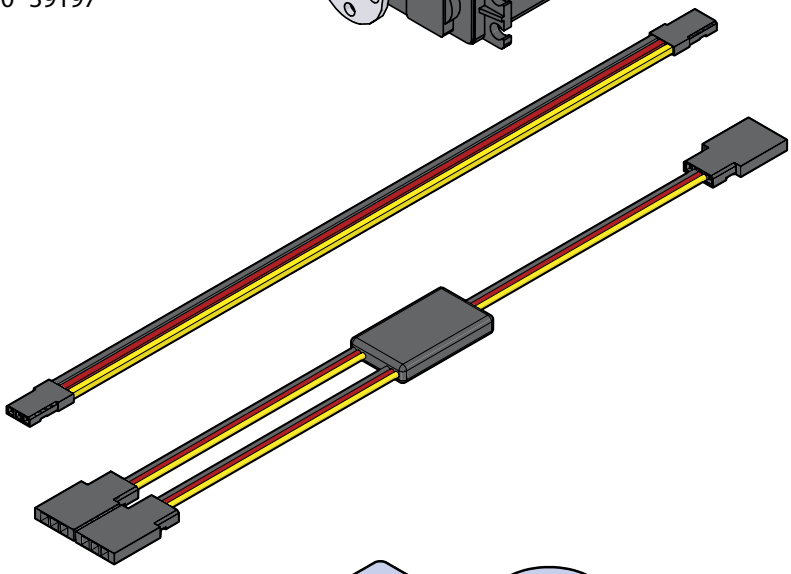
Стандартный поворотный рычаг с подшипником 39593



Стандартный сервопривод HS-485HB с поворотом вала на 180° 39197

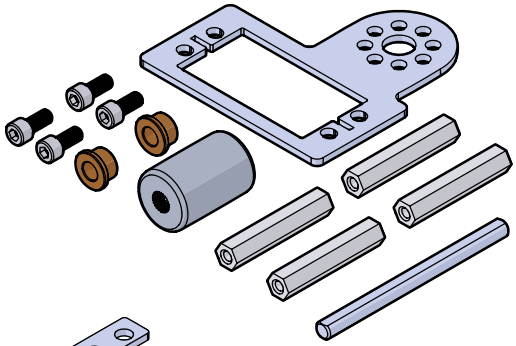


Удлинитель для сервоприводов 39081

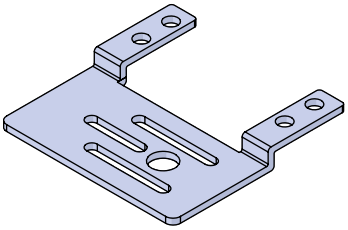


Разветвлённый соединительный кабель для сервоприводов 39082

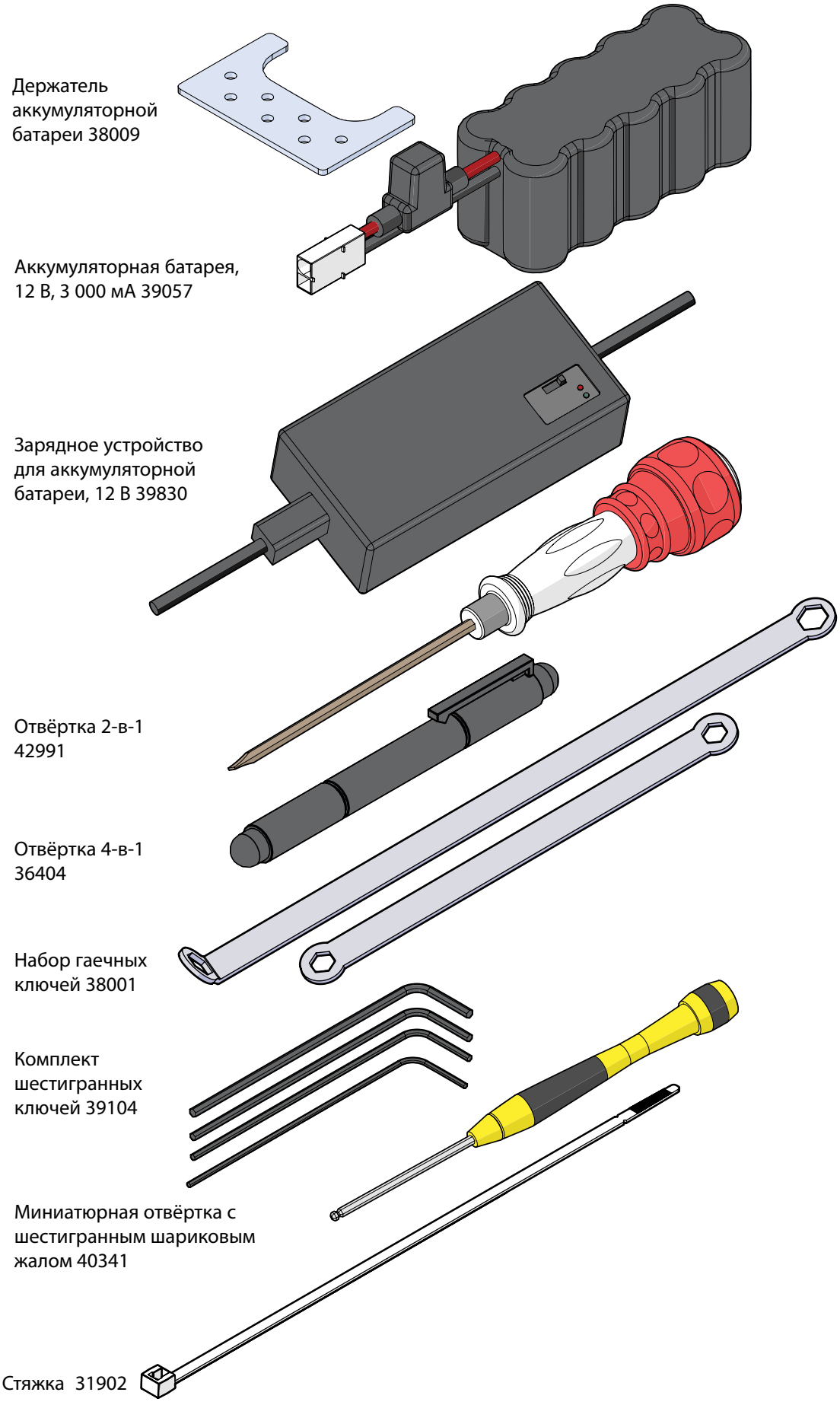
Монтажный комплект для стандартных сервоприводов 41789



Регулируемый кронштейн для сервопривода 39280

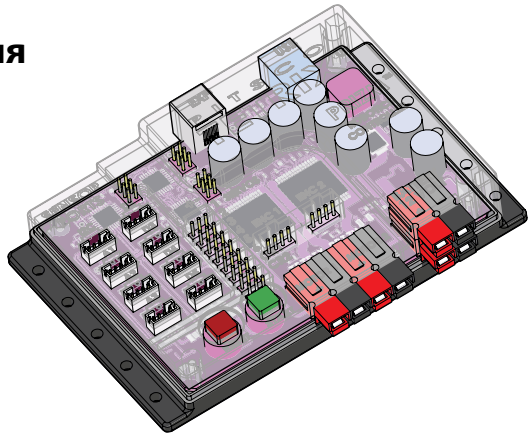


Электропитание, инструменты и принадлежности

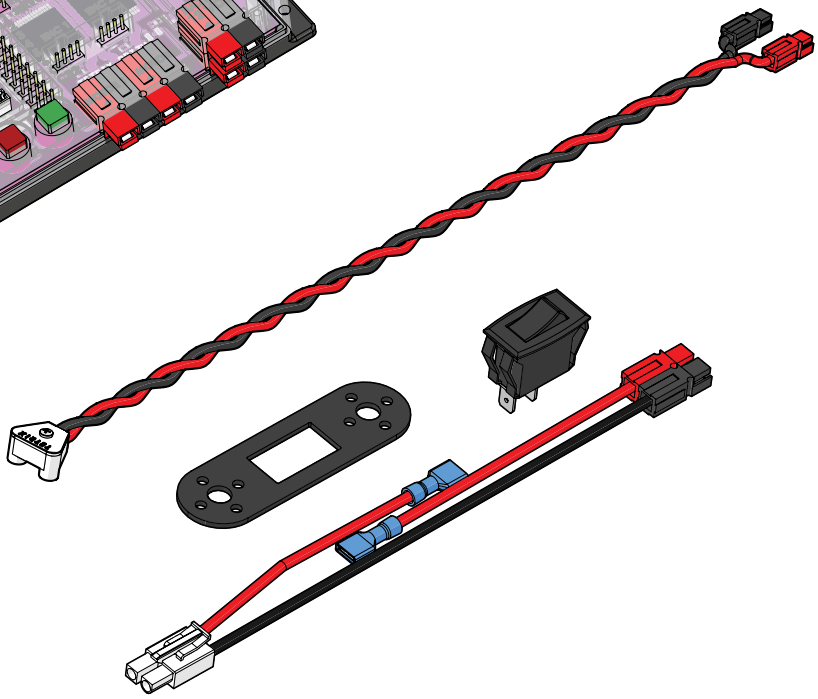


Органы управления

Контроллер PRIZM™
43000

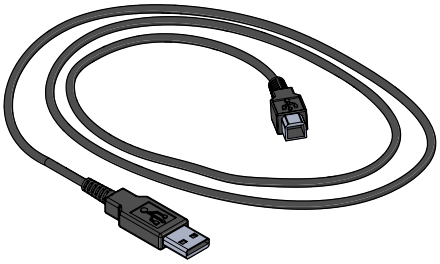


Кабель с разъёмами Powerpoles для электродвигателя TETRIX 41352

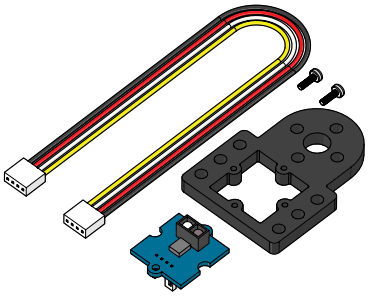


Переходник для выключателя аккумуляторной батареи контроллера PRIZM 43169

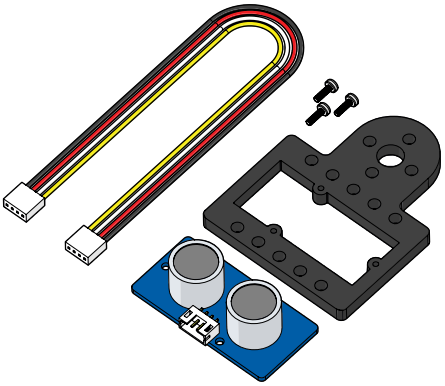
3-футовый кабель USB с вилками типа A-B
40967



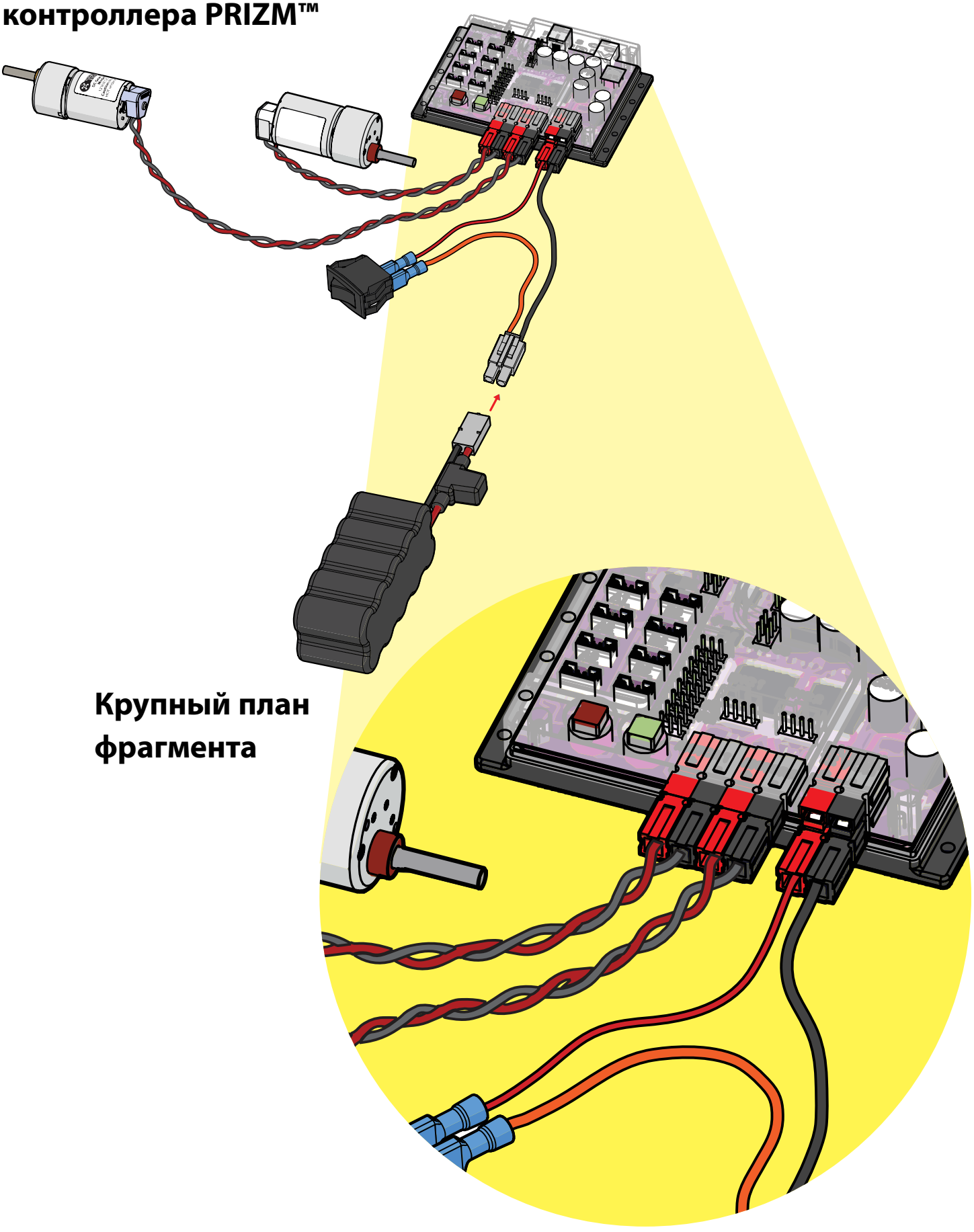
Комплект датчика линии
43056



Комплект ультразвукового датчика 43055



Наглядное изображение проводных соединений контроллера PRIZM™



Крупный план фрагмента

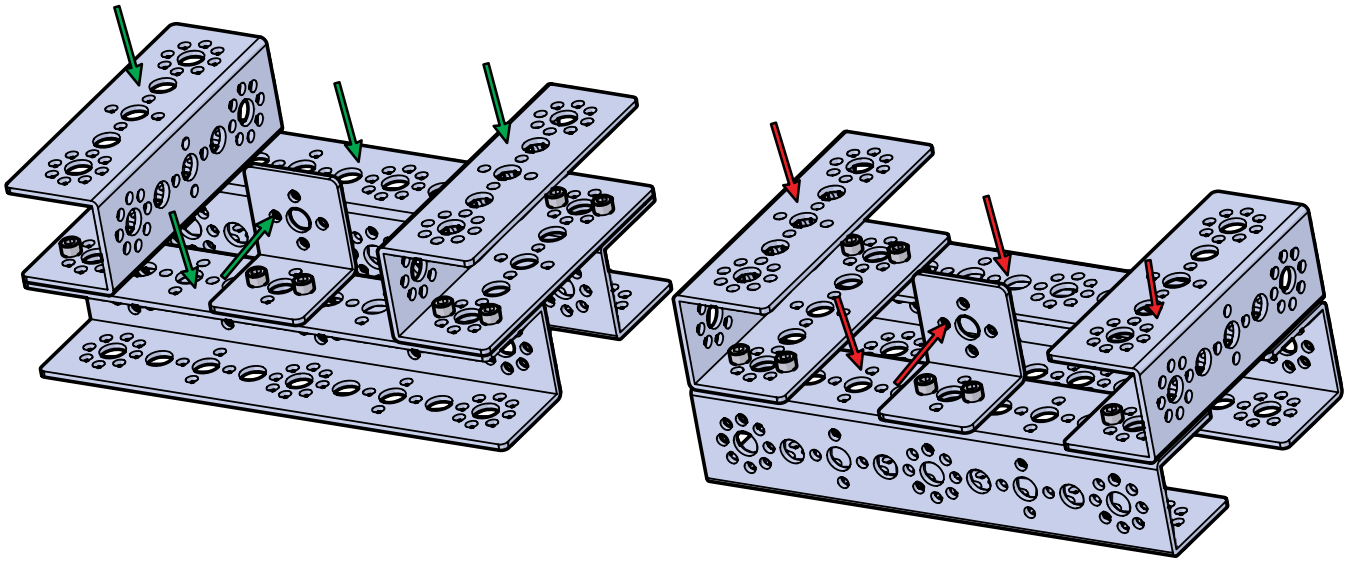
Советы по работе с руководством по сборке/с конструктором TETRIX MAX

Продуманный, удобный доступ к крепежу улучшит впечатление от сборки у всех её участников.

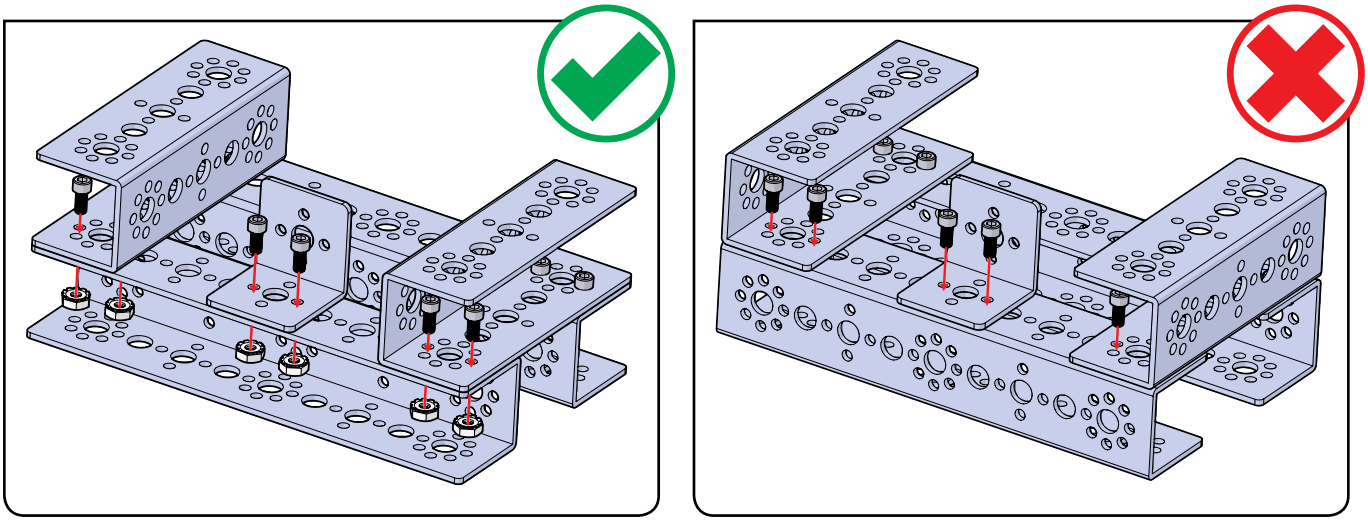
Обычно при сборке любой промежуточной конструкции полезно лишь наживить гайки и винты, пока вы не убедитесь, что все детали встали, как надо. После этого можно затянуть гайки и винты, и переходить к следующему шагу.

1. Установка профильных реек

Если потратить немного времени на планирование и заранее продумать, как совмещать детали конструктора, то сборка пойдёт легче, быстрее и с большей пользой.

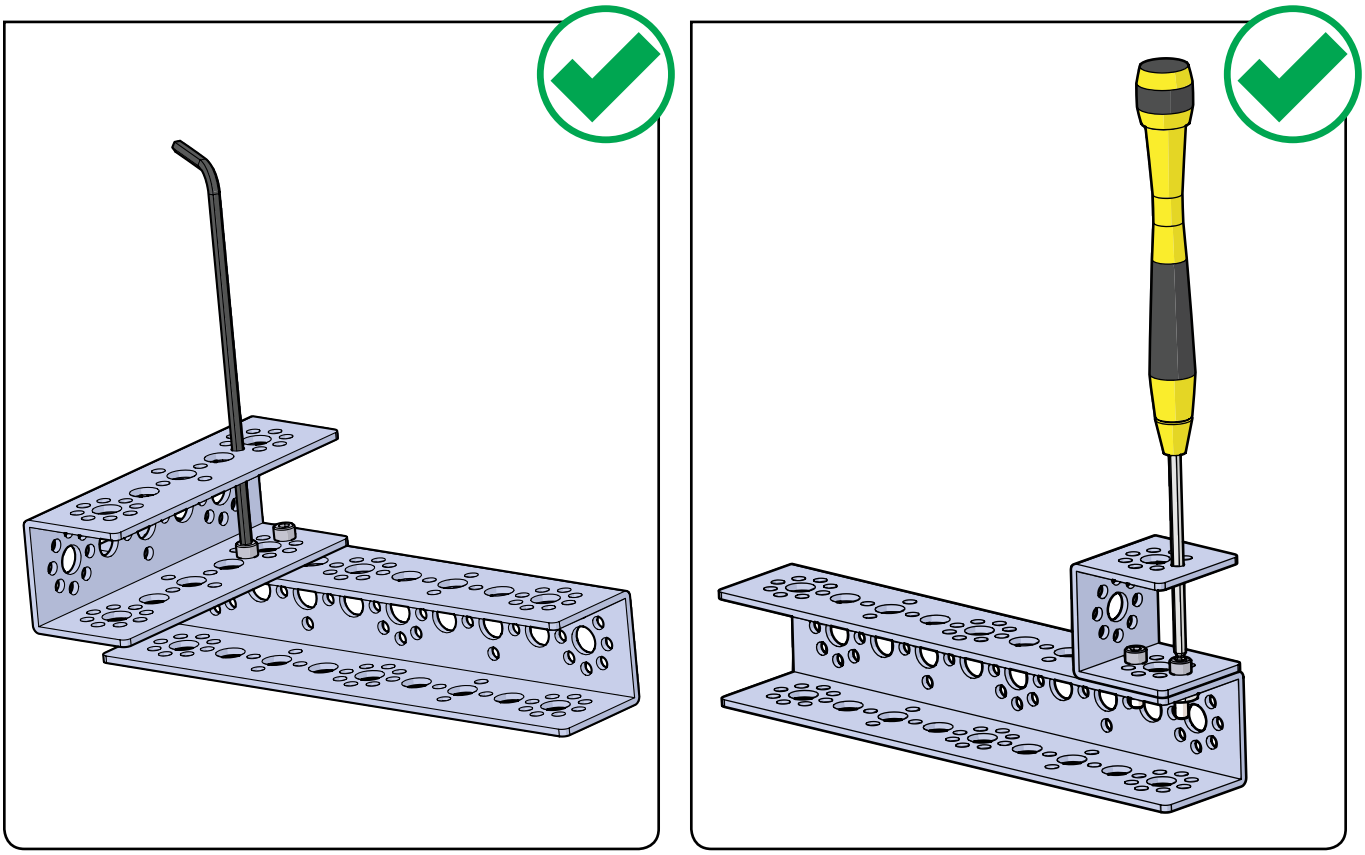


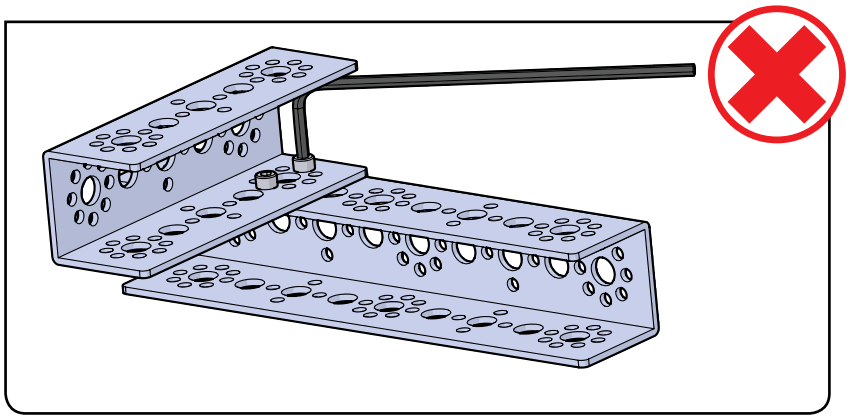
На рисунке вверху у обеих конструкций одинаковые сборочные поверхности, но соединять зубчатые гайки с винтами на одной удобно, а на другой — нет. Конструкция с удобной установкой крепежа предпочтительнее. По возможности не затрудняйте себе установку деталей и крепежа.



2. Использование инструментов

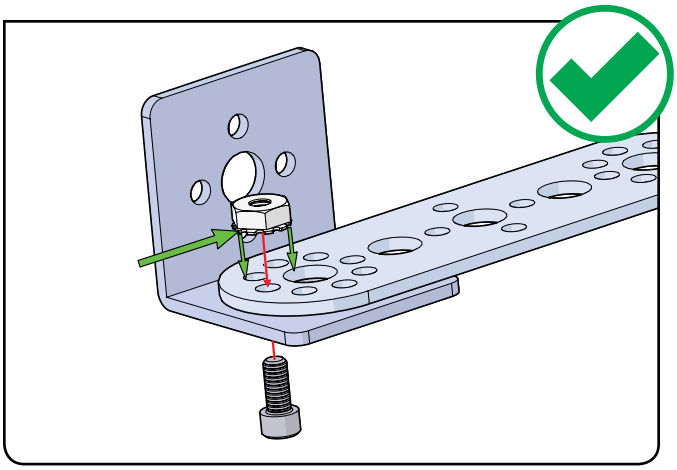
Правильное использование простейших инструментов делает сборку более плавной и приятной, а также ускоряет её.

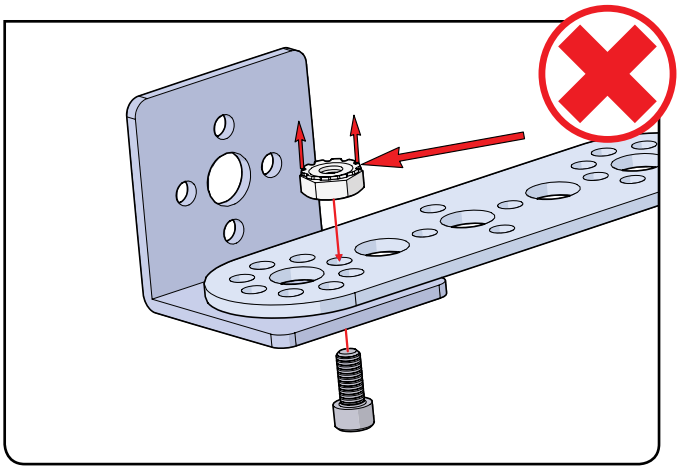




3. Используйте особенности конструкции

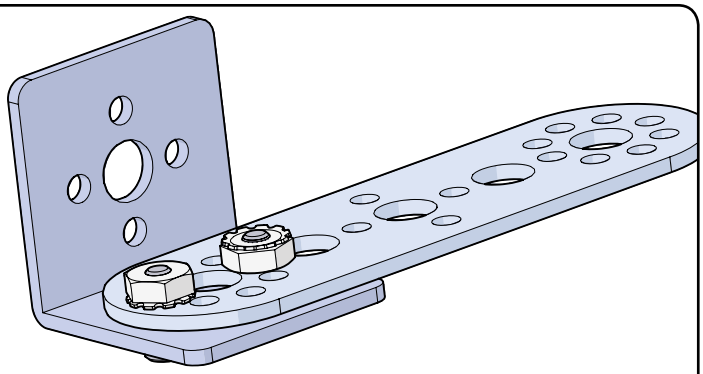
Особенности конструкции некоторых деталей улучшают их или преследуют определённую цель. Знание этих особенностей позволит вам в полной мере использовать их, отчего собранные вами конструкции станут крепче, будут служить дольше и работать лучше.





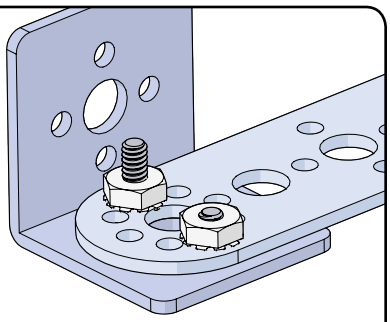
Вот каким должно быть положение зубчатых гаек. Стопорящие зубцы зубчатой гайки всегда должны находиться со стороны плоской поверхности конструктивного элемента, как в примере слева. Изображение справа — неправильное!

Ниже для сравнения представлены оба вида.



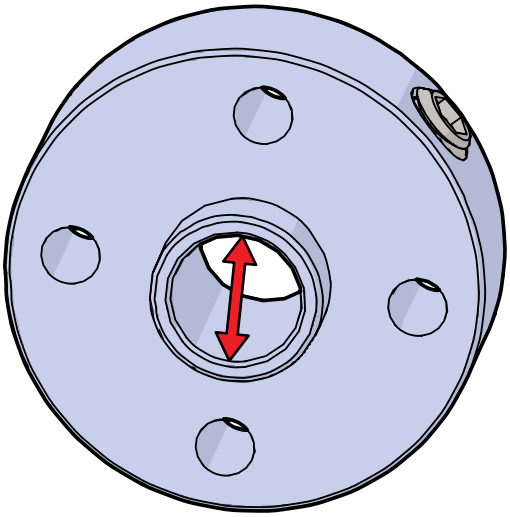
Положение зубчатой гайки слева — правильное.
Положение зубчатой гайки справа — неправильное.

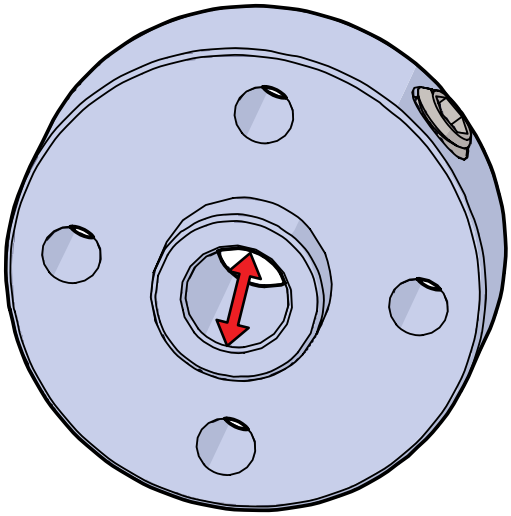
Использование винта соответствующей задаче длины будет самым разумным способом распорядиться имеющимися в вашем распоряжении запасами крепежа. Здесь лучше использовать самый короткий винт, а винты подлиннее оставьте — они понадобятся для другого.



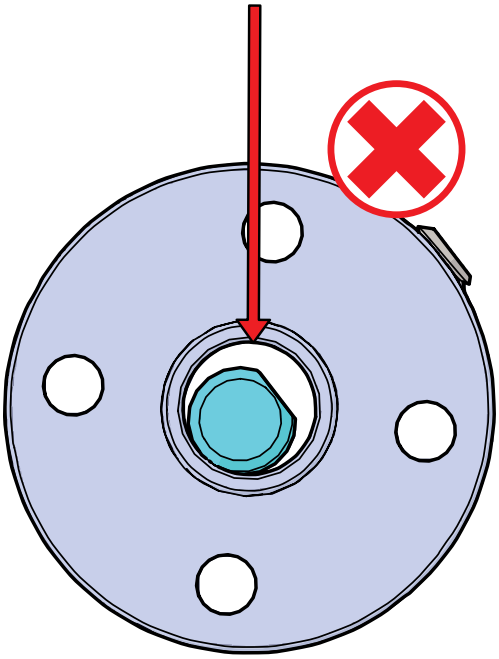
Хотя для конструкции, изображённой выше, подойдёт любой винт, разумнее использовать из имеющегося запаса тот, что справа.

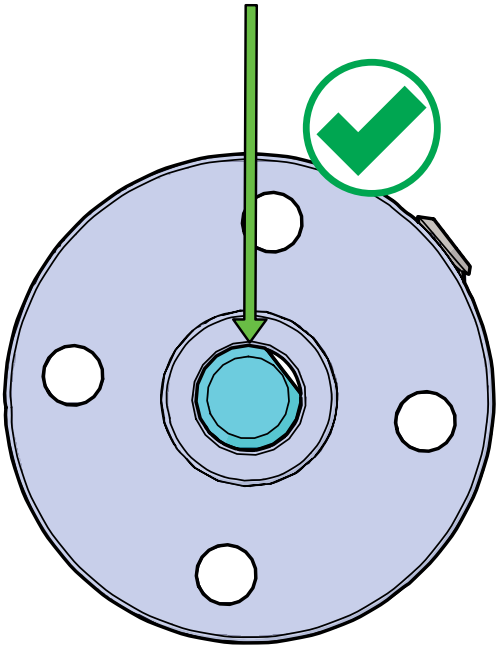
Ступицы для осей и ступицы для валов электродвигателей нередко путают, — ведь у них одинаковый вид и то же назначение, но применяются они в разных условиях.





Эти две детали отличаются внутренним диаметром центрального отверстия. У ступицы для оси он подходит к оси, а у ступицы для вала электродвигателя — к валу электродвигателя.



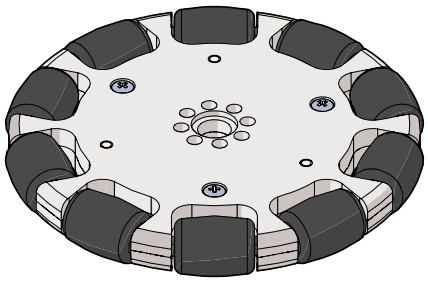


Из-за особого диаметра ступицу для оси можно использовать только с осями, а вот ступицу для вала электродвигателя можно по ошибке надеть на обычную ось, из-за чего потом возможны сбои в работе и путаница при установке колёс или шестерён. Слева изображена ступица для вала электродвигателя, надетая на обычную ось. Обратите внимание на несовпадение наружного диаметра оси и внутреннего диаметра ступицы для вала электродвигателя. Справа изображена ступица для оси, надетая на обычную ось. Обратите внимание, что детали совпадают по размеру и будут работать исправно.

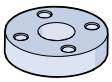
4. Сборка деталей особого назначения

Сборка всенаправленных роликовых колёс —

Необходимые детали и принадлежности



2x
Всенаправленное роликовое колесо 4" в комплекте/в сборе 36466

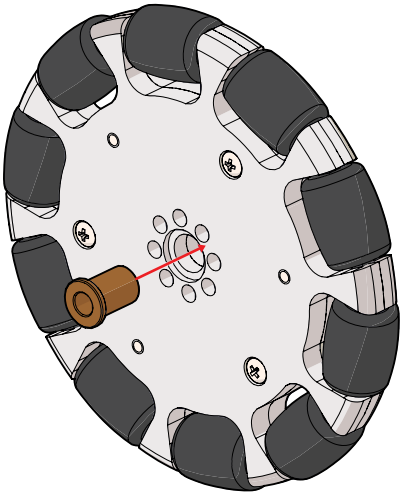


1x
Всенаправленное роликовое колесо Прокладка 36466

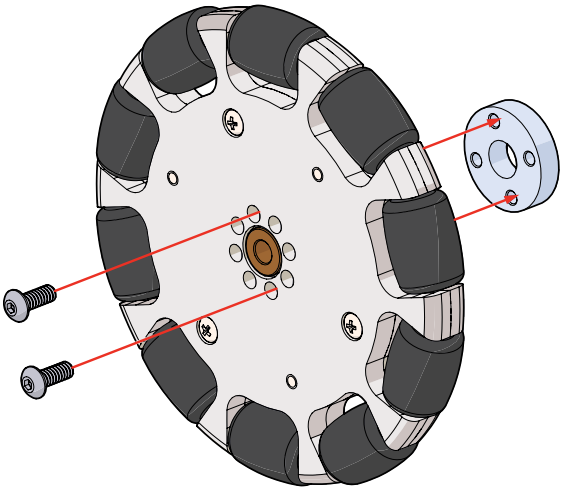
2x
Бронзовые втулки 39091

4x
Винт со сферической головкой 3/8" 39111

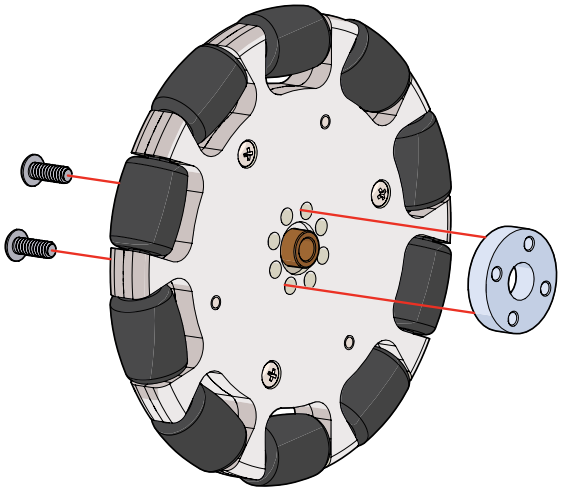
Шаг 1.0



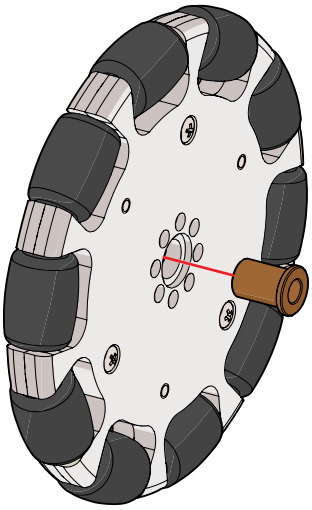
Шаг 1.1 (вид спереди)



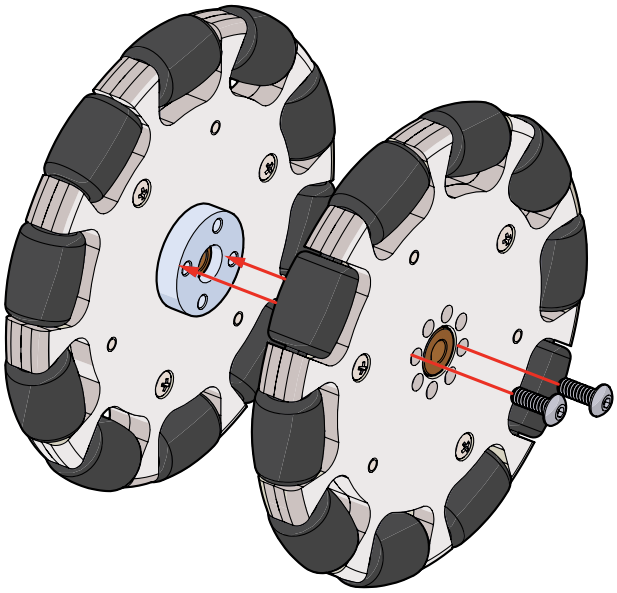
Шаг 1.1 (вид сзади)

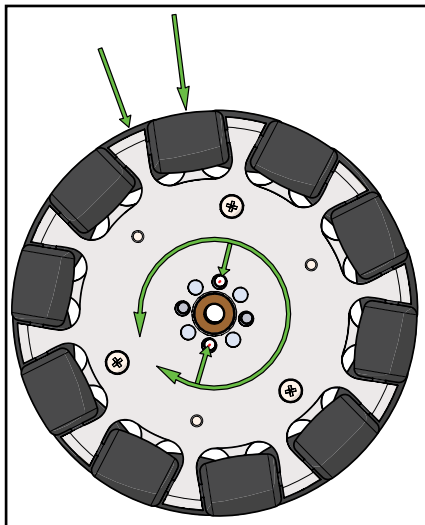


Шаг 1.2

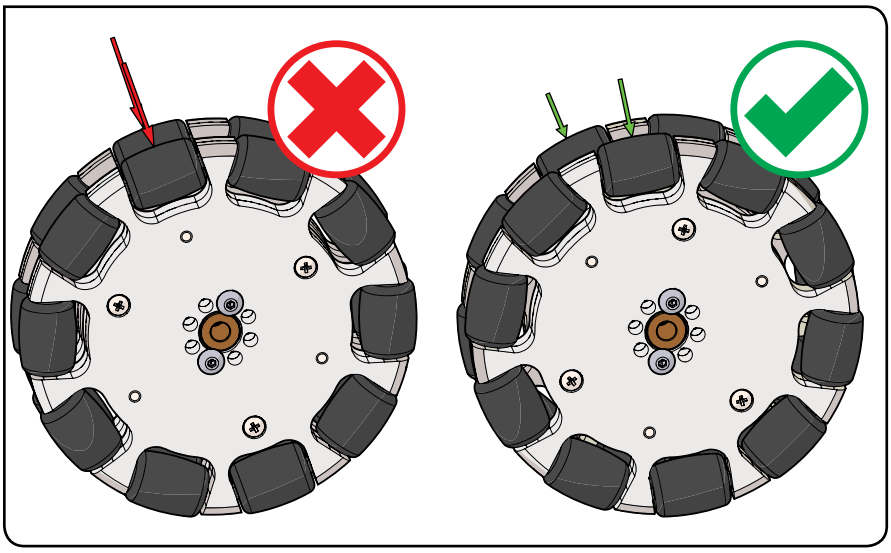


Шаг 1.3





Подсказка: Чтобы получить нужный зазор между роликами обоих колёс, снимите винты (шаг 1.3) и поверните показанное колесо в любом направлении. Установите винты на место.



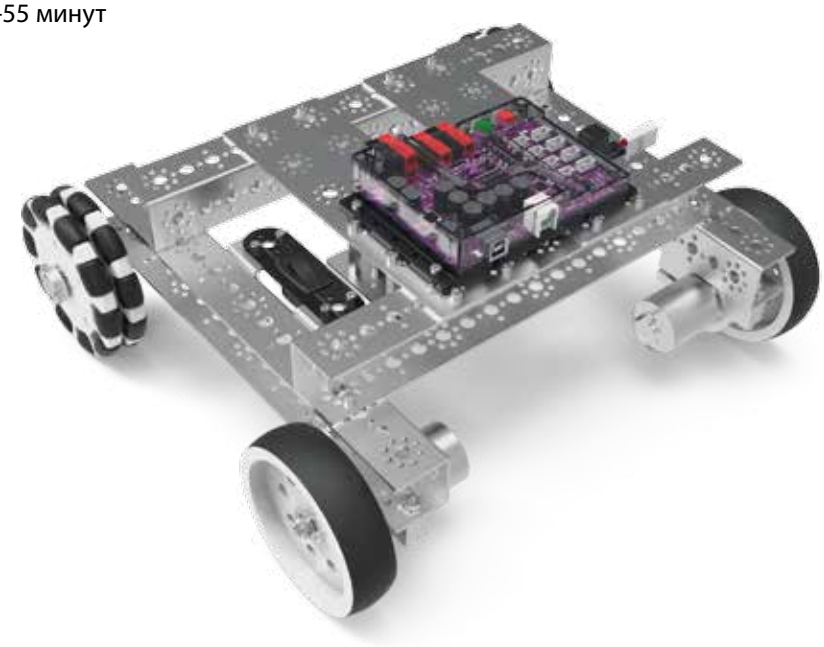
Упражнение 6: Сборка робота-исполнителя

Нам нужен робот. Робот нужен несложный, поскольку цель руководства — освоение работы с контроллером PRIZM и *ПО Arduino (IDE)*. Исходя из сказанного мы придумали конструкцию робота-исполнителя PRIZM. Робот-исполнитель должен быть простым, лёгким в сборке и точно соответствовать целям, поставленным в этом руководстве, без каких-либо ненужных частей. Но всё, чему вы научитесь при помощи этого простейшего робота, можно применить в работе с более сложной робототехнической моделью в ходе освоения премудростей робототехники.

Это хороший пример робота-исполнителя с дифференциальным приводом: в нём используется 2 электродвигателя на 12 В, которые установлены тыльными сторонами друг к другу. Сочетание легкоуправляемого привода и двух всенаправленных роликовых колёс в сборе на противоположных концах ходовой части дают идеальное транспортное средство, на котором можно испытать возможности контроллера PRIZM. Начнём с простейшего привода открытого исполнения (рамы с приводом), а в последующих упражнениях будем добавлять к нему новые детали.

Такая конструкция представляется нам идеальной для учащихся со слабым или нулевым представлением о том, как работать с металлическими конструкторами.

Предположительная длительность сборки
45-55 минут



Примечания для учителя: Длительность сборки зависит от множества обстоятельств, в том числе таких вещей, как организованность деталей конструктора, а также наличие или отсутствие у сборщика напарника. Выше указано предполагаемое время, которое требуется усреднённому сборщику с усреднённым опытом, привыкшему собирать поделки из хорошо организованного, полнокомплектного конструктора. На деле длительность может быть разной.

Связь с настоящей жизнью

От конструкции машины зависит её полезность и долговечность. Инженеры проектируют машины (в том числе роботов) для строго определённых целей. Так, бульдозер строят из очень прочных и износостойких материалов, чтобы он выдерживал нагрузки, связанные с валкой деревьев и сносом строений или рытьём больших ям в земле.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Устройство и назначение

Технология

- Материалы
- Крепёж

Проектирование

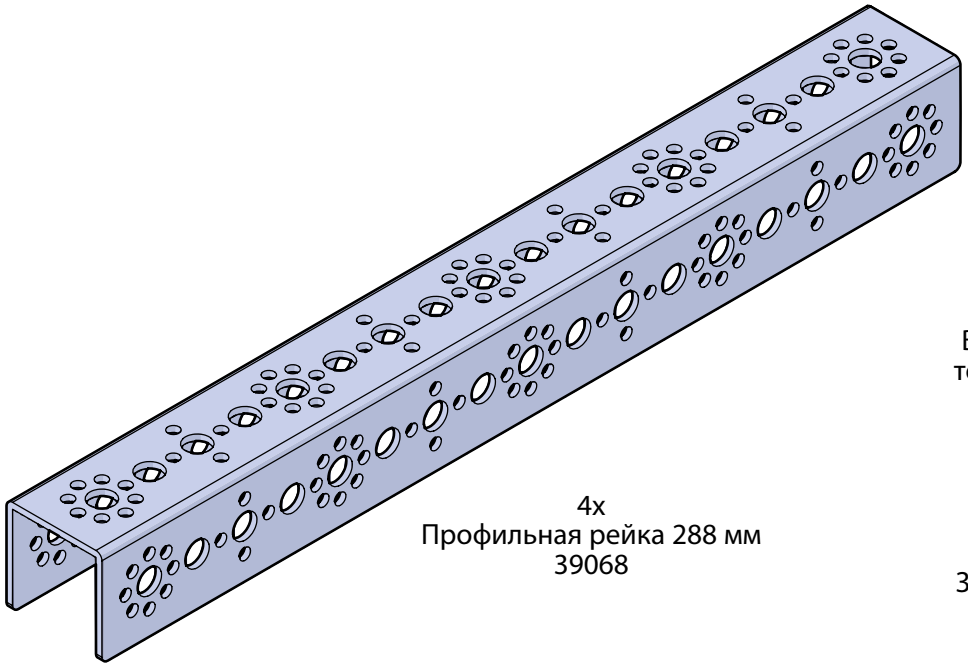
- Конструирование машин

Математика

- Измерение
- Модуль

Шаг 1

Необходимые детали и принадлежности



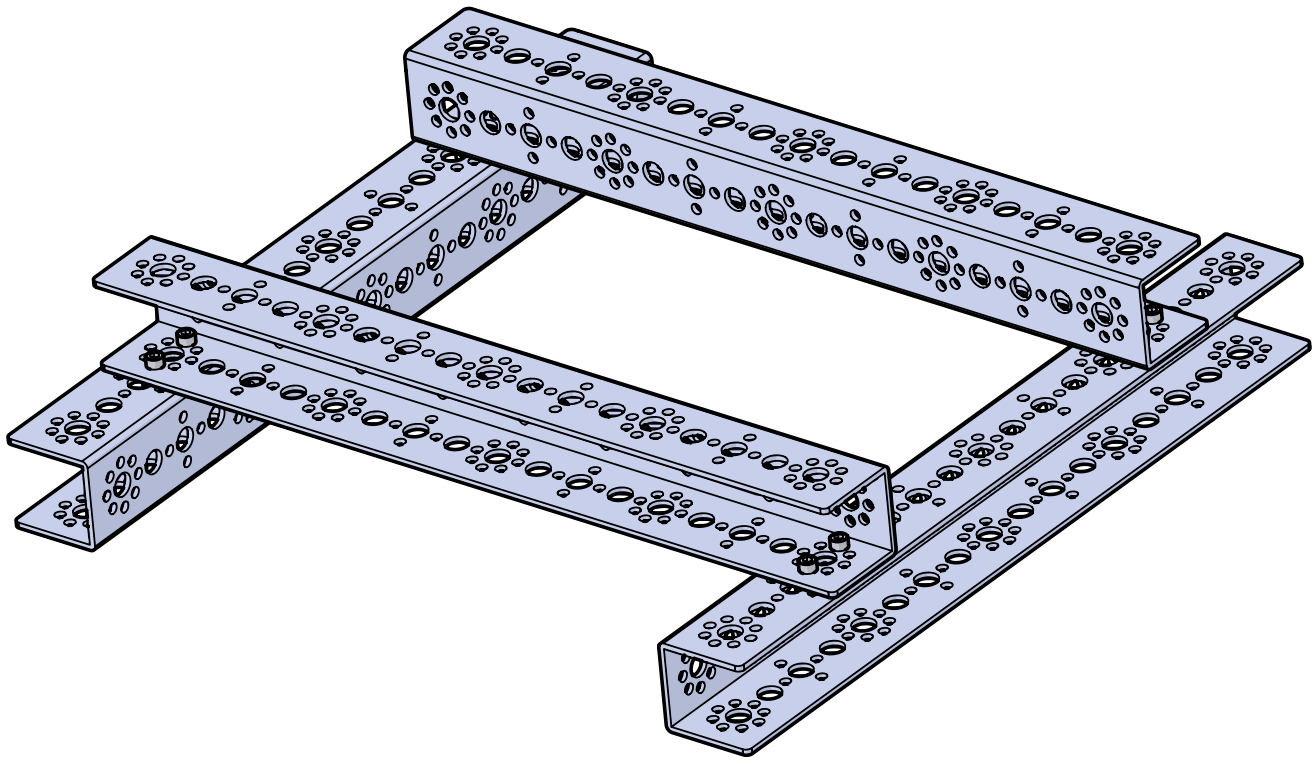
4x
Профильная рейка 288 мм
39068

8x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

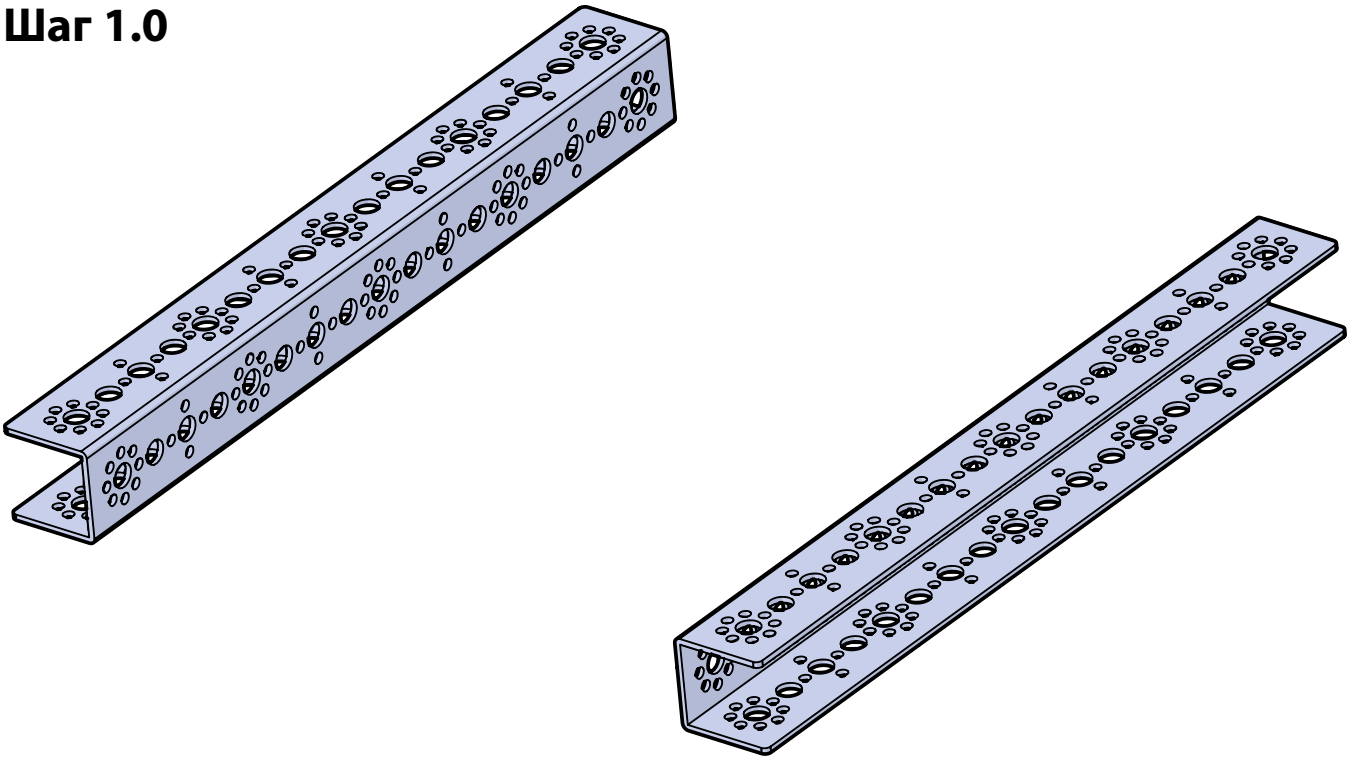
8x
Зубчатая гайка 39094

Подсказка: Чтобы узнать, какие профильные рейки мы выпускаем, см. с. 36. Запомните, мы различаем их по длине.

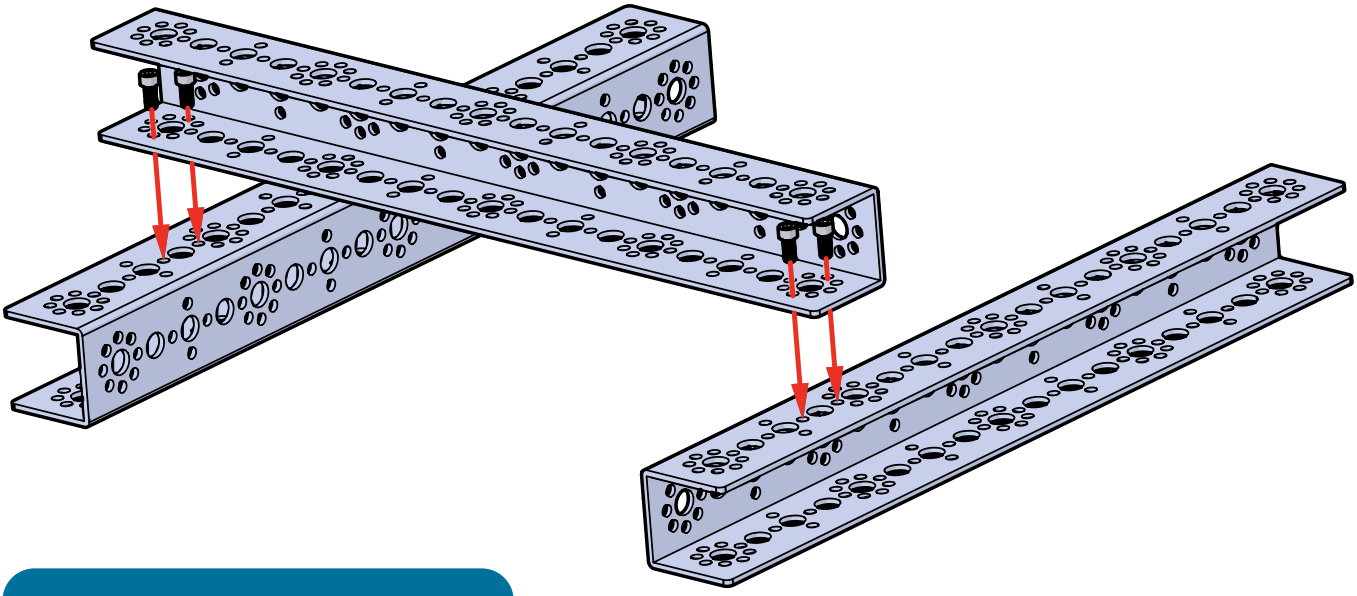
Частично собранная конструкция должна выглядеть так.



Шаг 1.0

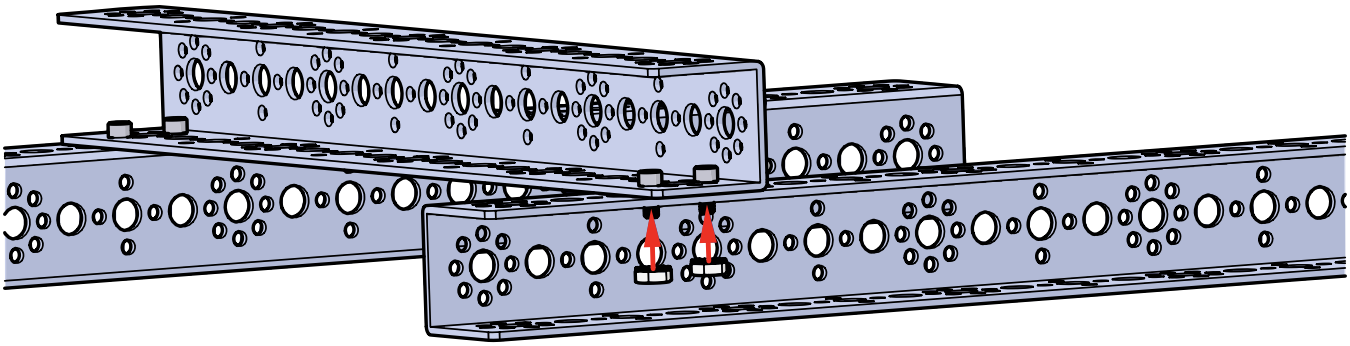


Шаг 1.1

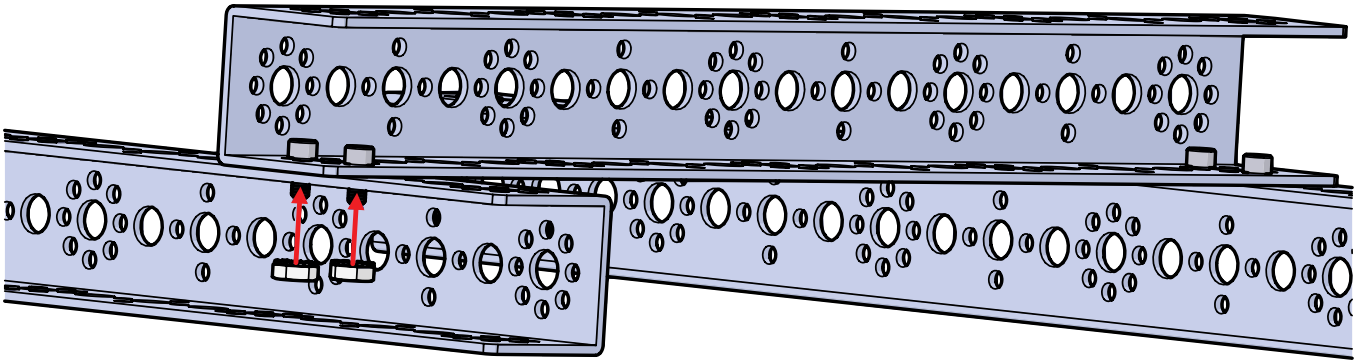


Подсказка: Достаточно лишь немного завернуть гайки и винты, пока не будут соединены все четыре профильные рейки.

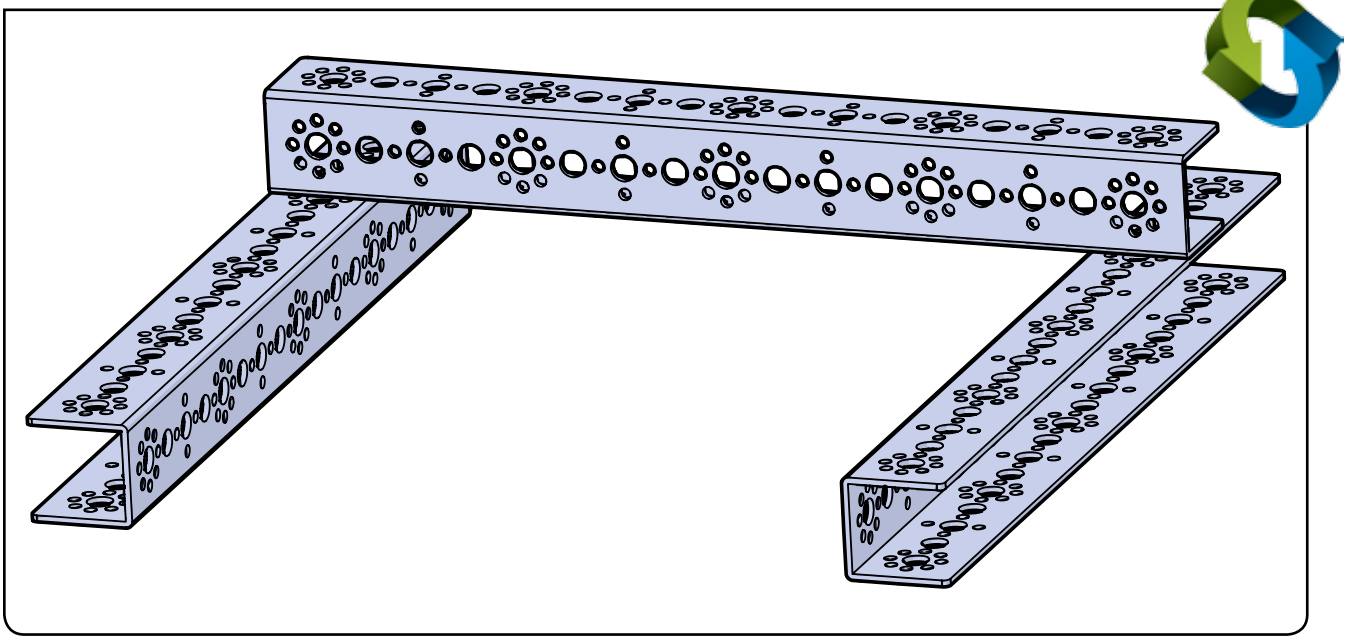
Шаг 1.2



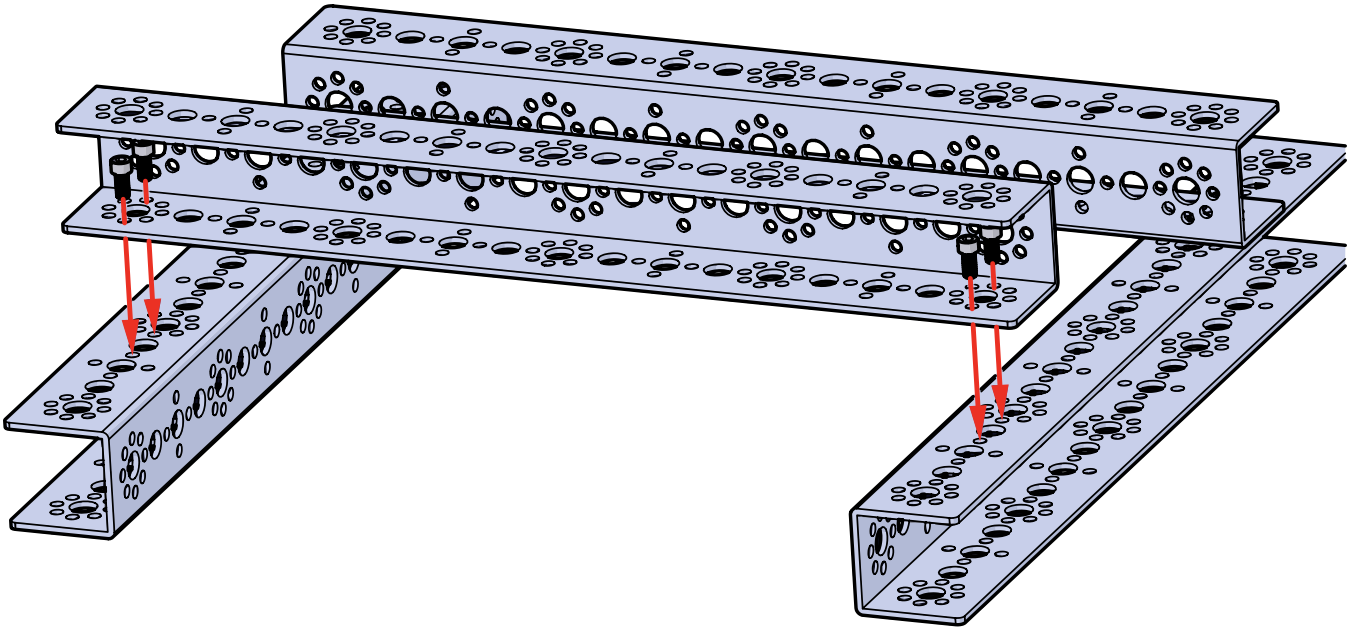
Шаг 1.3



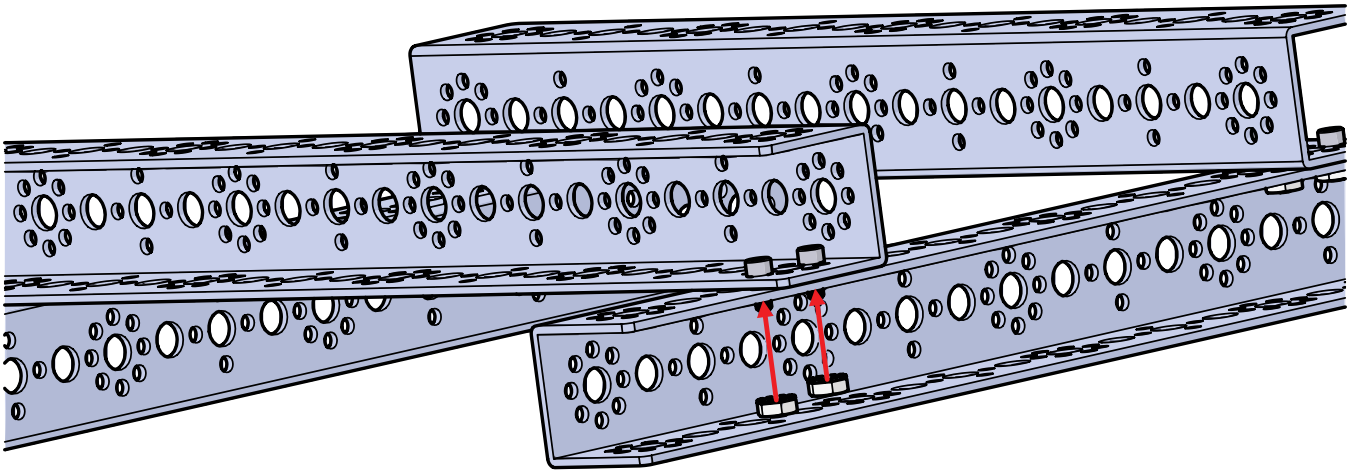
Поверните собранную конструкцию, чтобы сличить её с изображением.



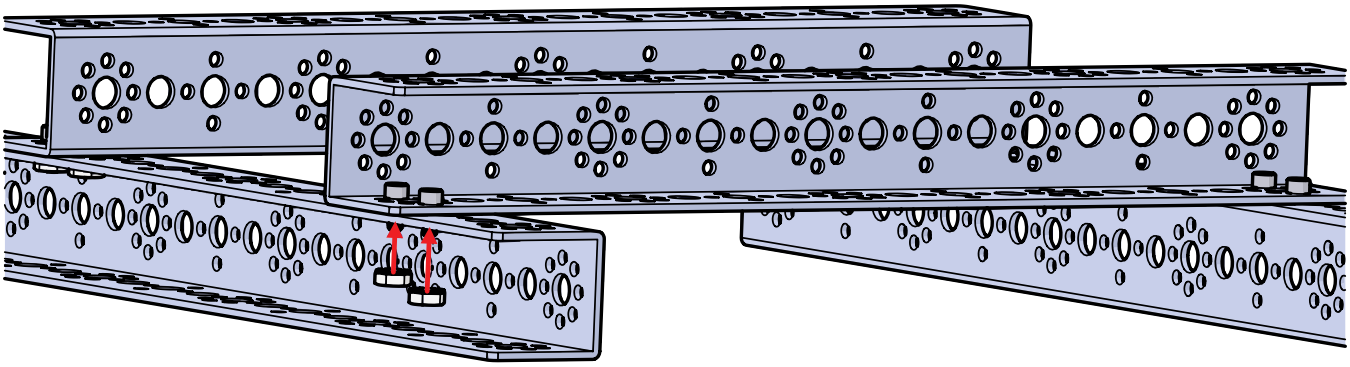
Шаг 1.4



Шаг 1.5



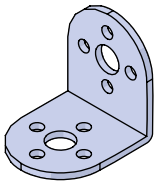
Шаг 1.6



Подсказка: Соединив все четыре профильные рейки и получив прямоугольную раму, не забудьте затянуть все винты и гайки.

Шаг 2

Необходимые детали и принадлежности

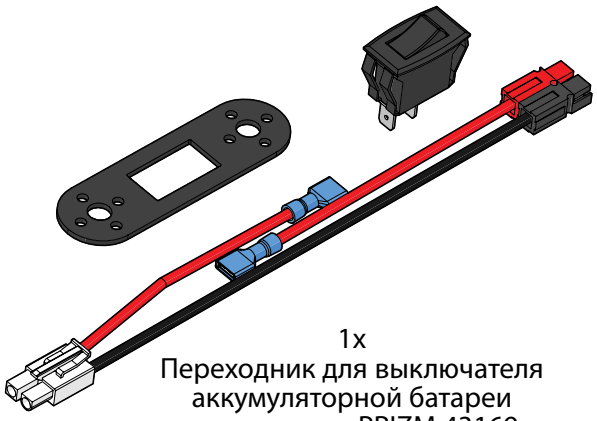


2x
Г-образная скоба
39062

8x
Зубчатая гайка
39094

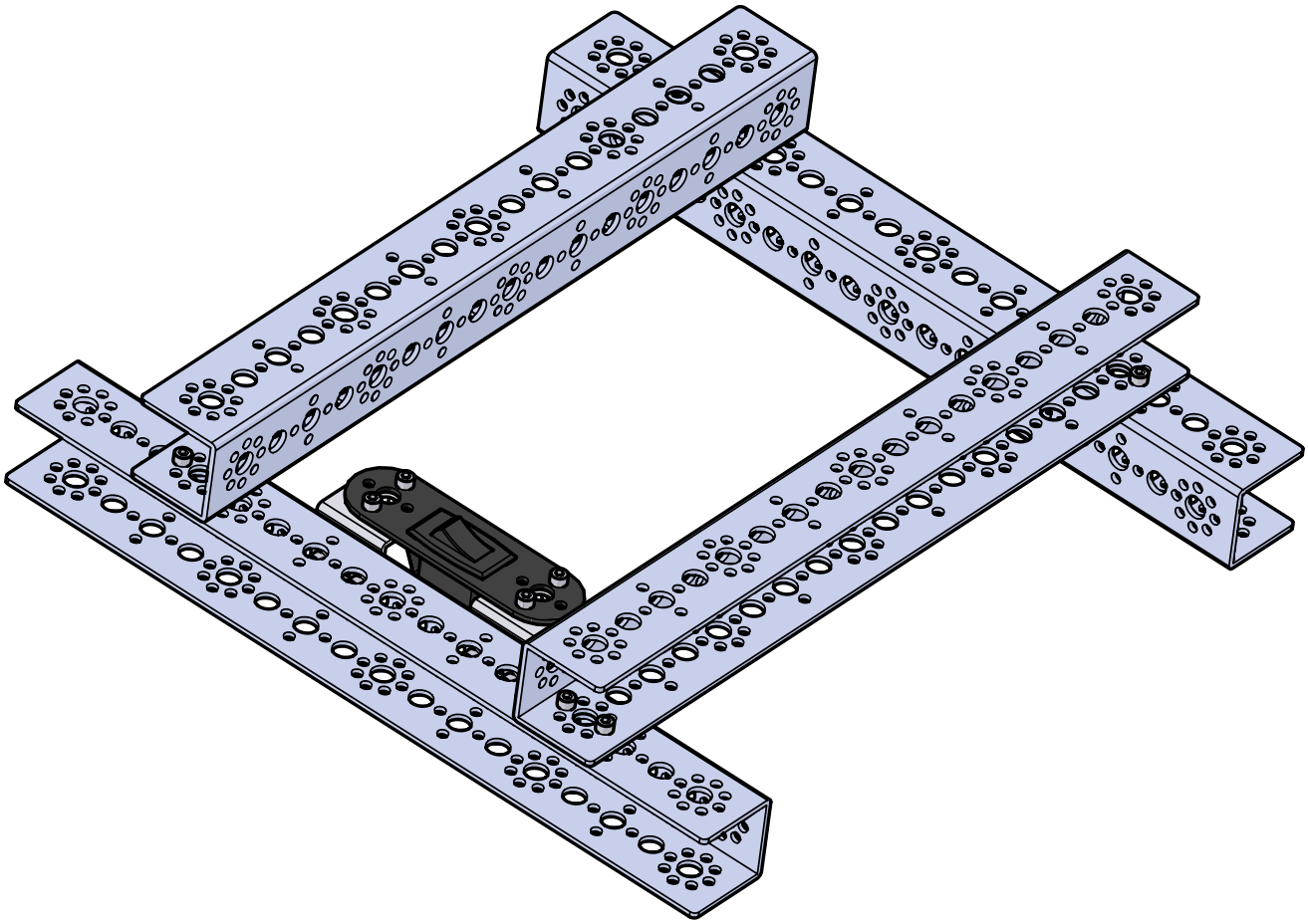
4x
Винт с головкой под торцевой ключ 6-32 x 1/2" 39097

4x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

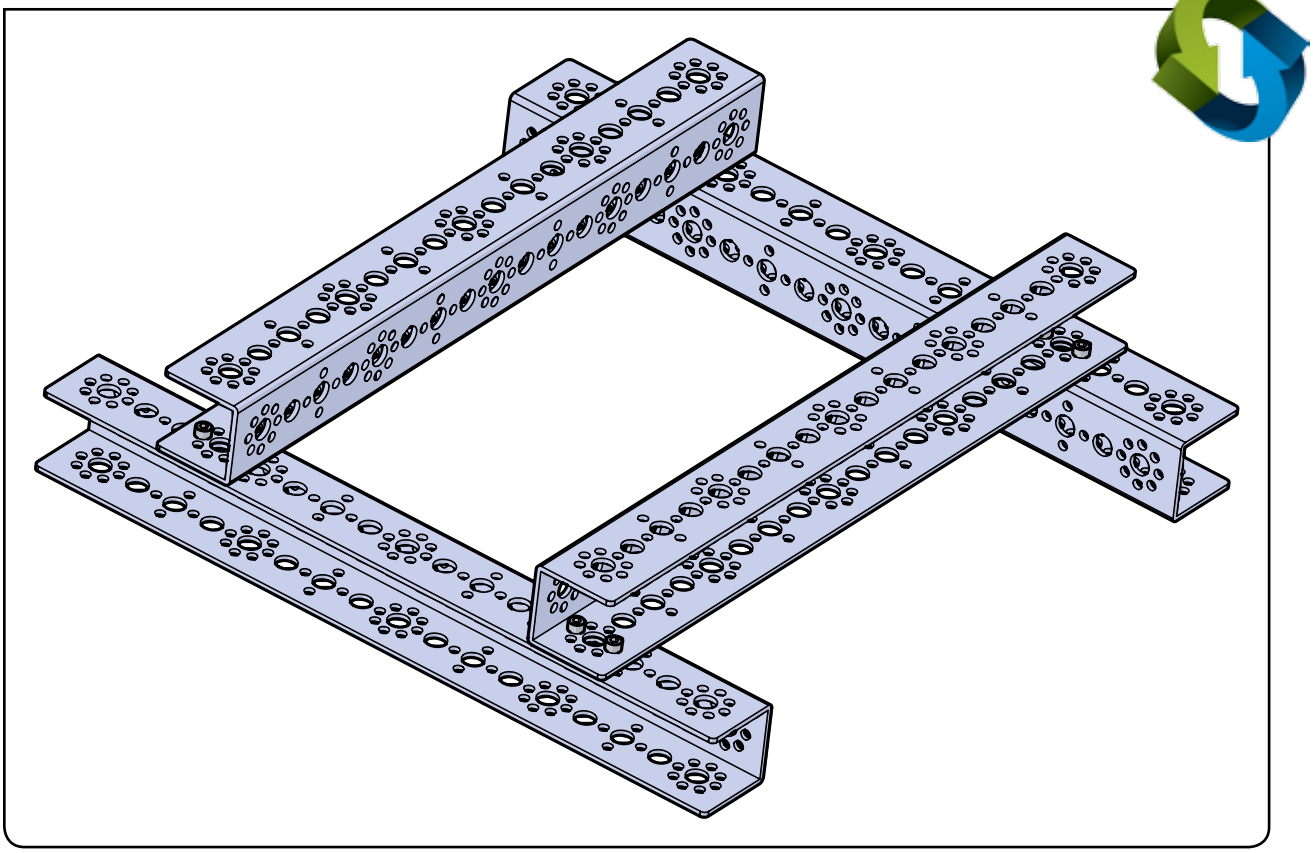


1x
Переходник для выключателя аккумуляторной батареи контроллера PRIZM 43169

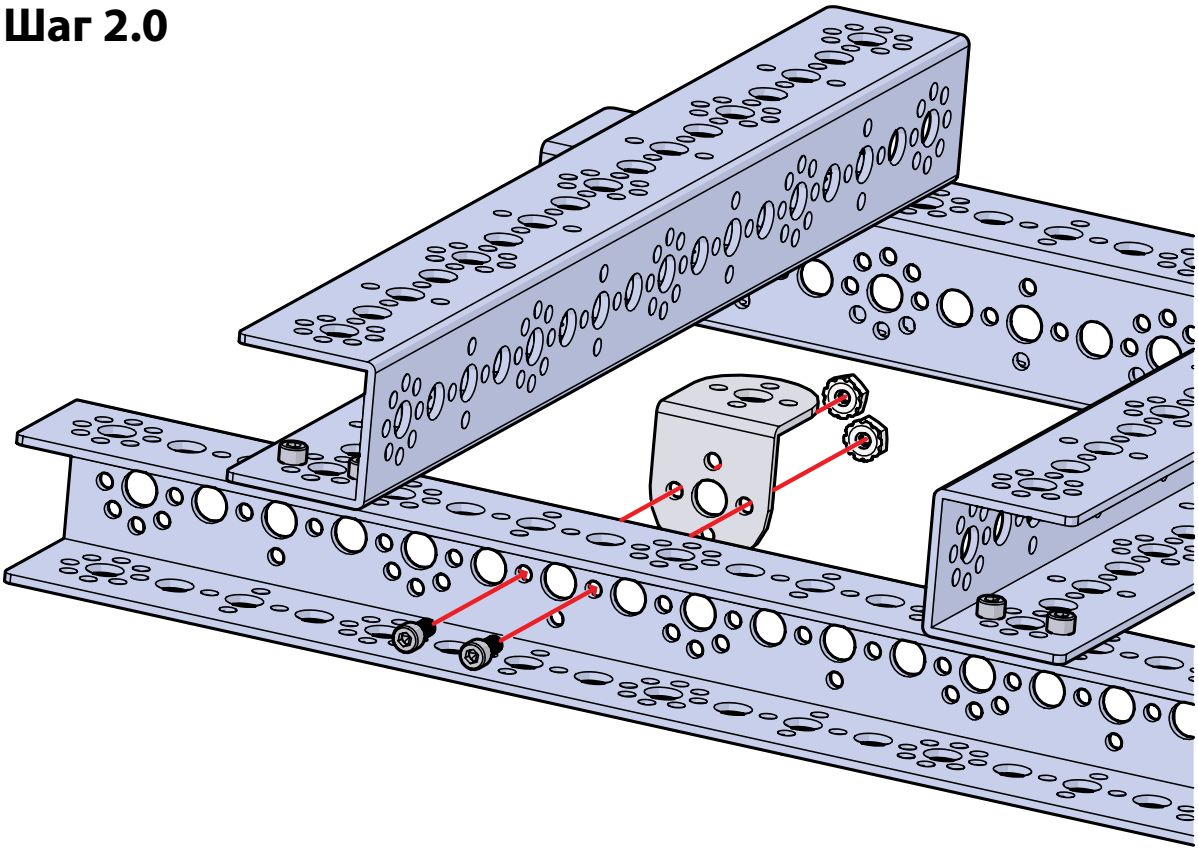
Частично собранная конструкция должна выглядеть так.



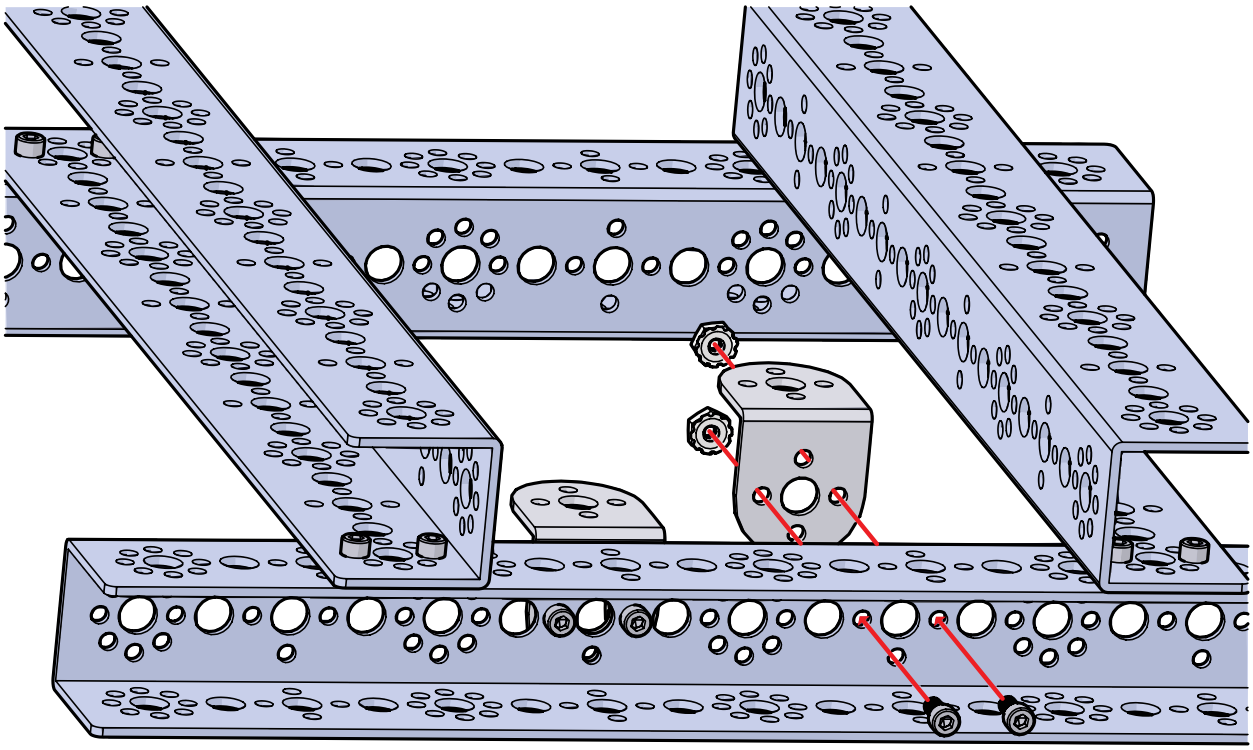
Поверните собранную конструкцию, чтобы сличить её с изображением.



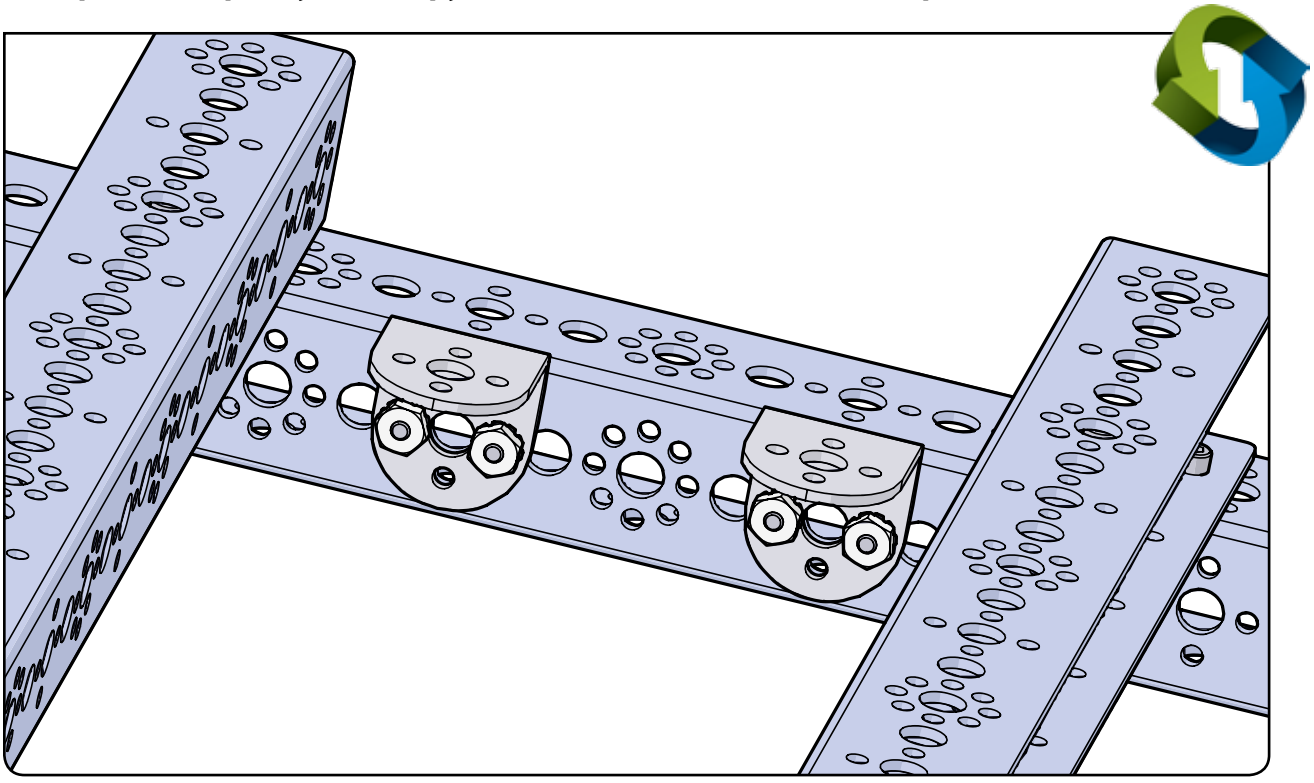
Шаг 2.0



Шаг 2.1

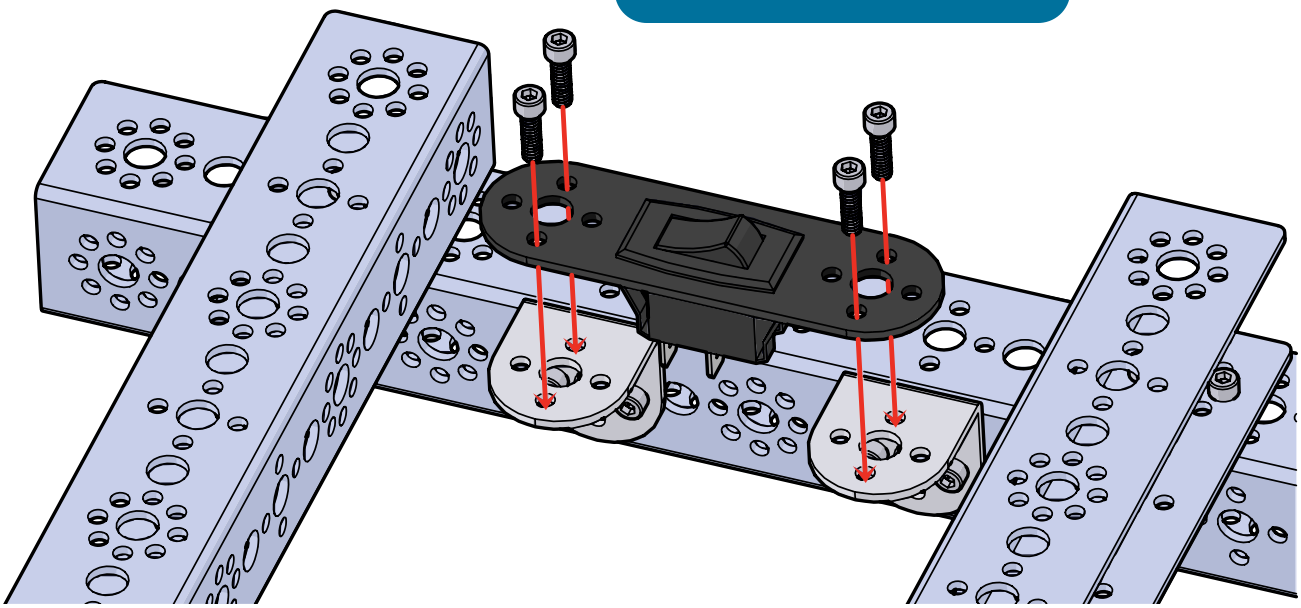


Поверните собранную конструкцию, чтобы сличить её с изображением.



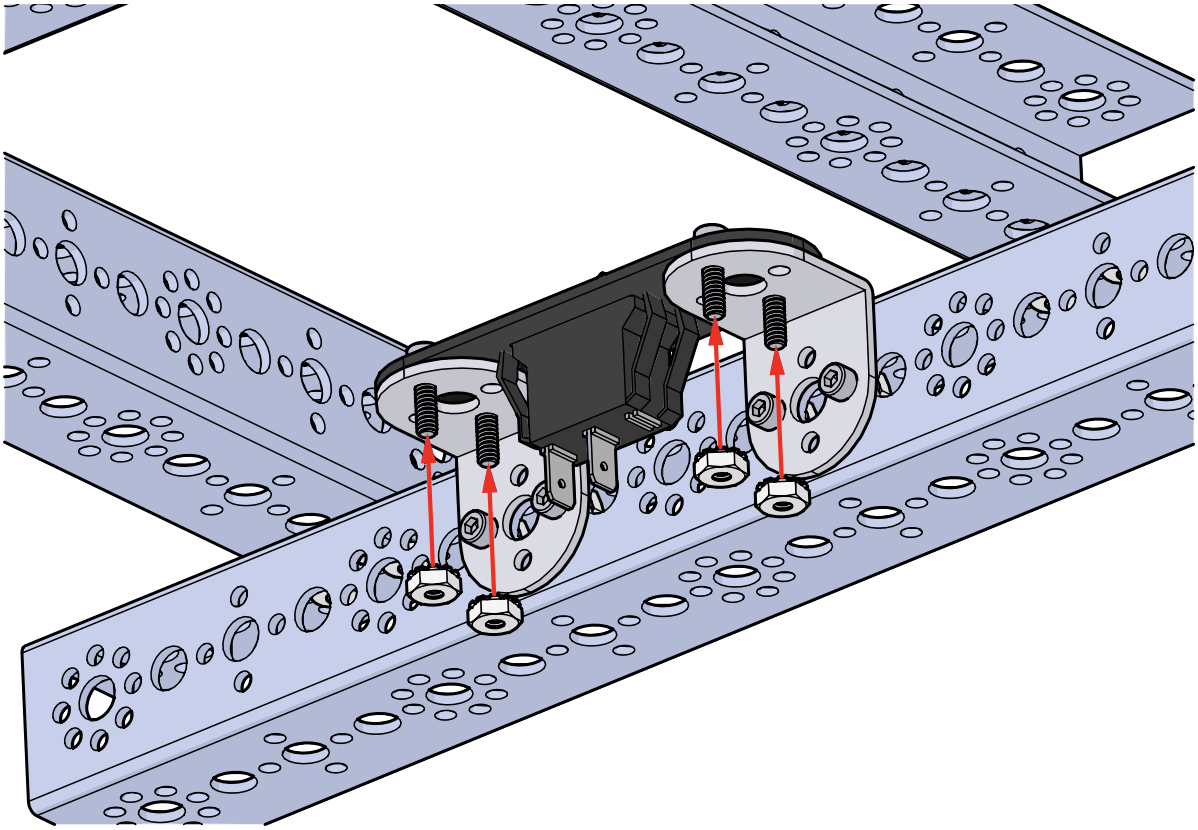
Шаг 2.2

Подсказка: На этом шаге используются винты с головкой под торцевой ключ (39097).



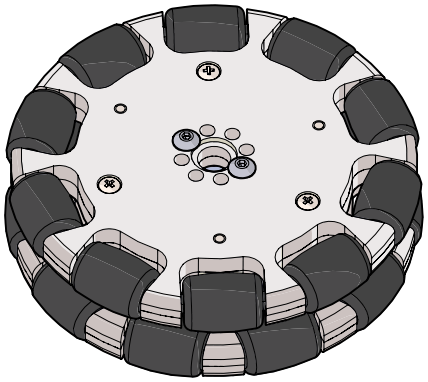
Подсказка: Всё равно, в какую сторону будет повёрнут выключатель после установки на пластину. Это дело исключительно вашего личного предпочтения.

Шаг 2.3

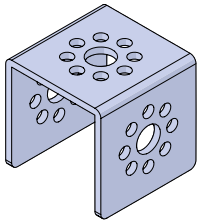


Шаг 3

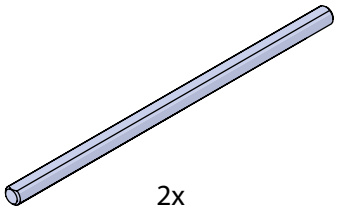
Необходимые детали и принадлежности



2x
Всенаправленное роликовое колесо 4" в комплекте/в сборе 36466



2x
Профильная рейка 32 мм 39065



2x
Ось 100 мм 39088

4x
Установочное кольцо на ось 39092

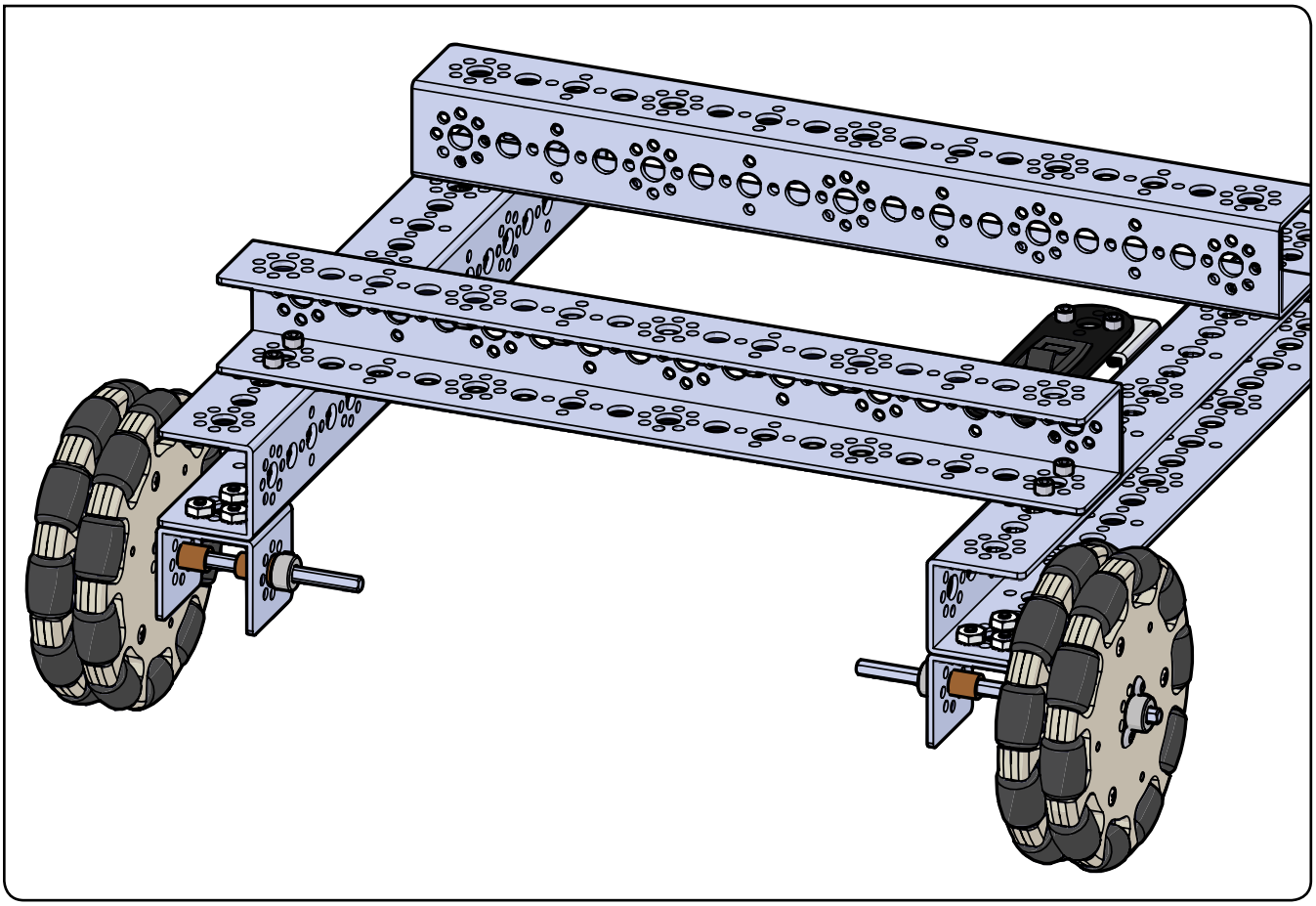
4x
Бронзовая втулка 39091

6x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

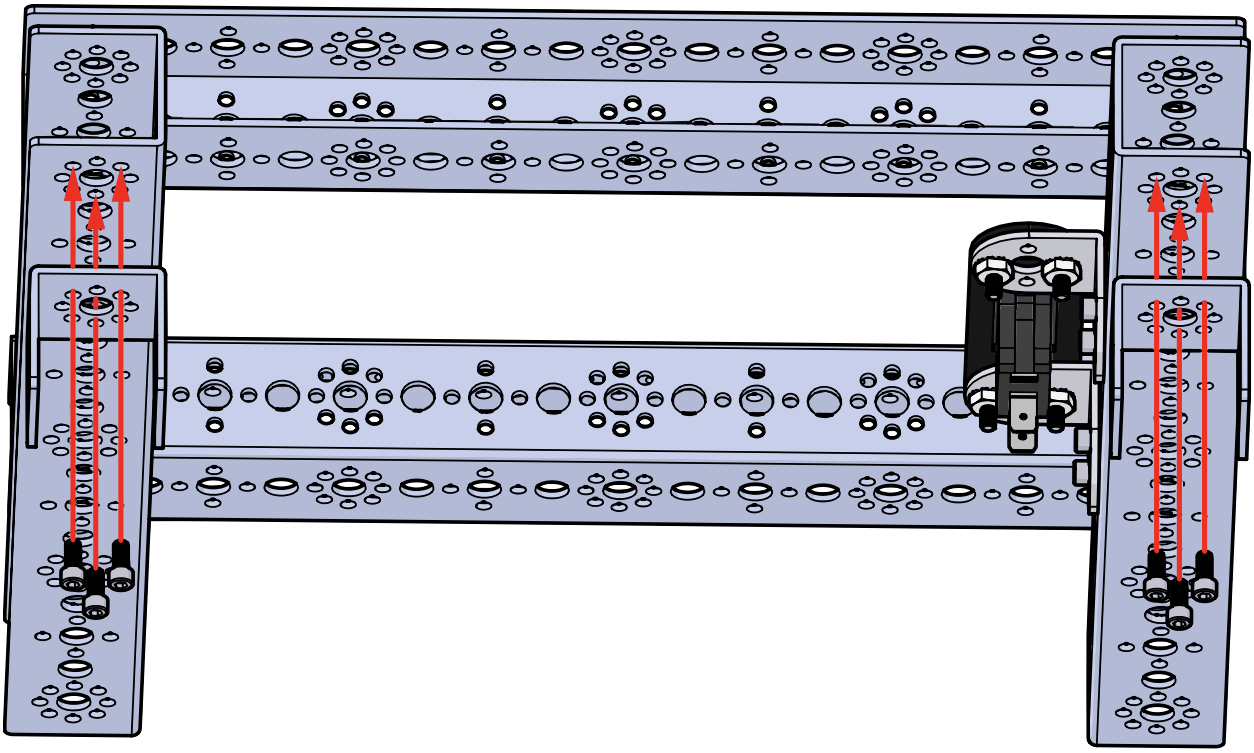
6x
Зубчатая гайка 39094

2x
Прокладка для оси 1/8" 39100

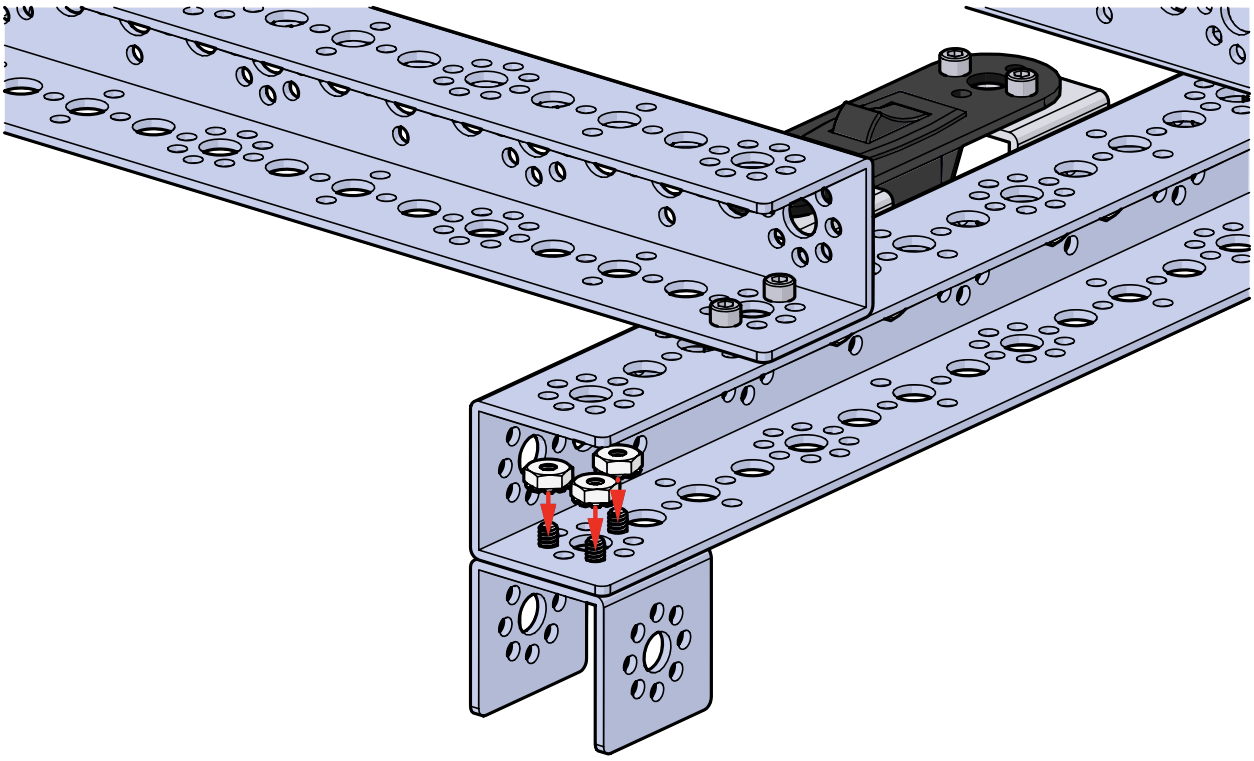
Частично собранная конструкция должна выглядеть так.



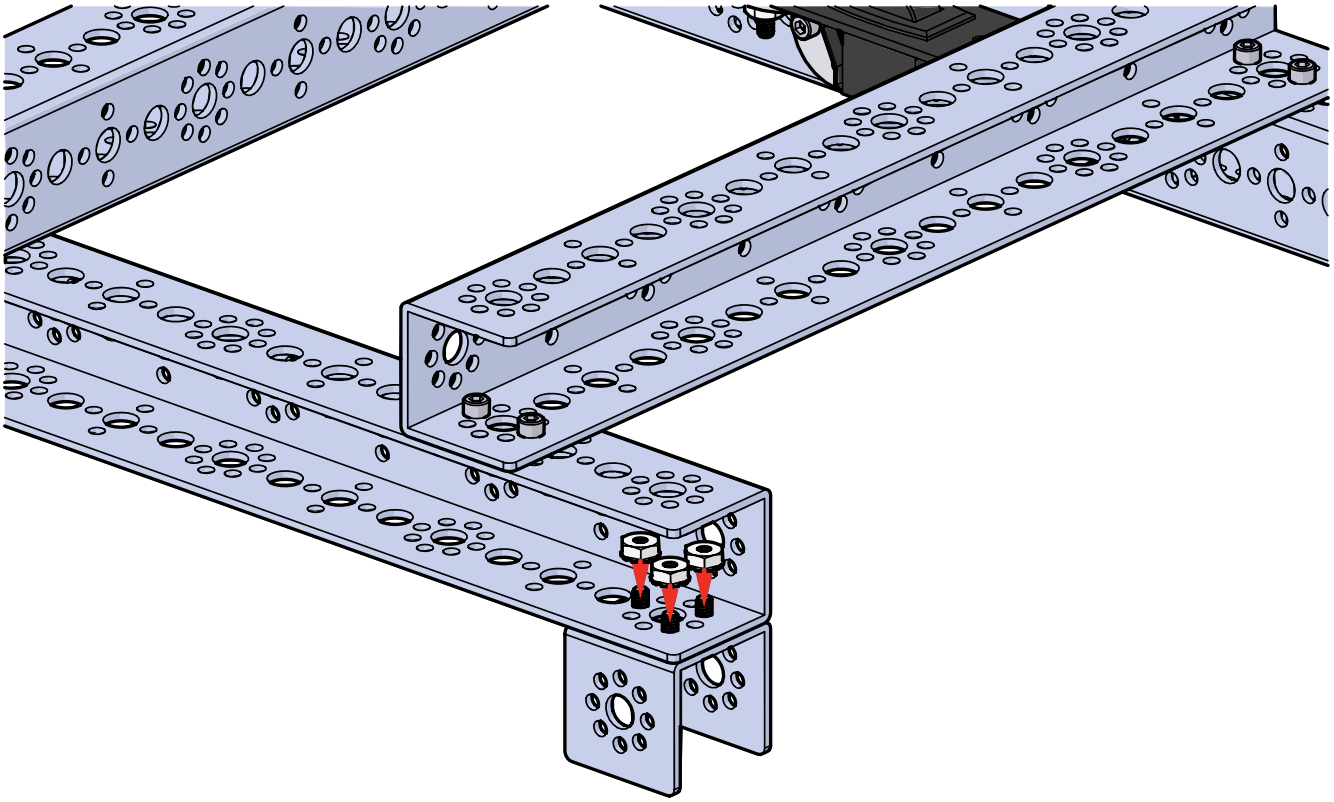
Шаг 3.0



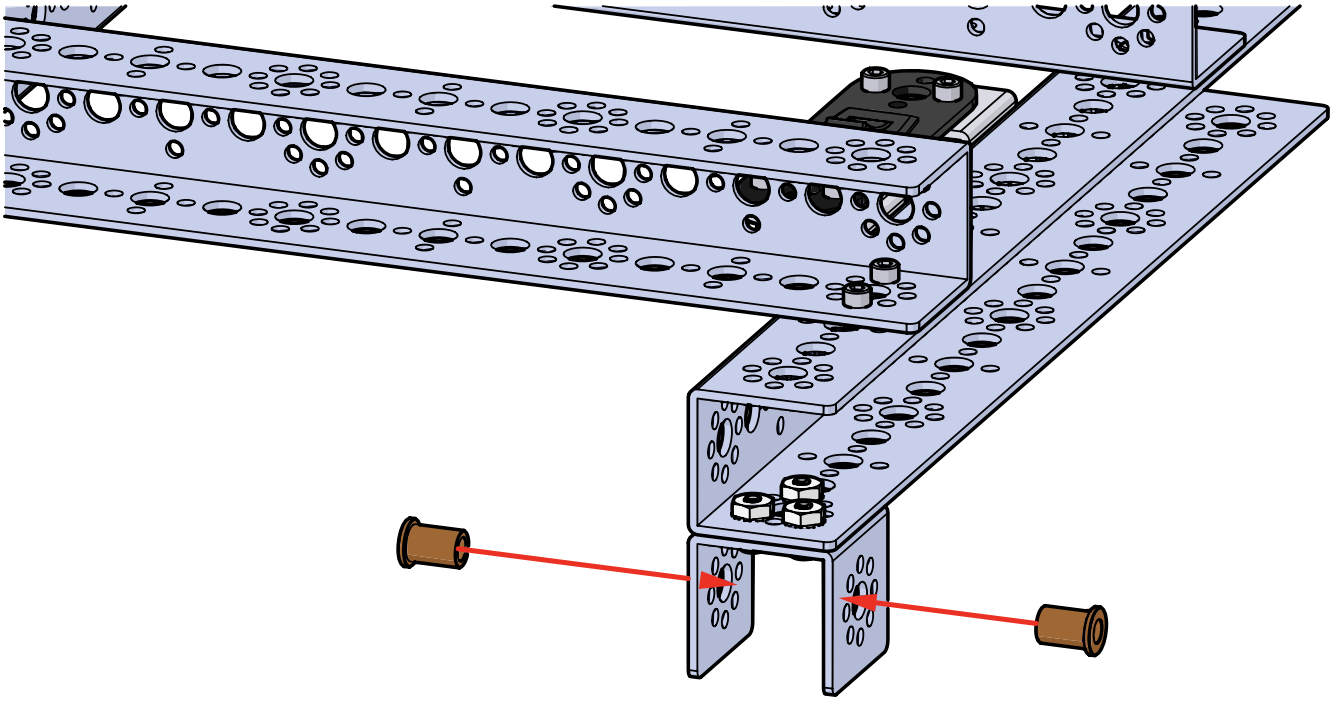
Шаг 3.1



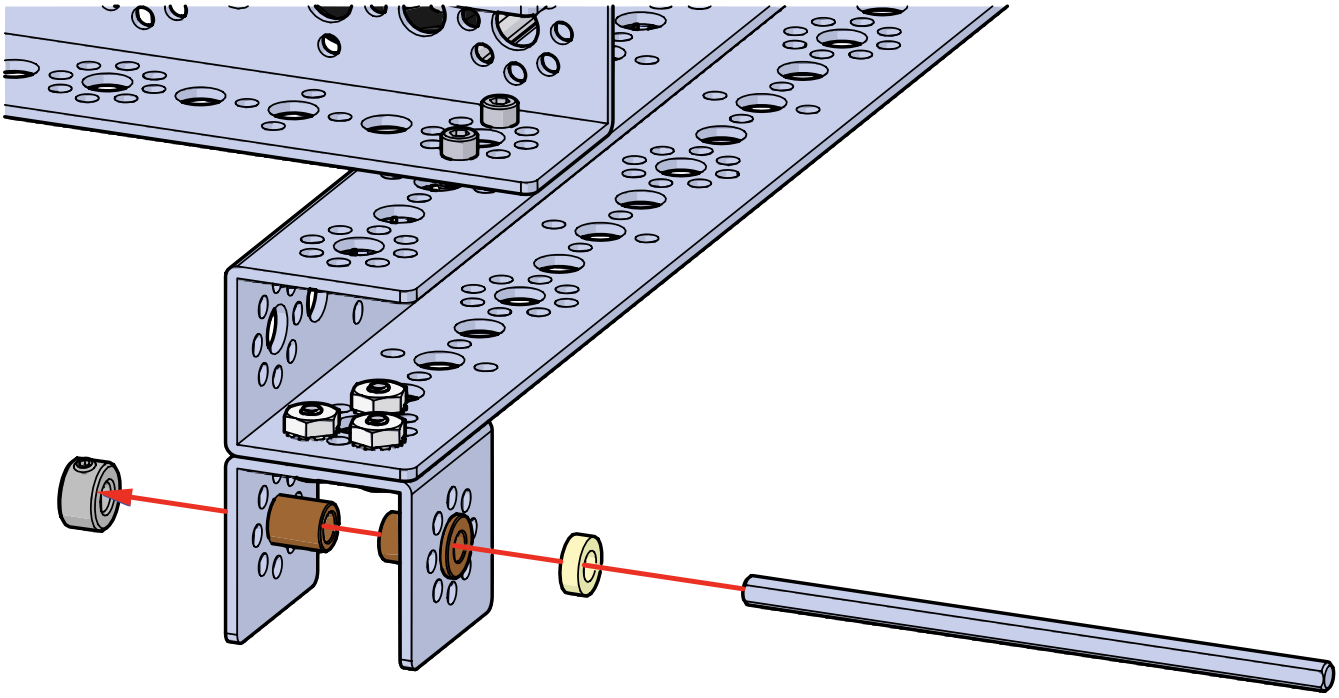
Шаг 3.2



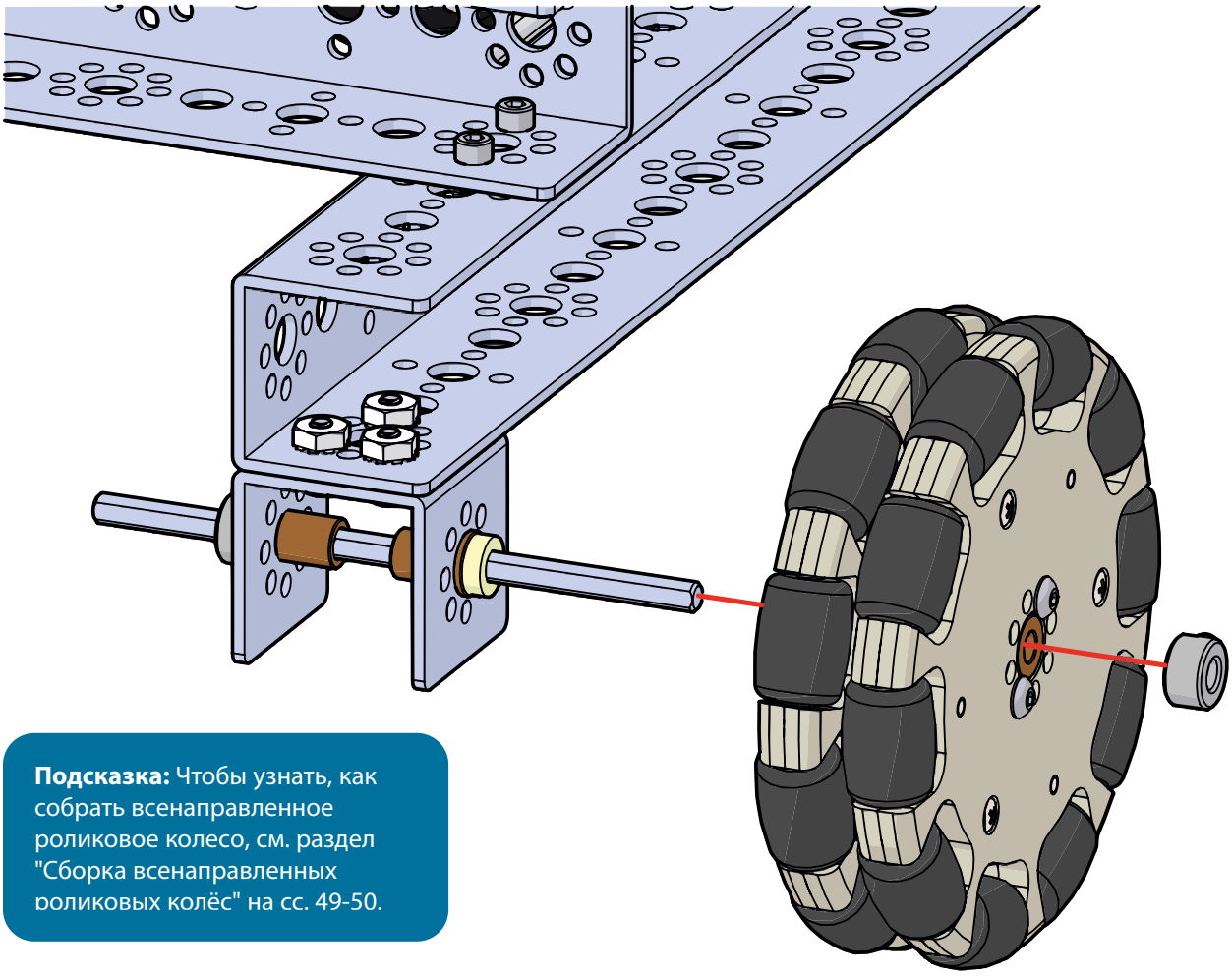
Шаг 3.3



Шаг 3.4

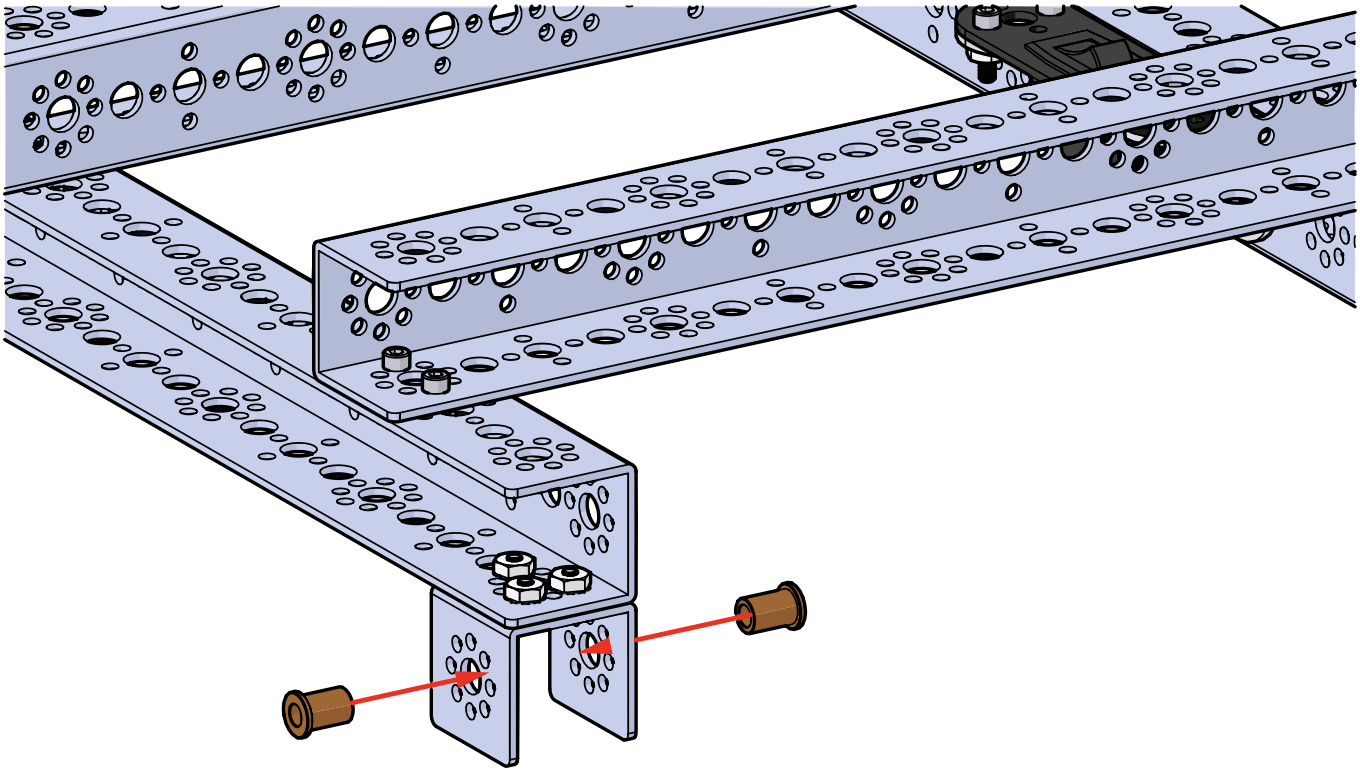


Шаг 3.5

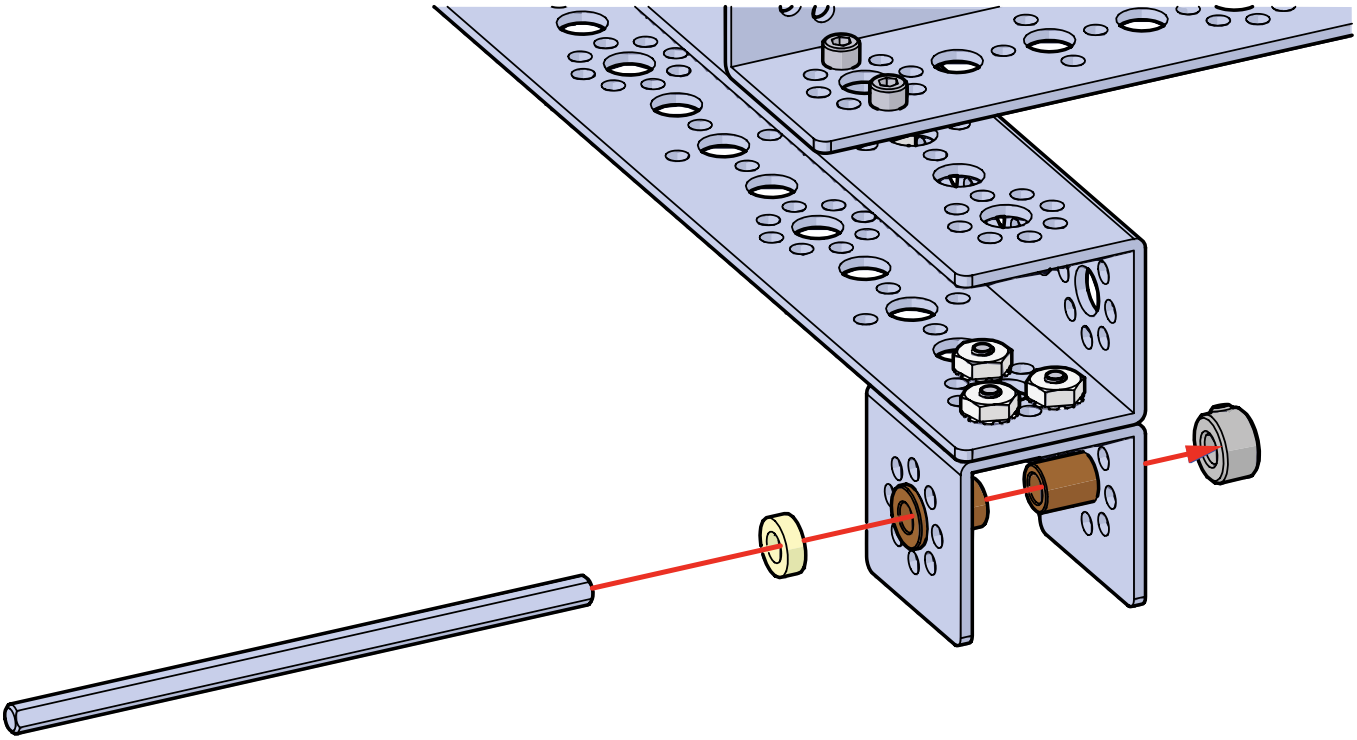


Подсказка: Чтобы узнать, как собрать всенаправленное роликовое колесо, см. раздел "Сборка всенаправленных роликовых колёс" на сс. 49-50.

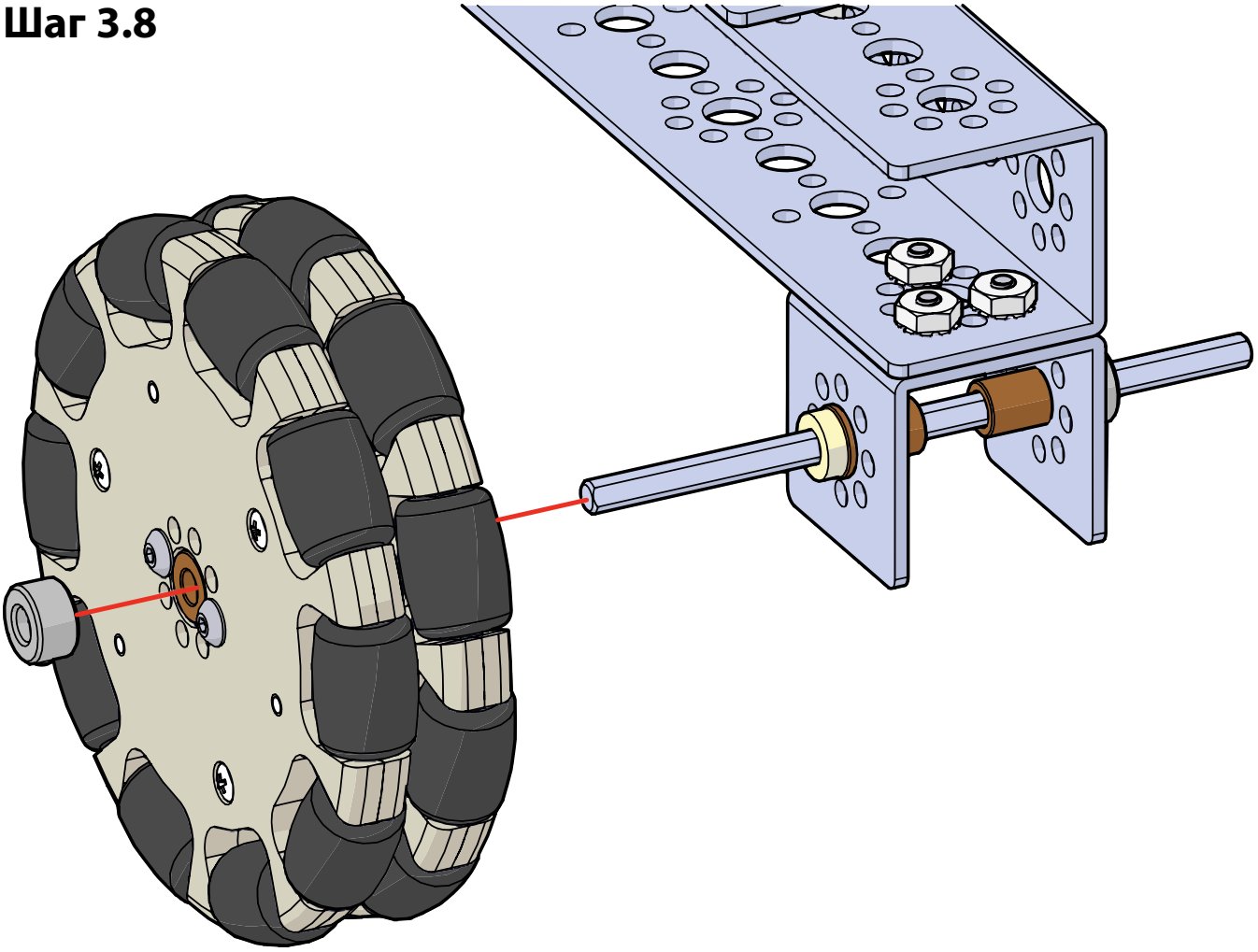
Шаг 3.6



Шаг 3.7

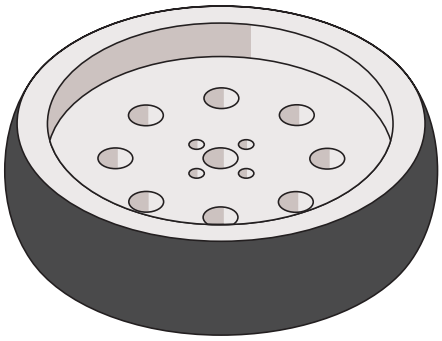


Шаг 3.8

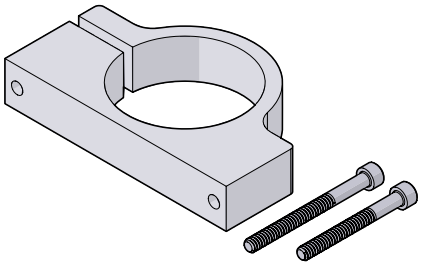


Шаг 4

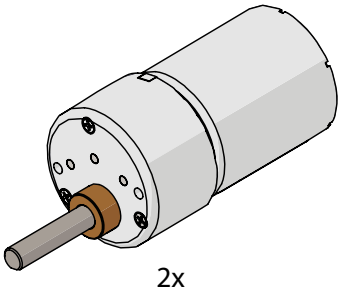
Необходимые детали и принадлежности



2x
Колесо 4" 39055



2x
Монтажная опора электродвигателя 39089



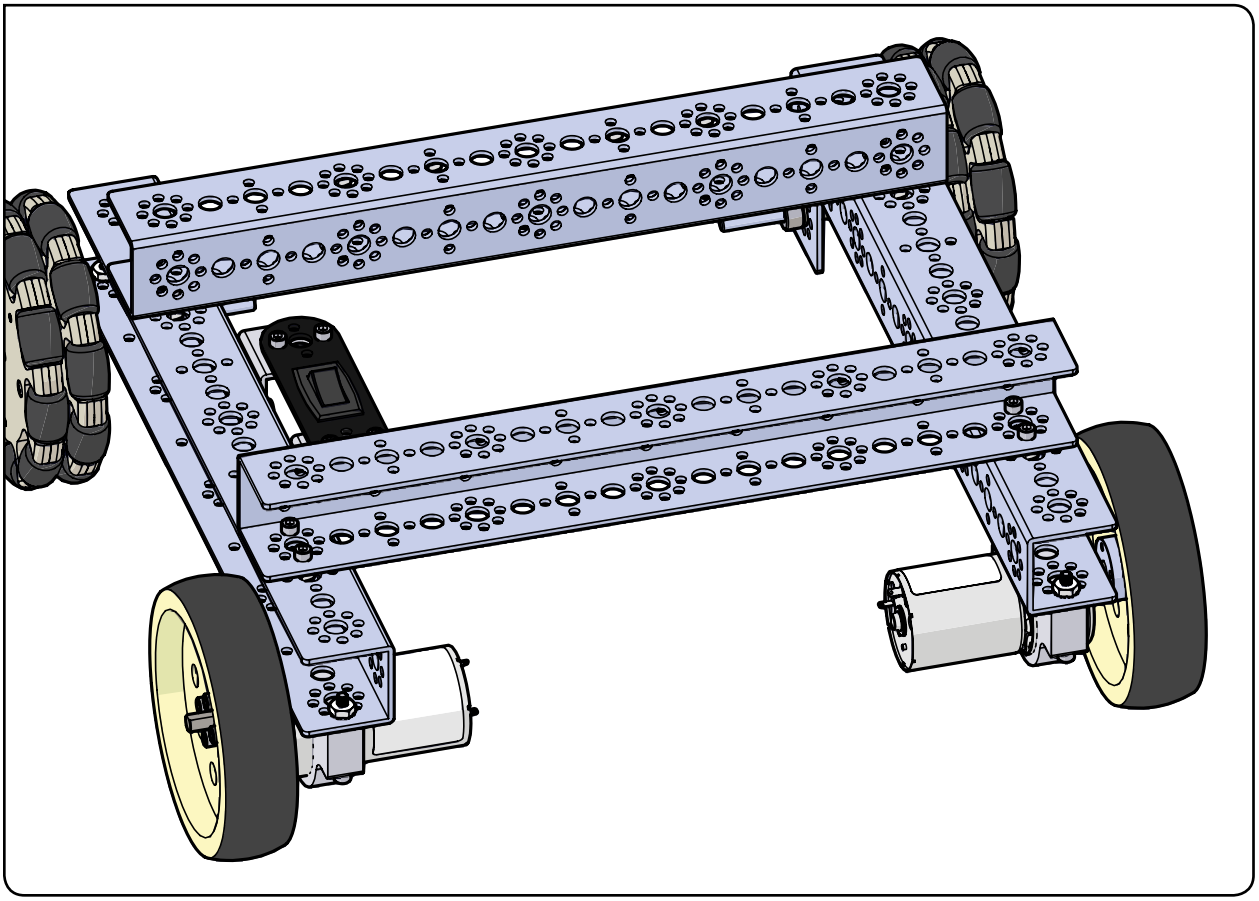
2x
Редукторный электродвигатель постоянного тока 39530

2x
Ступица для вала электродвигателя 39079

8x
Винт с головкой под торцевой ключ 6-32 x 1/2" 39097

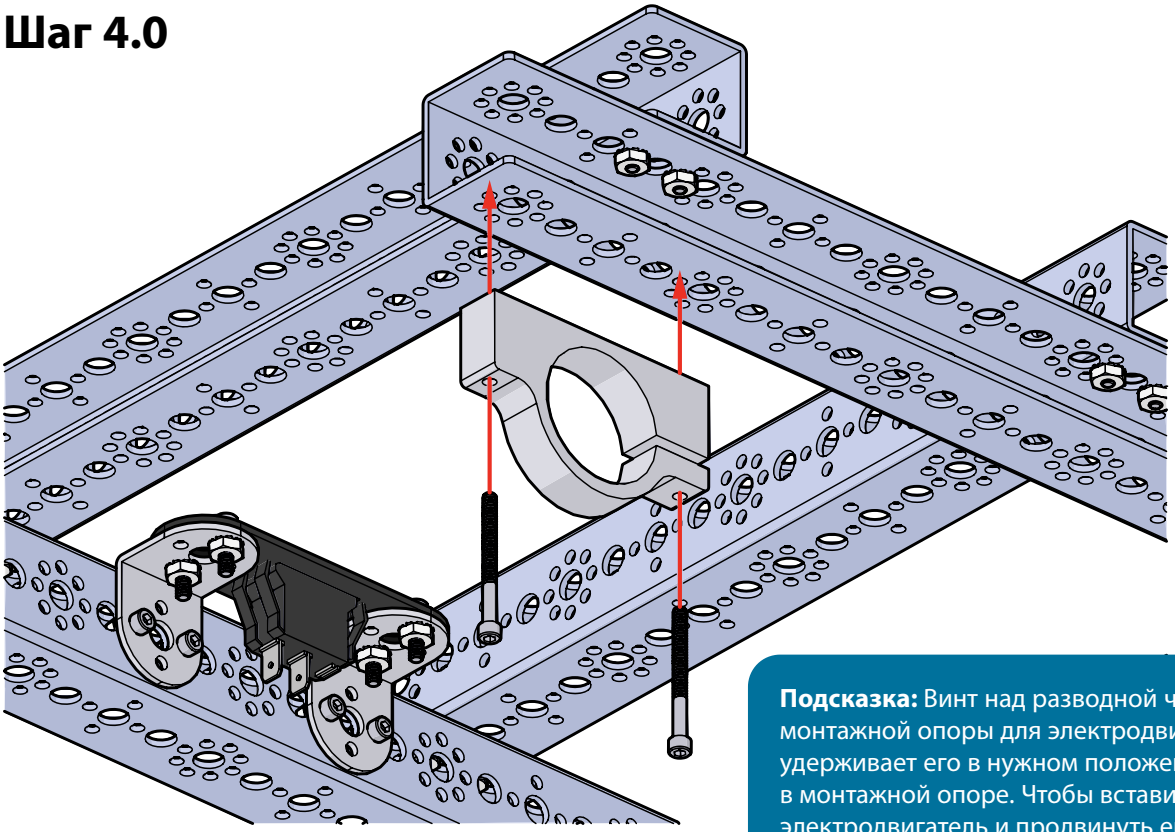
4x
Зубчатая гайка 39094

Собранная модель должна выглядеть так.



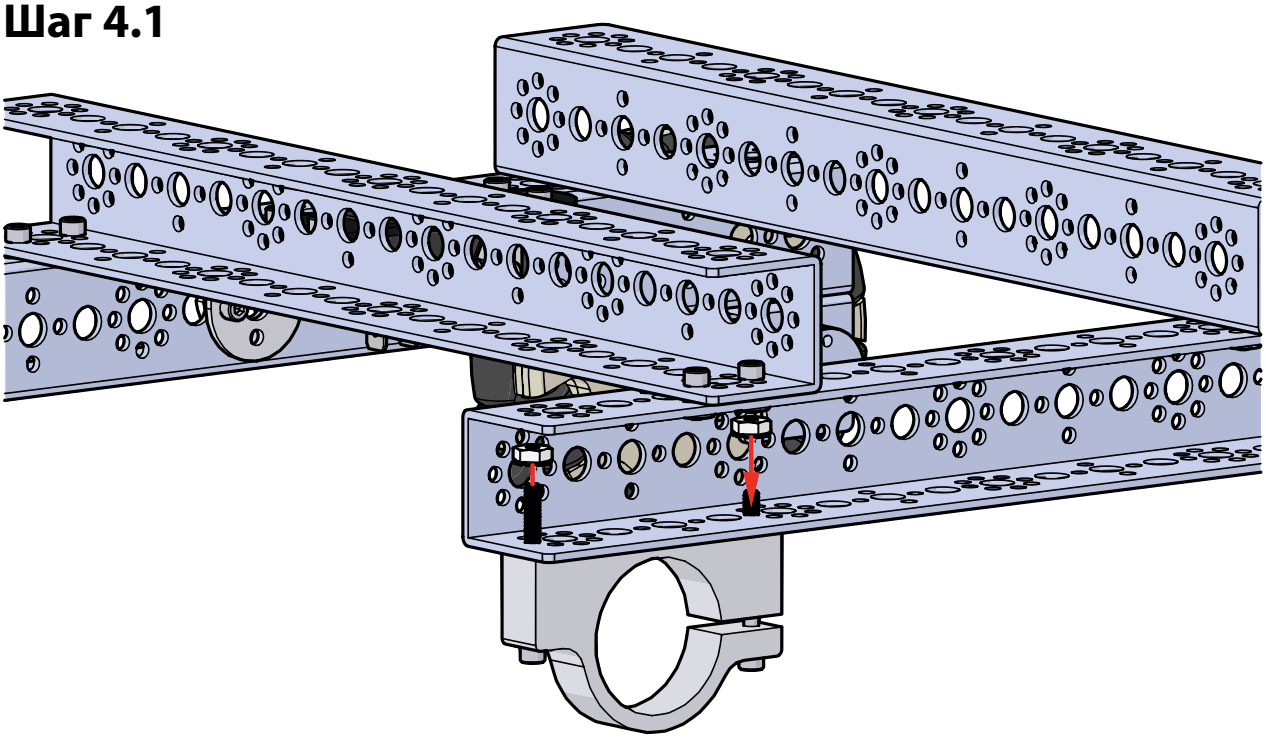
Подсказка: Чтобы научиться различать ступицу для вала электродвигателя и ступицу для оси см. с. 48.

Шаг 4.0

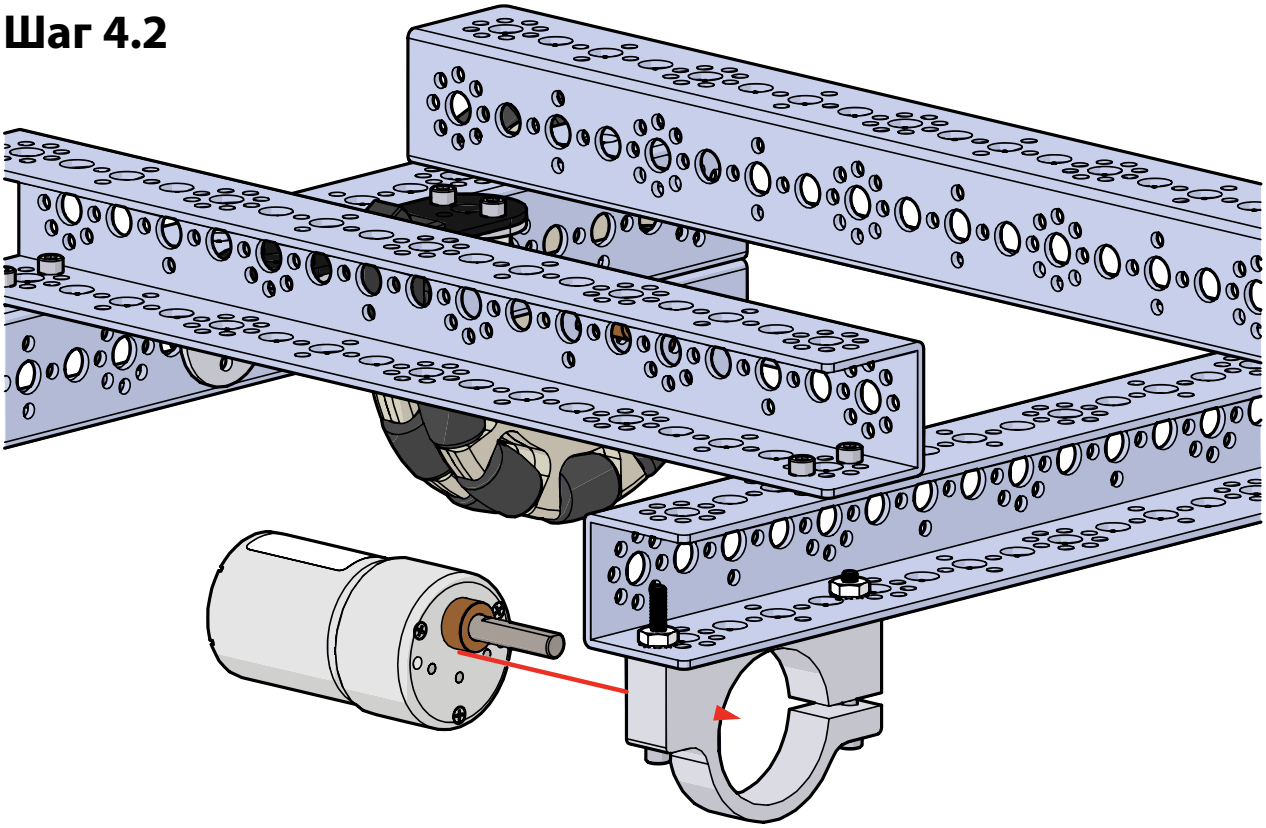


Подсказка: Винт над разводной частью монтажной опоры для электродвигателя удерживает его в нужном положении в монтажной опоре. Чтобы вставить электродвигатель и продвинуть его в нужное положение, винт необходимо ослабить. Не забудьте затянуть этот винт, после того как вставите электродвигатель и продвинете его в нужное положение.

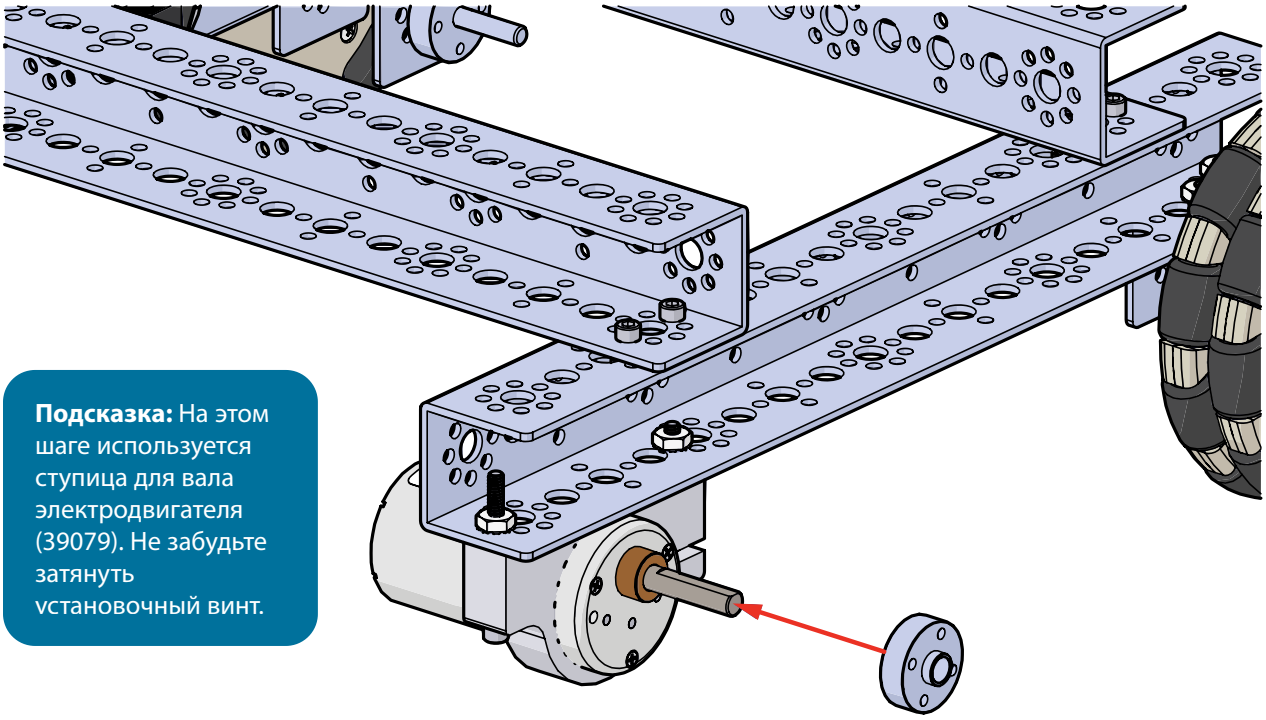
Шаг 4.1



Шаг 4.2

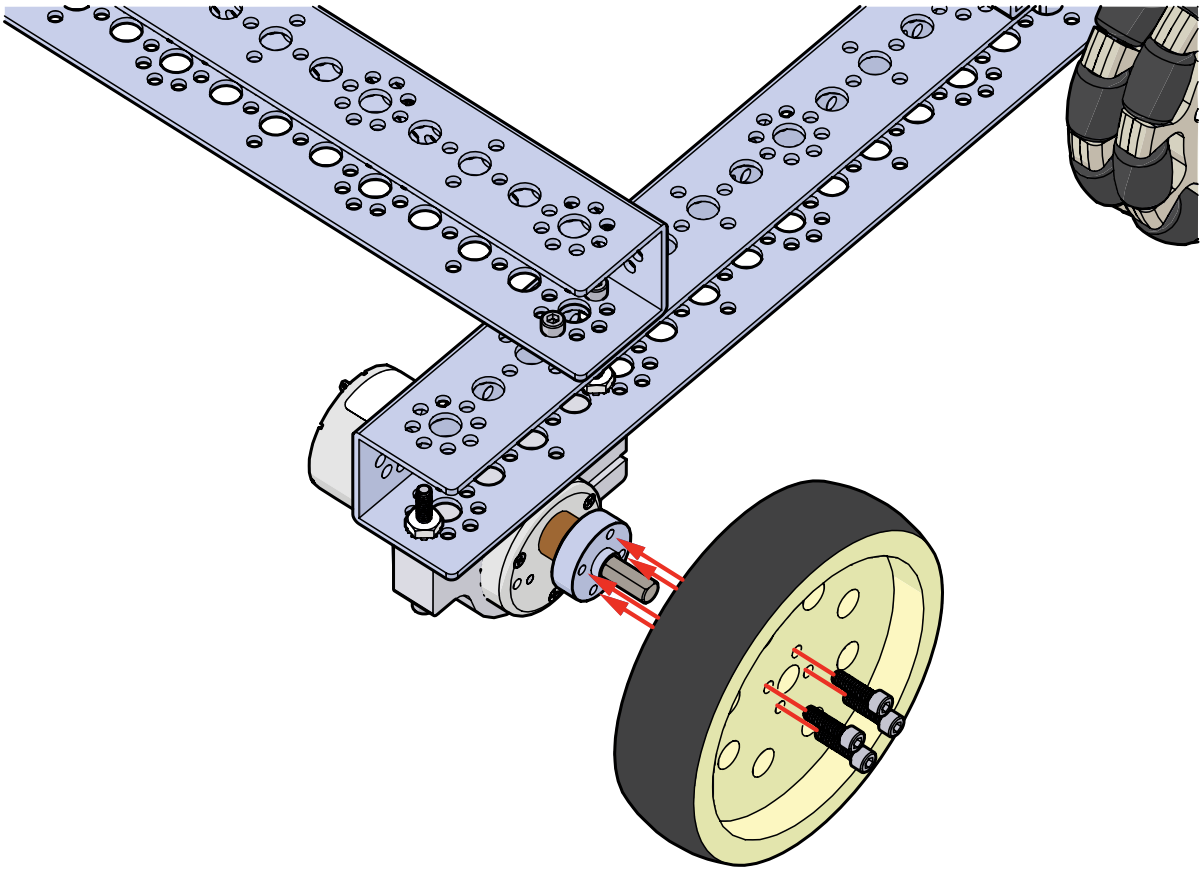


Шаг 4.3



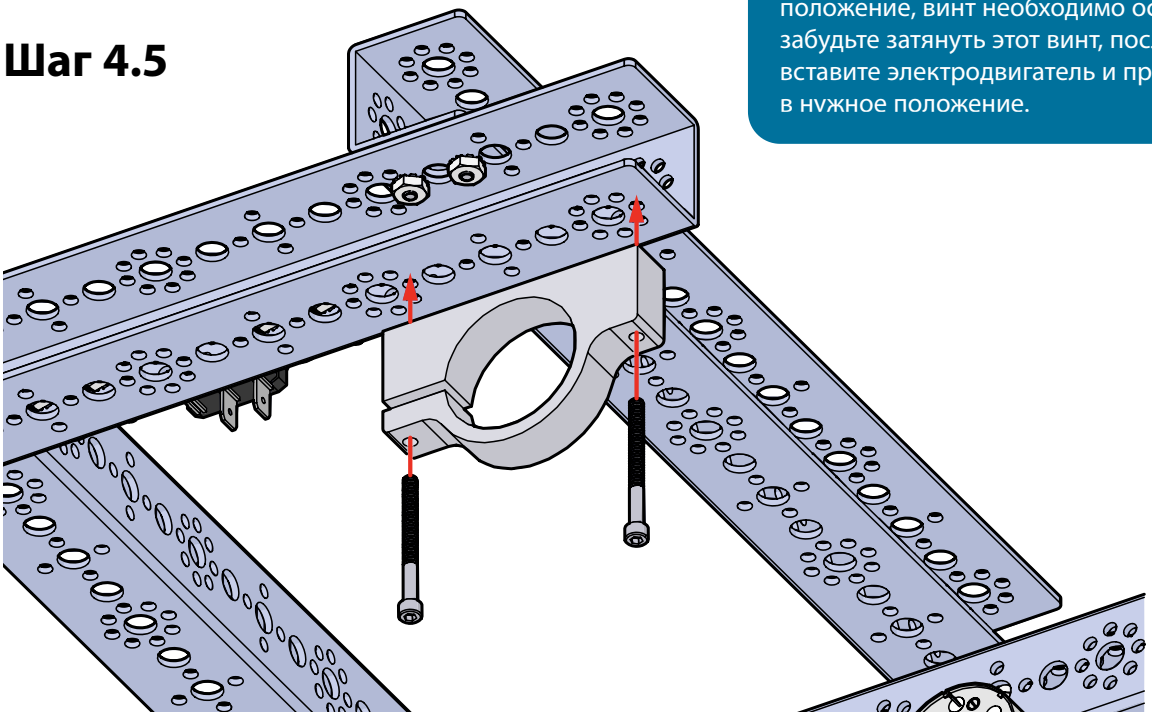
Подсказка: На этом шаге используется ступица для вала электродвигателя (39079). Не забудьте затянуть установочный винт.

Шаг 4.4

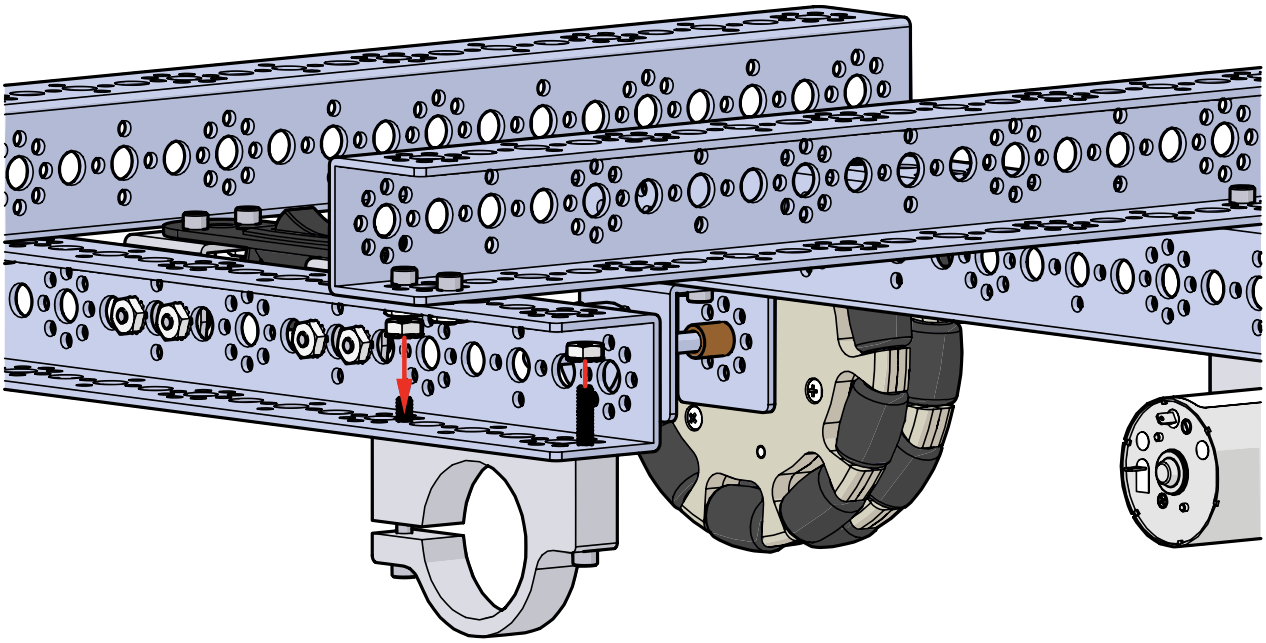


Подсказка: Винт над разводной частью монтажной опоры для электродвигателя удерживает его в нужном положении в монтажной опоре. Чтобы вставить электродвигатель и продвинуть его в нужное положение, винт необходимо ослабить. Не забудьте затянуть этот винт, после того как вставите электродвигатель и продвинете его в нужное положение.

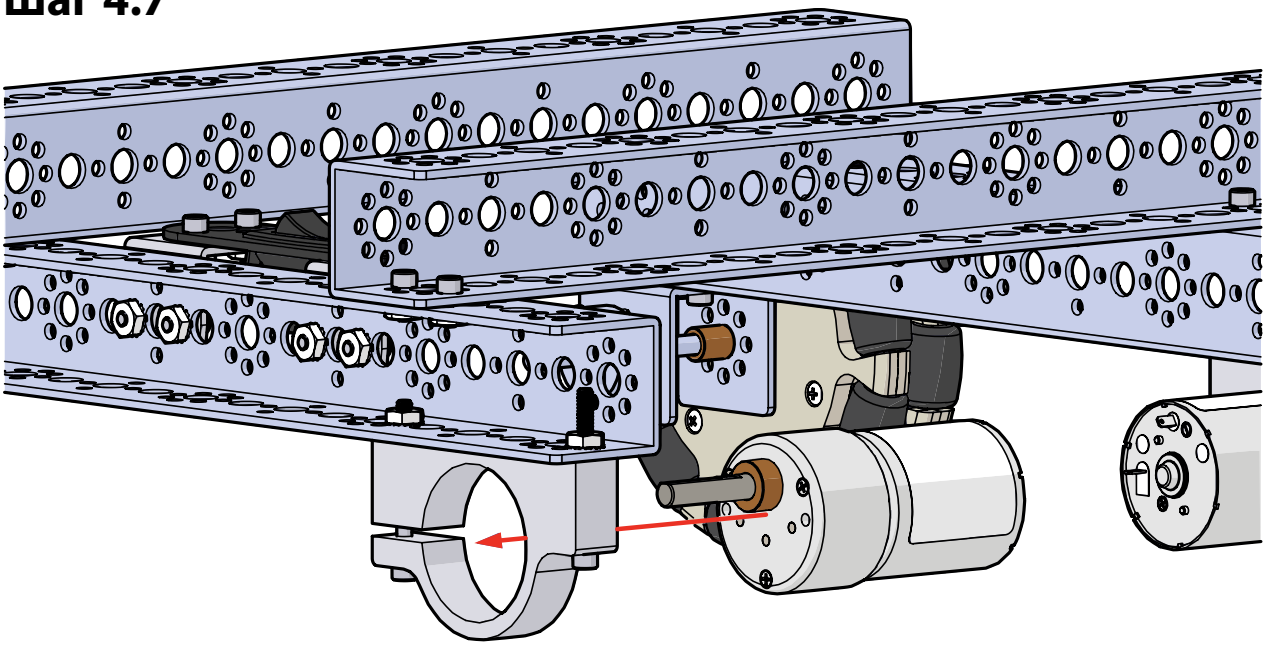
Шаг 4.5



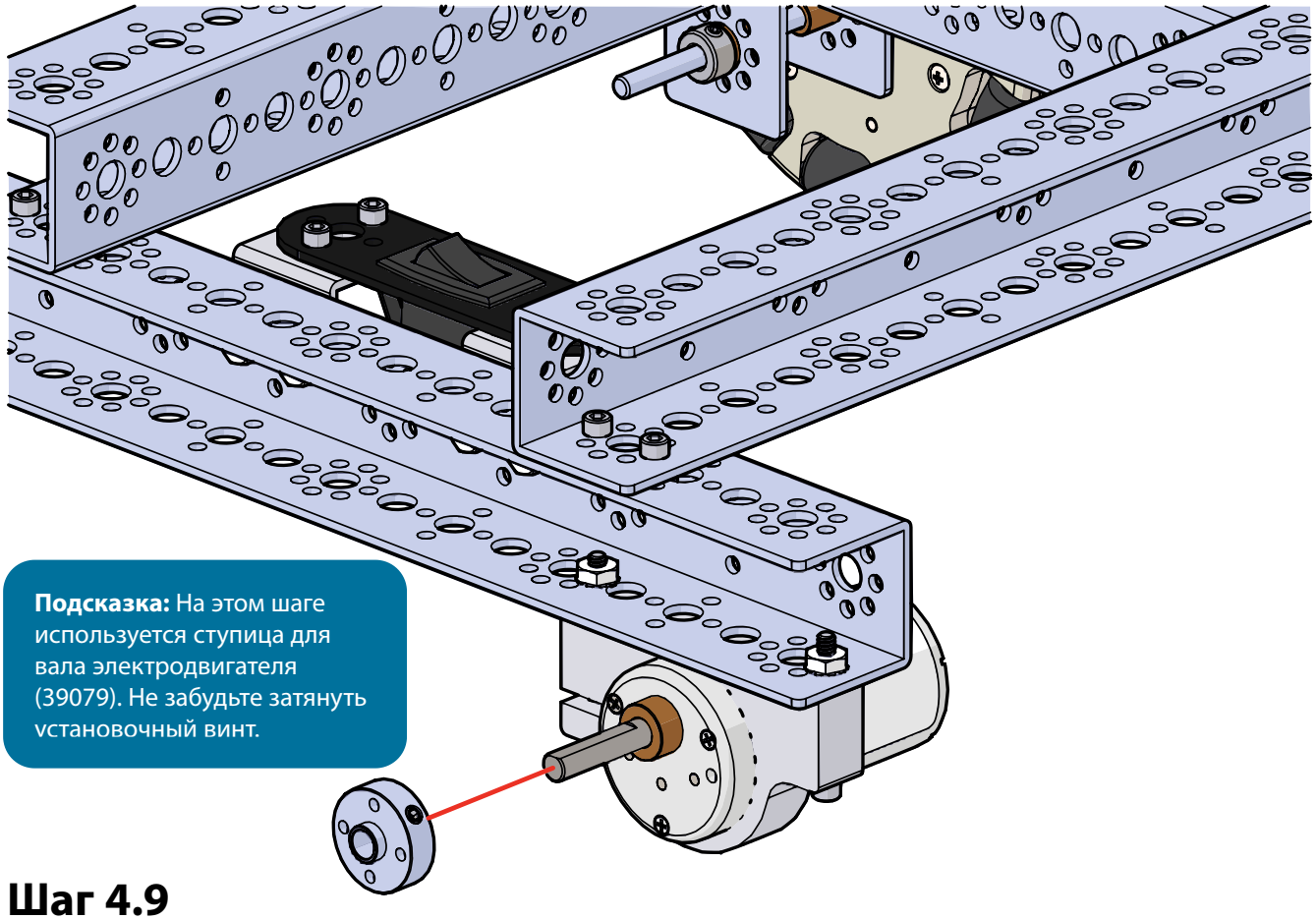
Шаг 4.6



Шаг 4.7

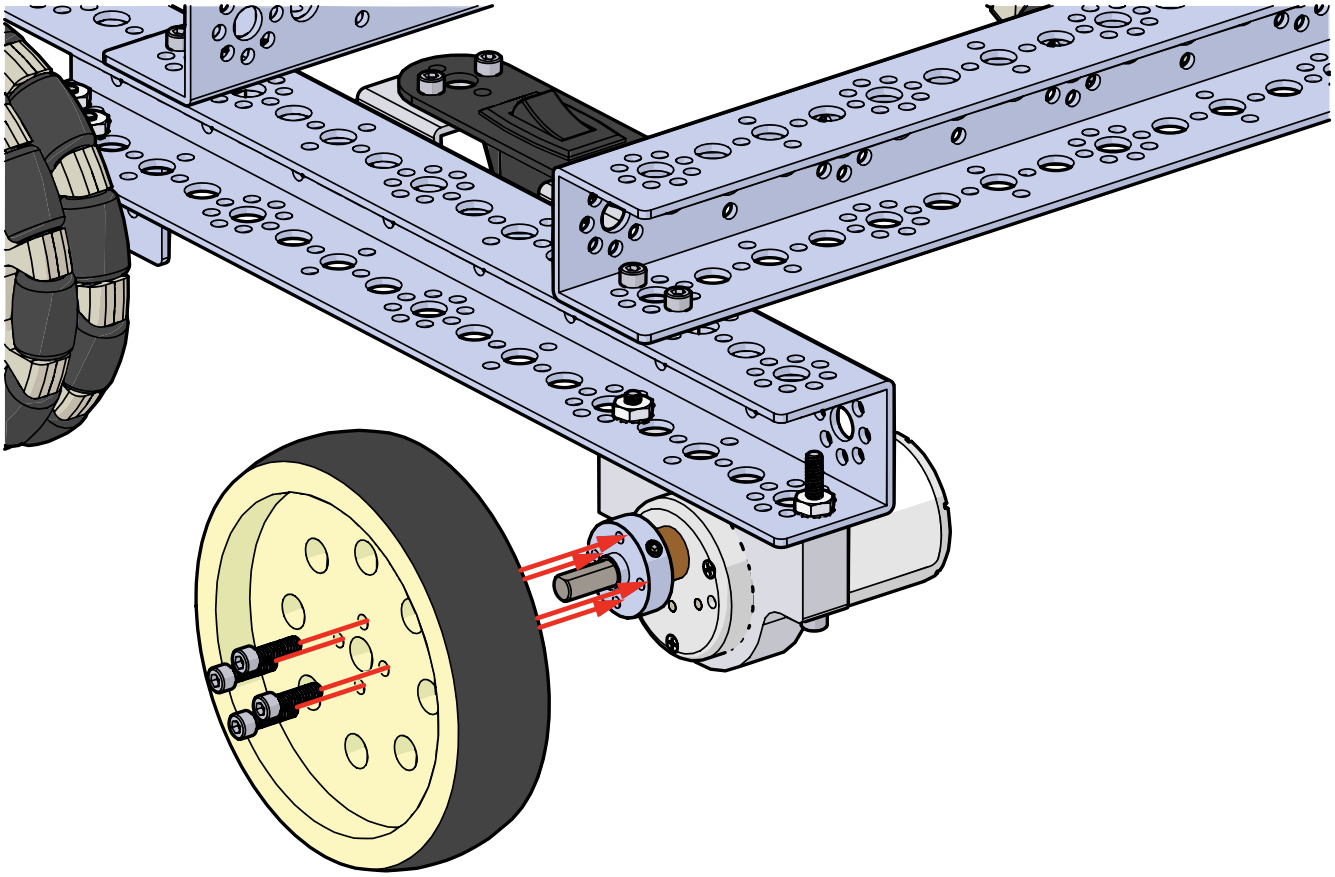


Шаг 4.8



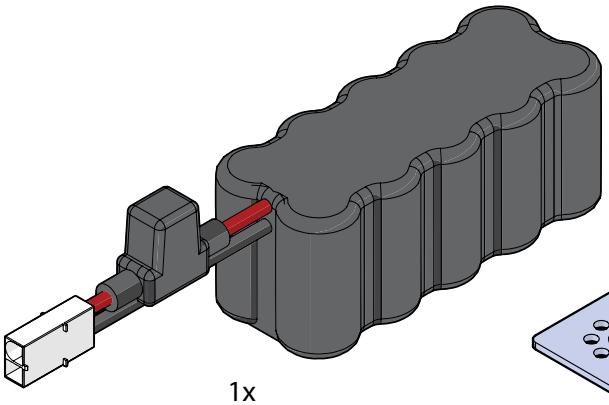
Подсказка: На этом шаге используется ступица для вала электродвигателя (39079). Не забудьте затянуть установочный винт.

Шаг 4.9

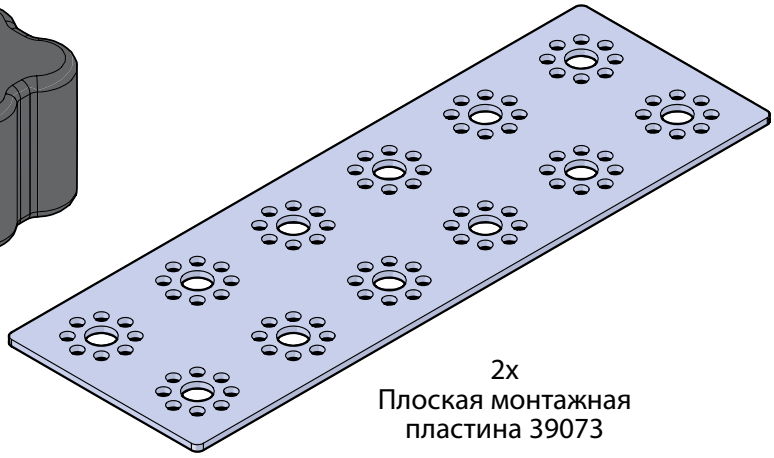


Шаг 5

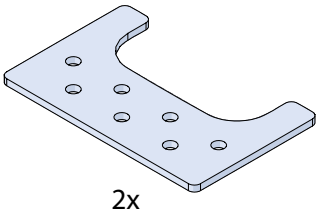
Необходимые детали и принадлежности



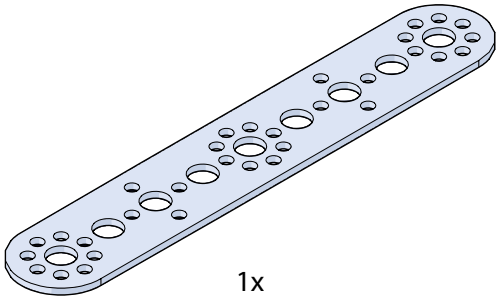
1x
Аккумуляторная батарея, 12 В, 3 000 мА 39057



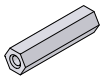
2x
Плоская монтажная пластина 39073



2x
Держатель аккумуляторной батареи 38009



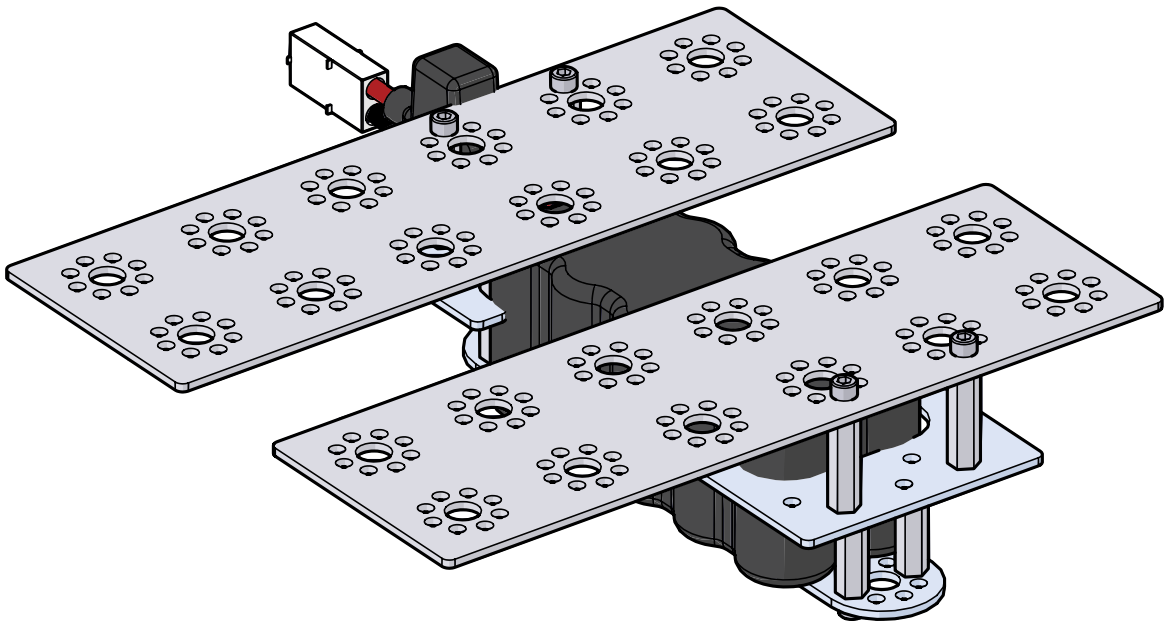
1x
Планка 160 мм x 27 мм 39272



8x
Распорная стойка 6-32 x 1" 39102

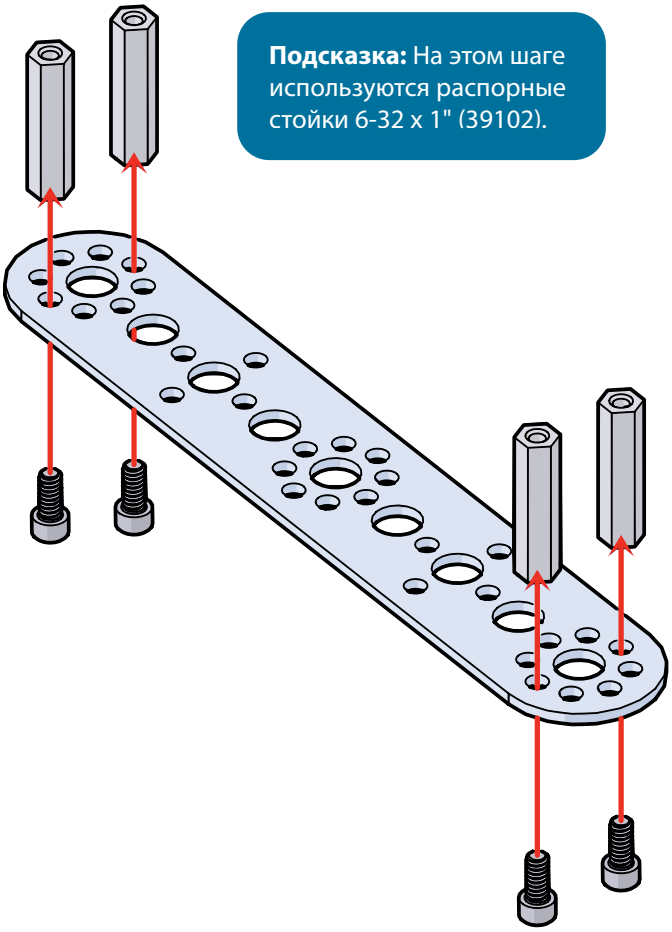
16x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

Собранная модель должна выглядеть так.

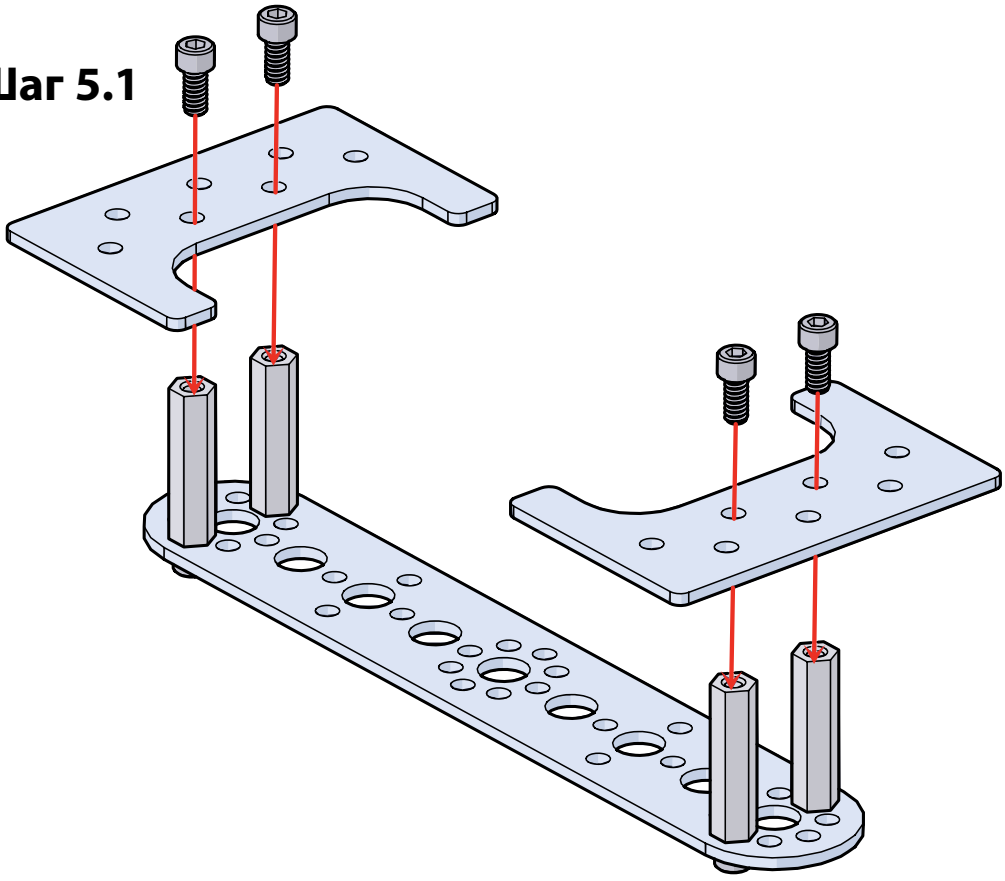


Шаг 5.0

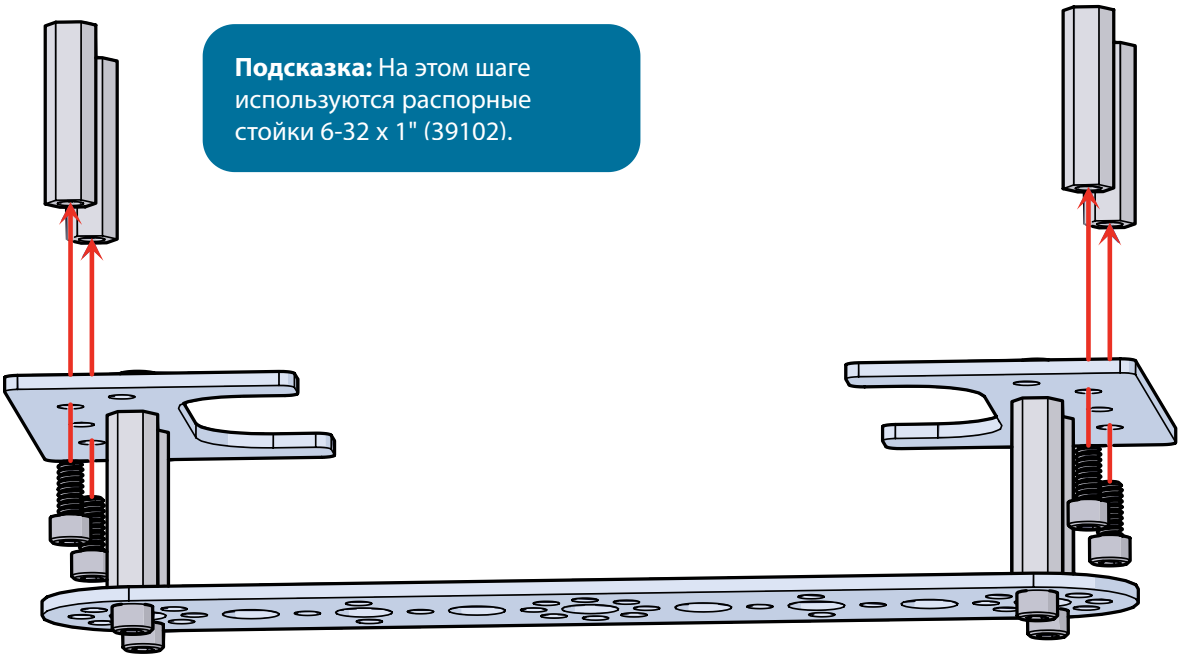
Подсказка: На этом шаге используются распорные стойки 6-32 x 1" (39102).



Шаг 5.1

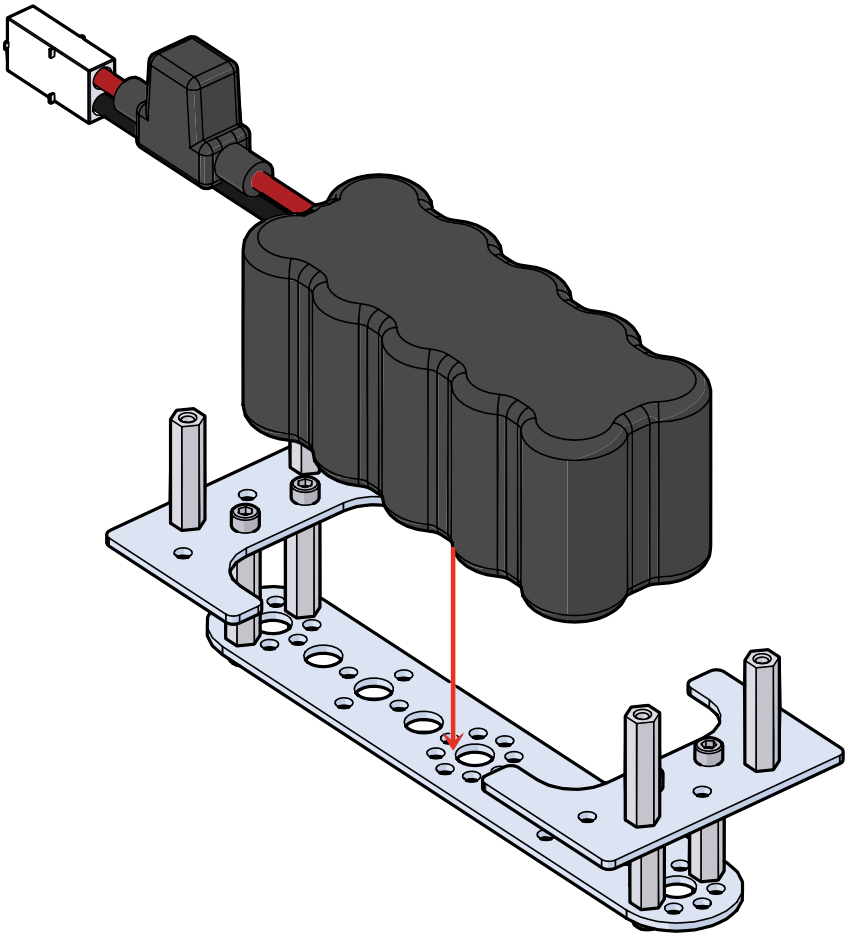


Шаг 5.2

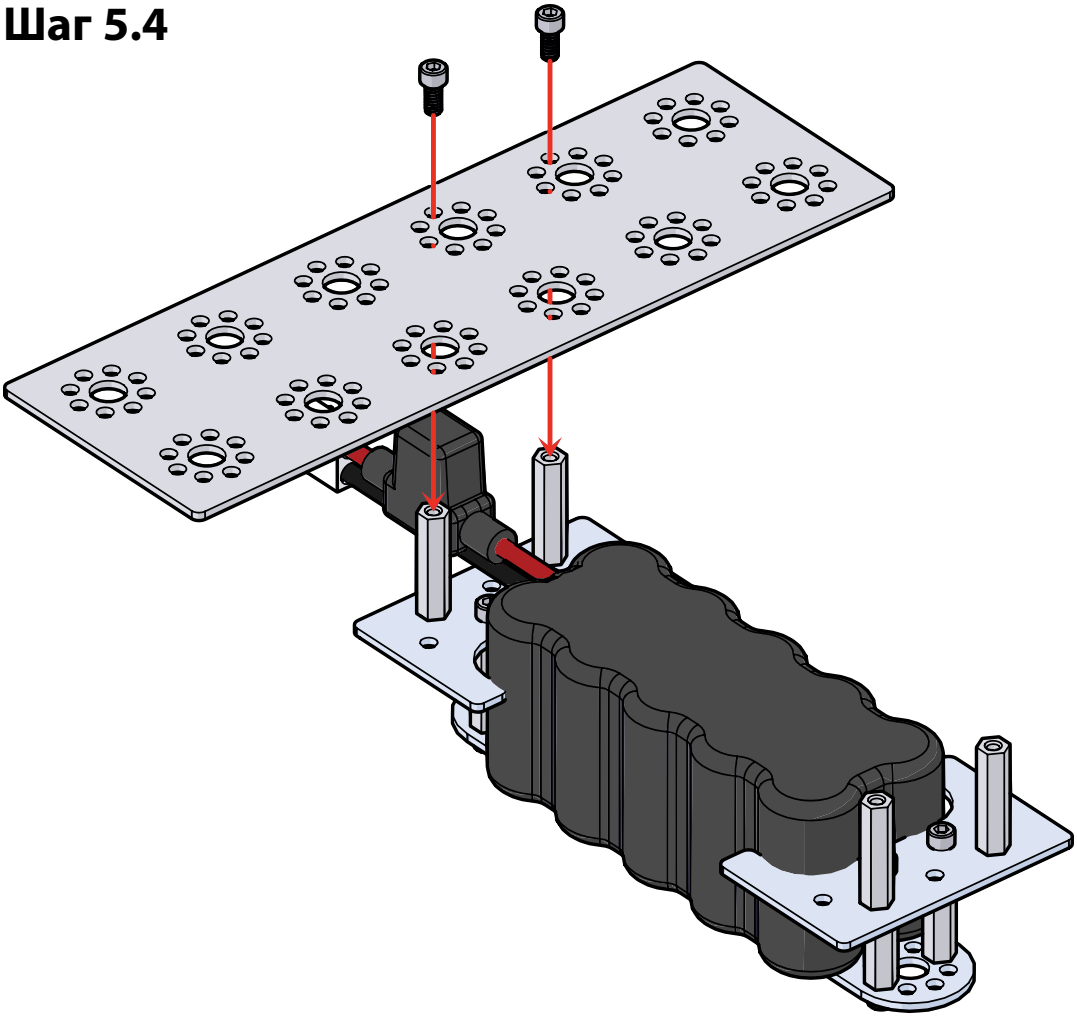


Подсказка: На этом шаге используются распорные стойки 6-32 x 1" (39102).

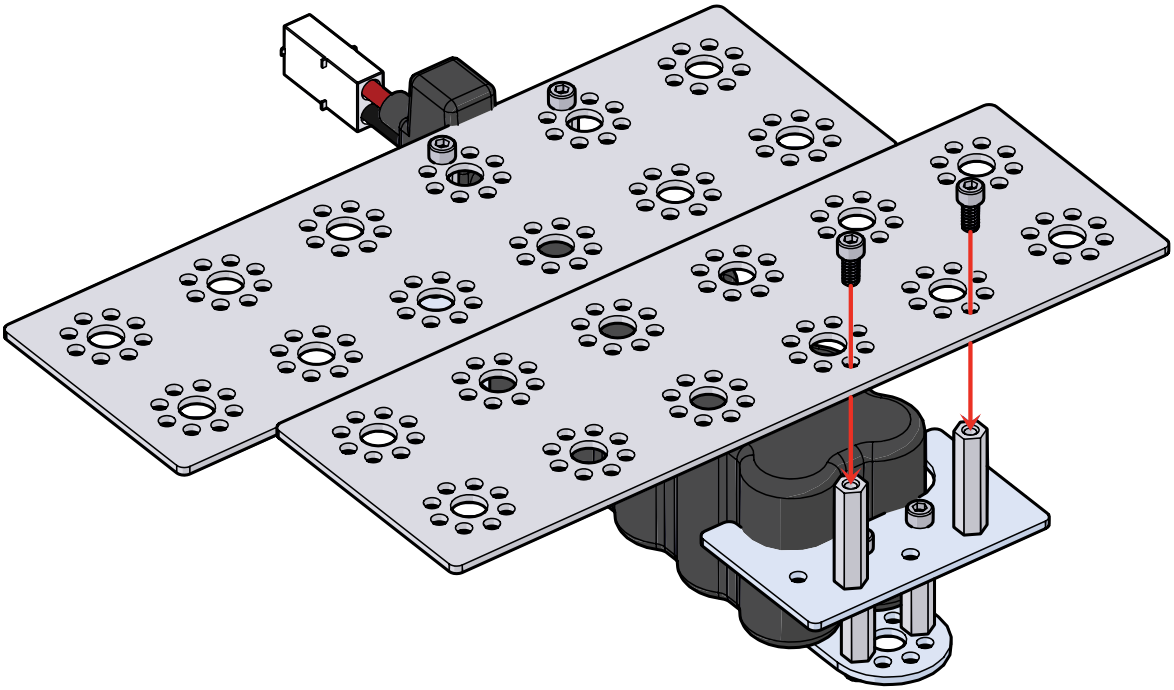
Шаг 5.3



Шаг 5.4



Шаг 5.5



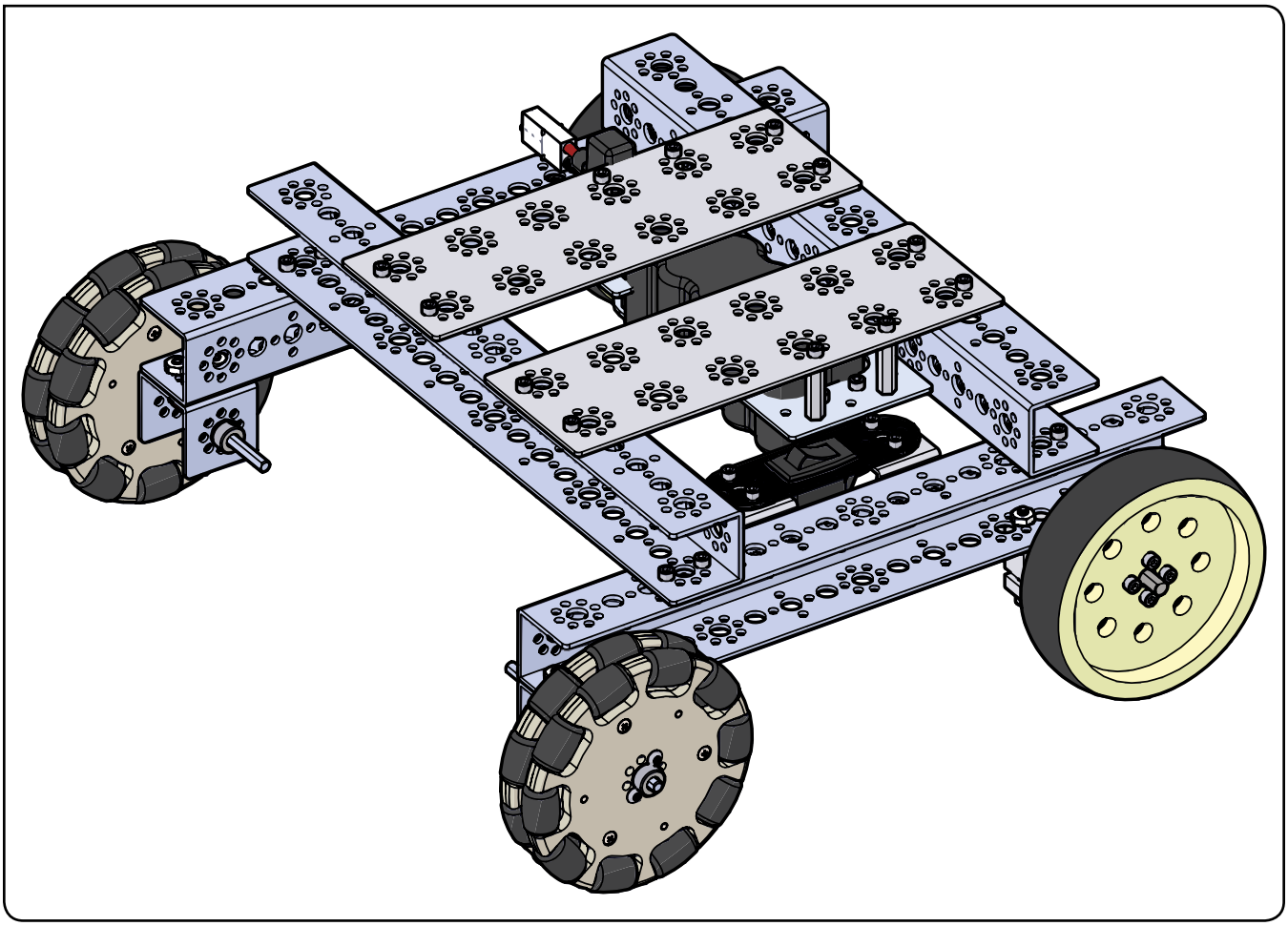
Шаг 6

Необходимые детали и принадлежности

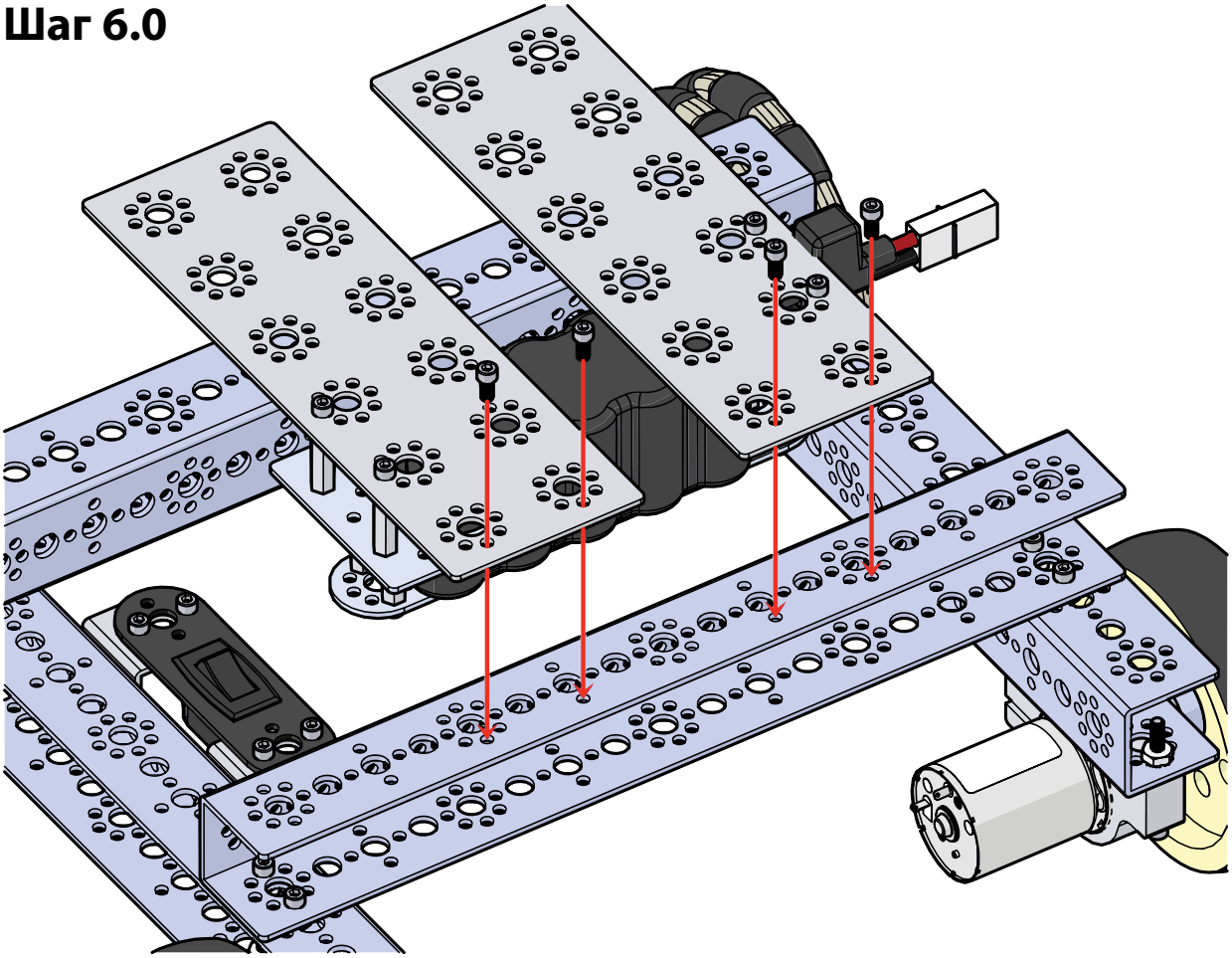
8x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

8x
Зубчатая гайка 39094

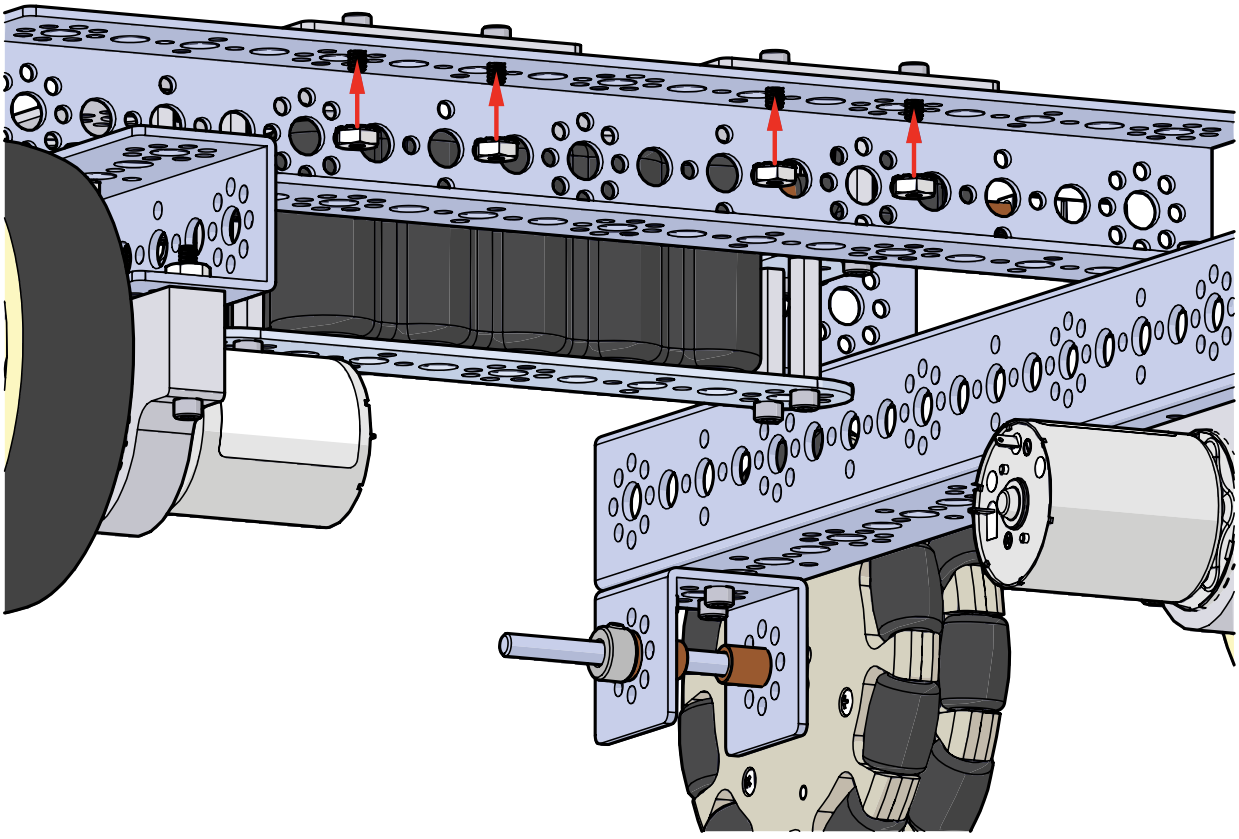
Собранная модель должна выглядеть так.



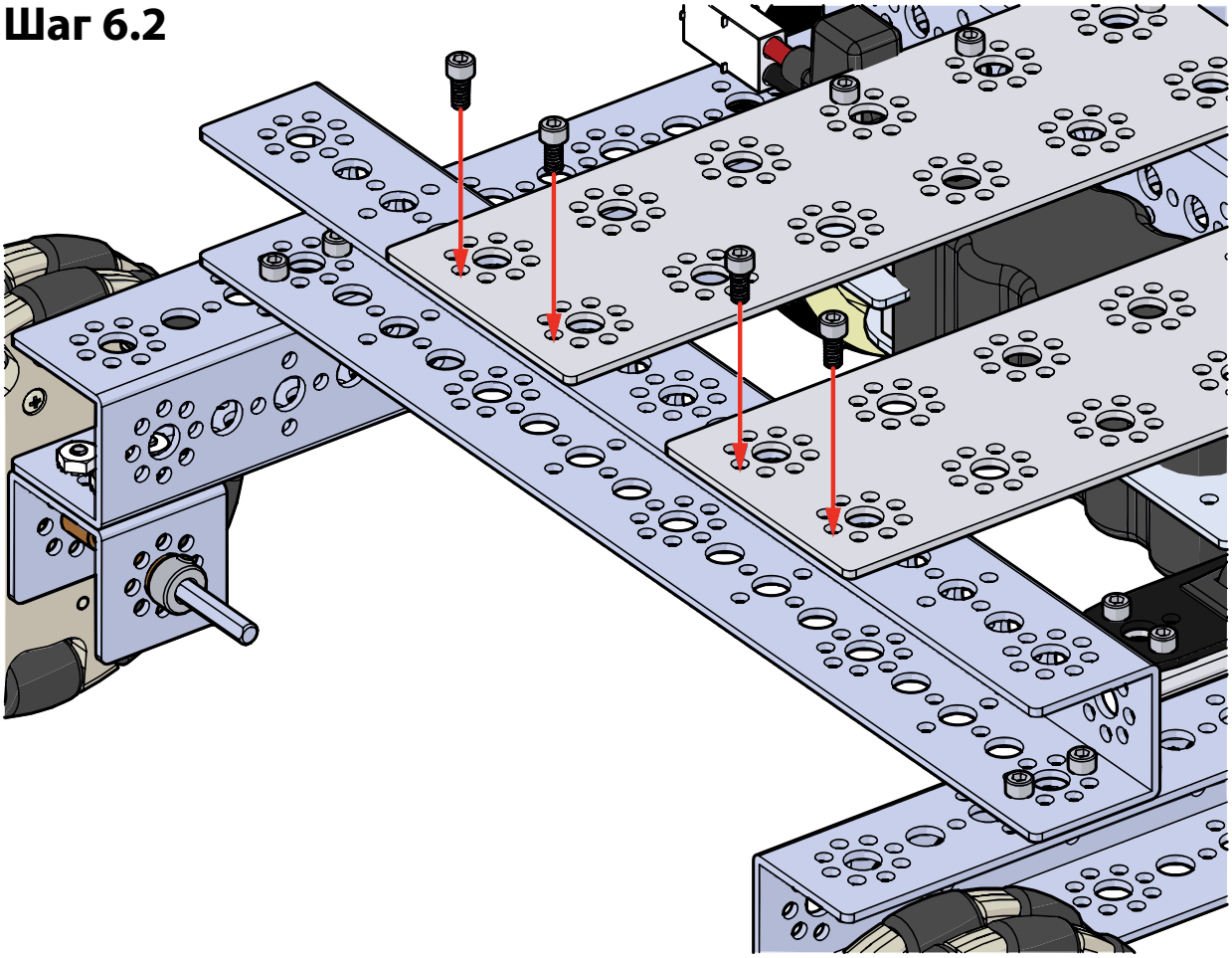
Шаг 6.0



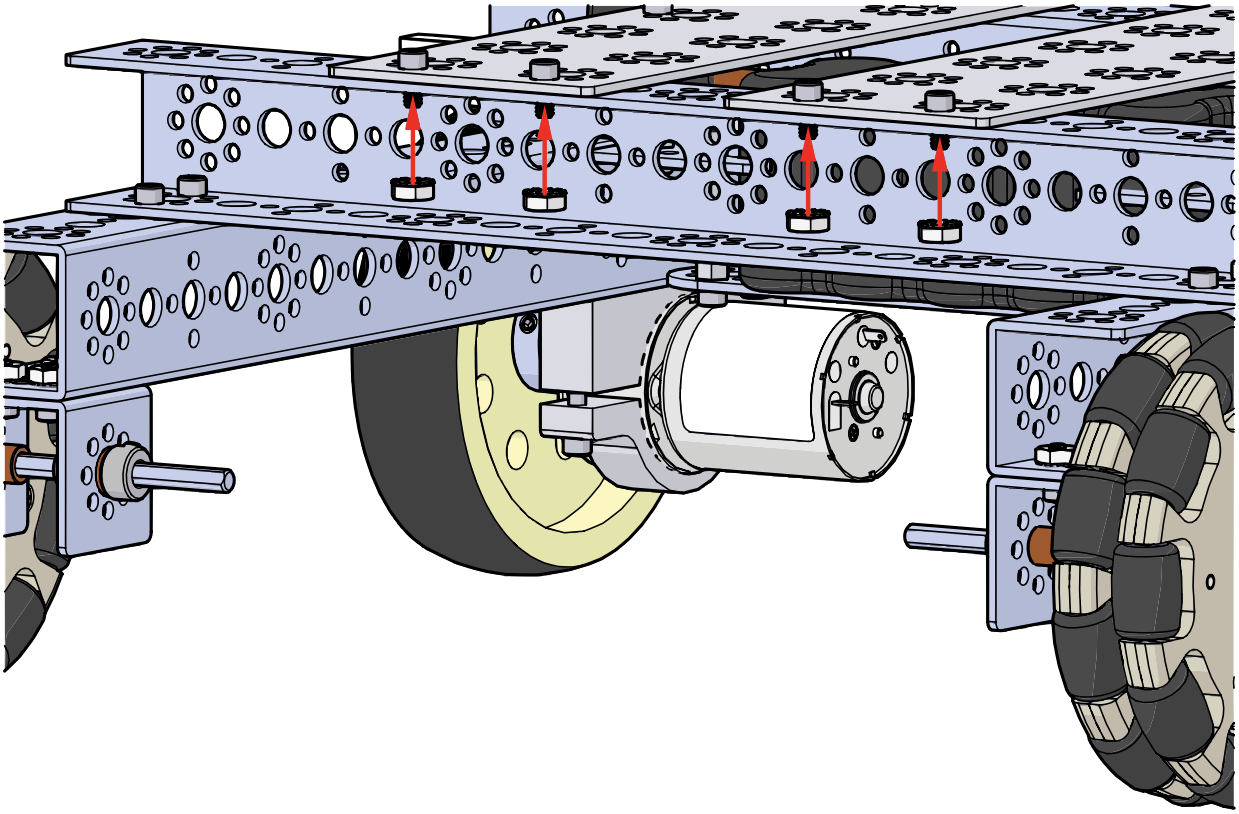
Шаг 6.1



Шаг 6.2

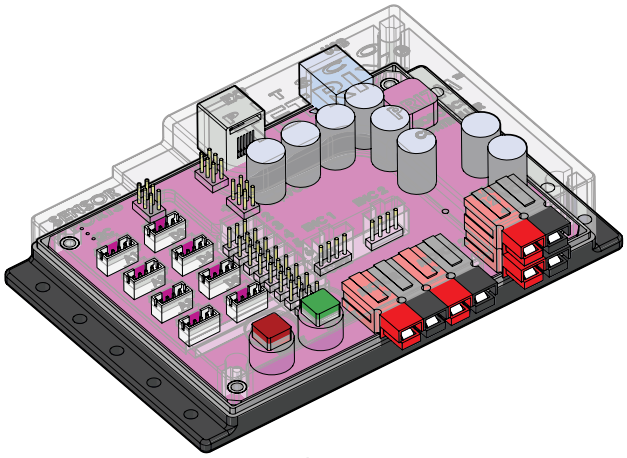


Шаг 6.3



Шаг 7

Необходимые детали и принадлежности

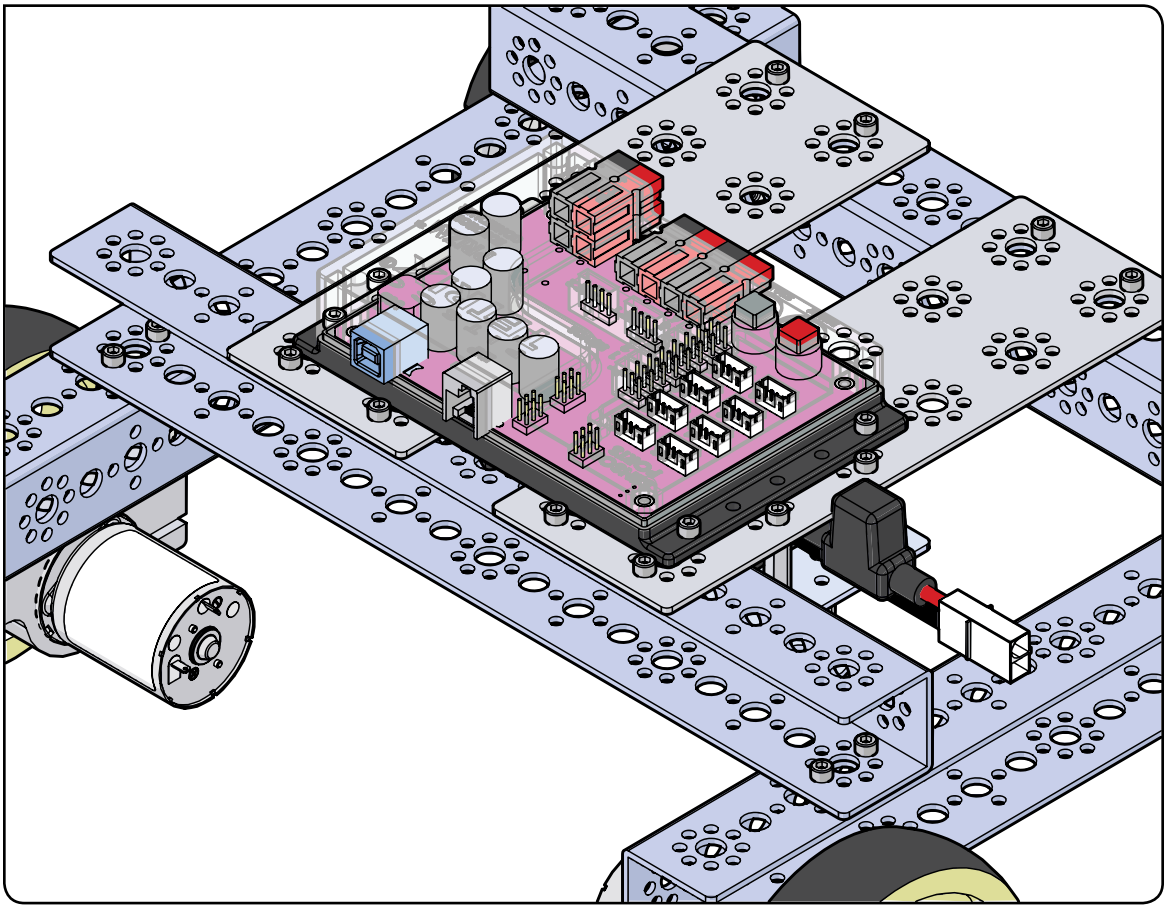


1x
Контроллер PRIZM 43000

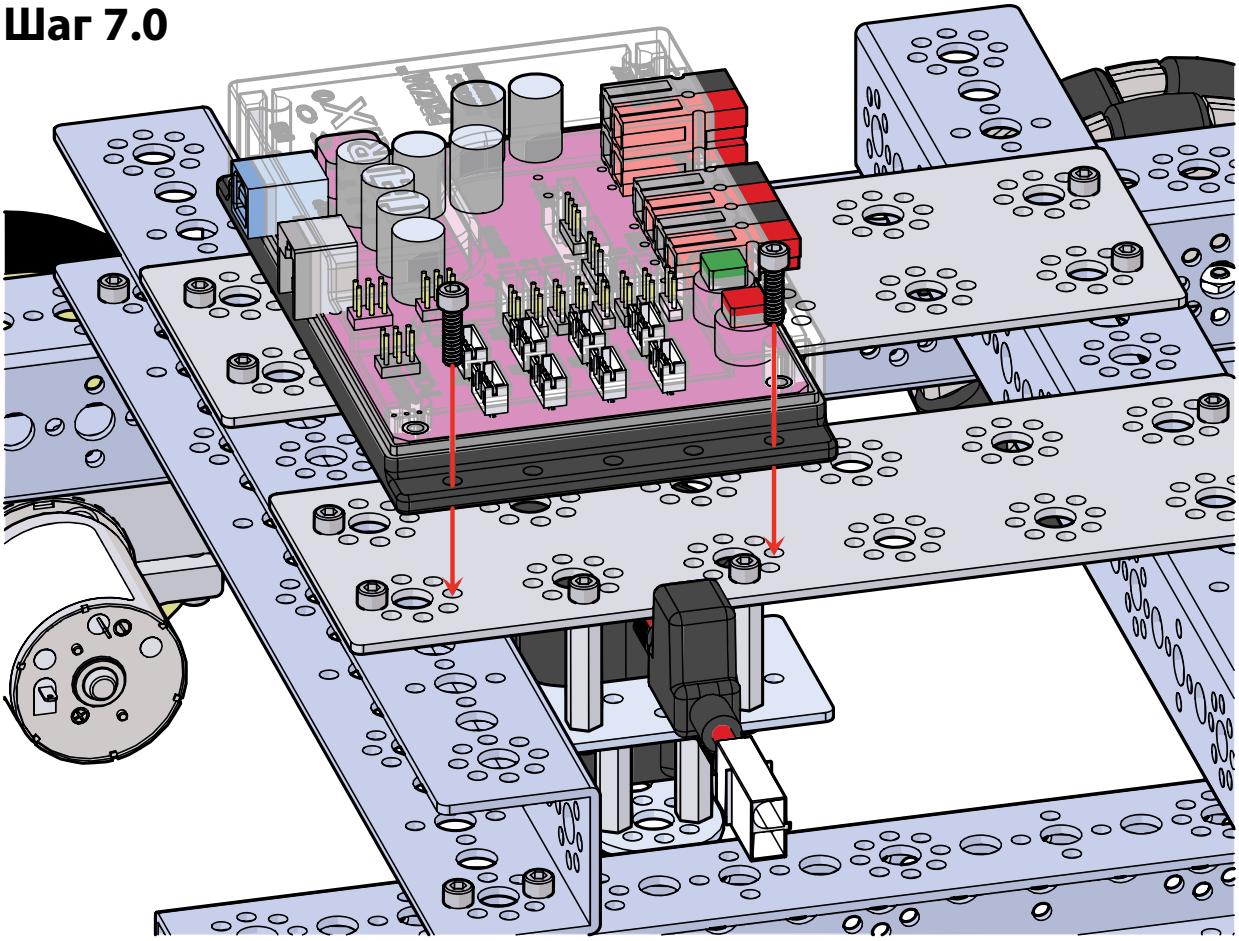
4x
Винт с головкой под торцевой ключ 6-32 x 1/2" 39097

4x
Зубчатая гайка 39094

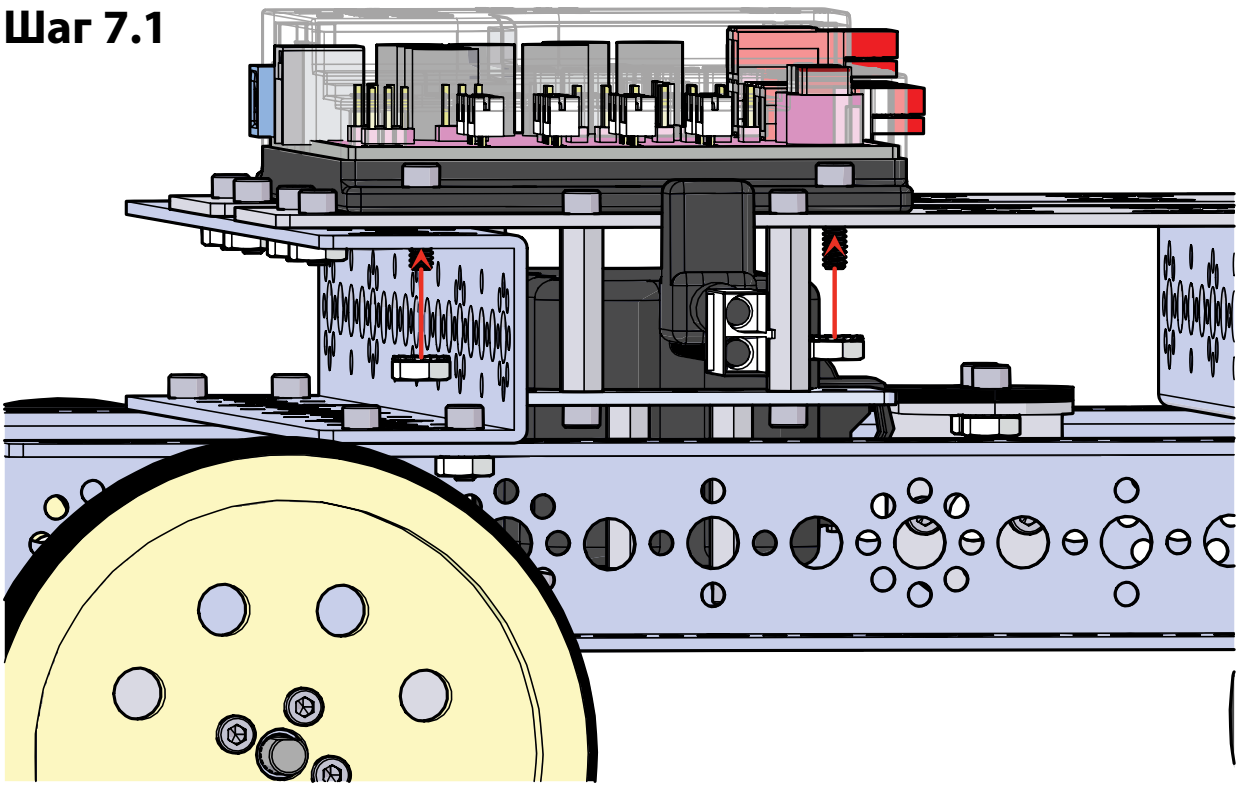
Собранная модель должна выглядеть так.



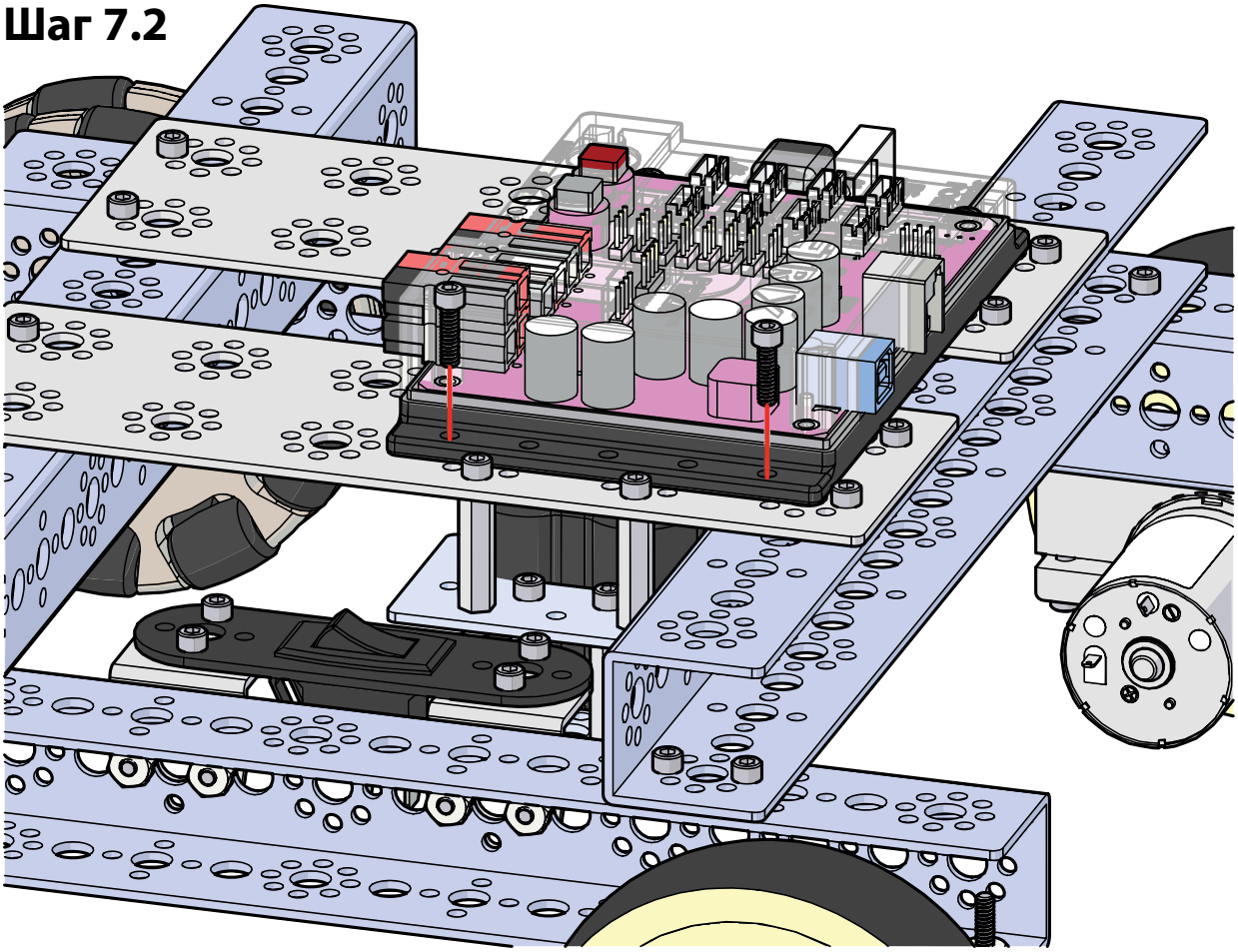
Шаг 7.0



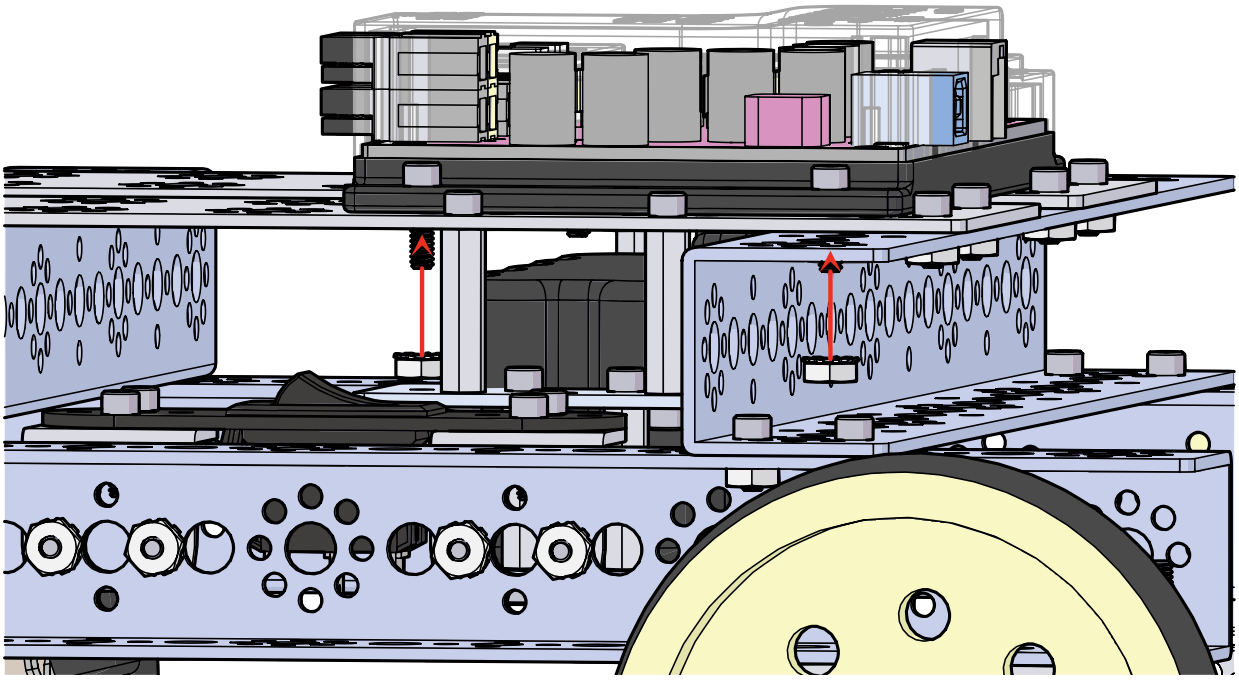
Шаг 7.1



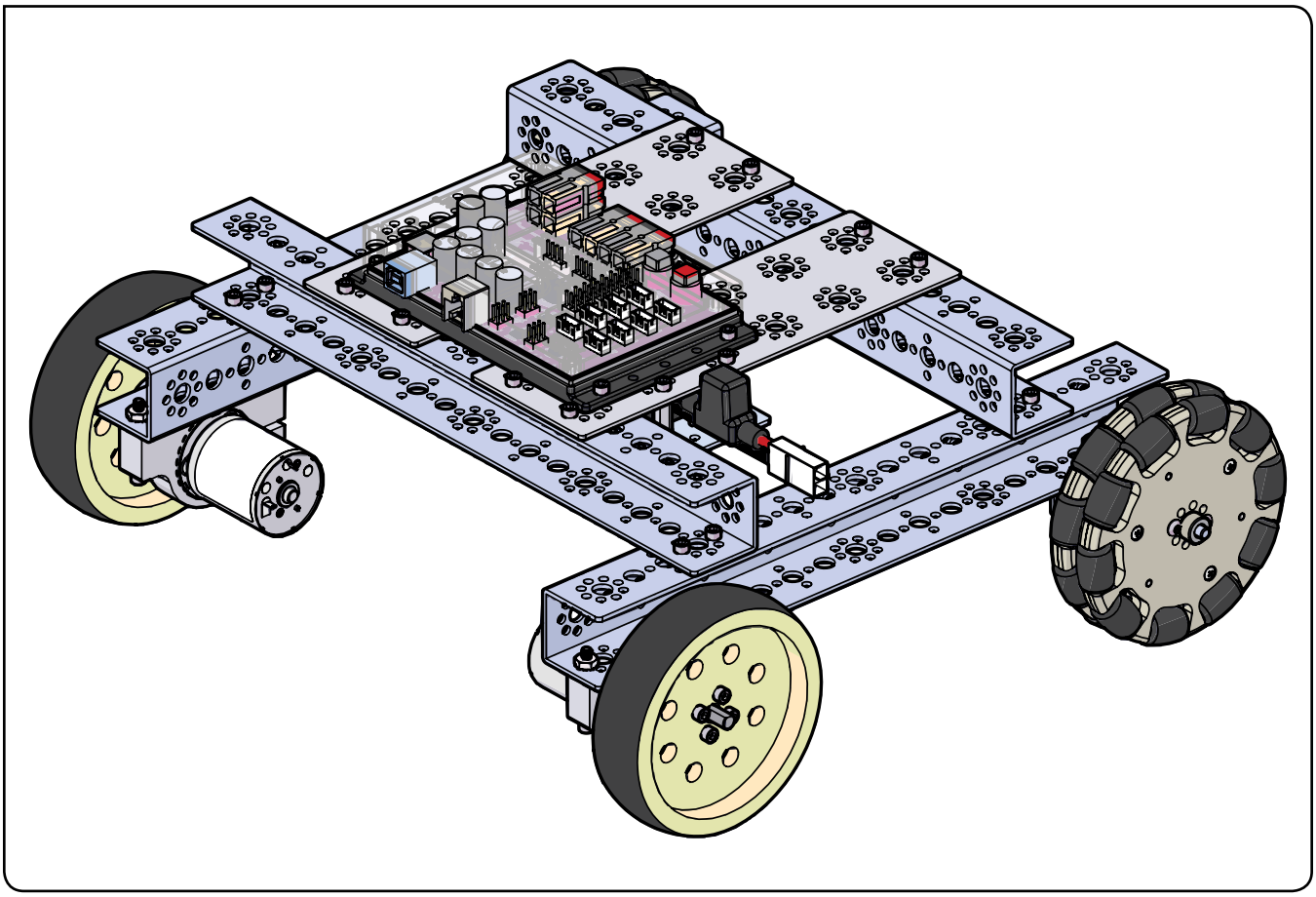
Шаг 7.2



Шаг 7.3



Собранная модель должна выглядеть так.



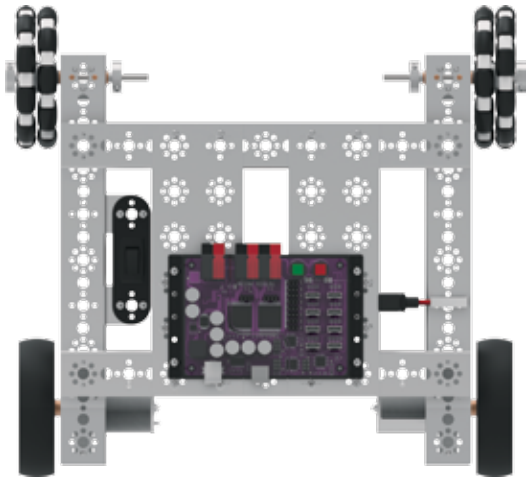
Упражнение 7: Движение передним ходом

Упражнение 7 — первое с применением робота-исполнителя. Начать следует с чего-то простого. В ходе упражнения мы создадим скетч, который заставит робота-исполнителя двинуться передним ходом, остановиться через три секунды и завершить программу.

Опираясь на усвоенное в ходе упражнения 2, мы добавим ещё один электродвигатель и заставим эту пару работать как единое целое. Одновременное использование двух электродвигателей — основополагающее требование к передвижным робототехническим моделям.

Необходимые детали и принадлежности

- Робот-исполнитель PRIZM в сборе
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act7_Drive_Forward**. Откроется окно нового скетча под названием TaskBot_Act7_Drive_Forward (рисунок 20).

Рисунок 20

Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда светодиод загорится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за направлением и продолжительностью движения робота-исполнителя. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Через 3 секунды робот-исполнитель должен остановиться.

Последующие действия

В скетче впервые появляются три новые функции: prizm.setMotorInvert(), prizm.setMotorPowers() и prizm.PrizmEnd().

Функция prizm.setMotorInvert() позволяет менять направление вращения электродвигателя на противоположное. Когда два электродвигателя стоят на противоположных сторонах шасси, функция позволяет дать обоим электродвигателям единую команду на вращение и работать согласованно. Программисту это облегчает работу. У функции два параметра. Первый обозначает канал управления электродвигателем, второй отсутствие или наличие инверсии (0 или 1).

Функция prizm.setMotorPowers() позволяет одновременно задать электродвигателю 1 и электродвигателю 2 уровень мощности. Эти два параметра задают угловую скорость каждого электродвигателя. В этом скетче мощность каждого электродвигателя равна 50 %.

Функция prizm.PrizmEnd() завершает скетч.

В этом простом скетче все функции работают сообща, заставляя робототехническую модель двинуться вперёд и через три секунды остановиться. Поскольку нам нужно, чтобы электродвигатели всегда работали вместе, функцию prizm.setMotorInvert достаточно использовать только при настройке. Всего одна функция prizm.setMotorPowers даёт обоим электродвигателям команду работать с мощностью 50 %. Чтобы завершить скетч, а не заставлять его повторяться, мы используем функцию prizm.PrizmEnd().

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 137-142: функции электродвигателя постоянного тока

Связь с настоящей жизнью

Локомотивы — очень хороший пример машин, движущихся передним ходом. Приводным двигателям локомотива (обычно дизельным) не нужно вращаться независимо для выполнения поворотов — они должны лишь сдвинуть (и двигать) локомотив вперёд. Иногда им приходится двигать его задним ходом, но для этого надо лишь переключить направление тяги.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Кинетическая энергия вращения
- Крутящий момент

Технология

- Непосредственный привод в сравнении с редукторным приводом
- Функции управления электродвигателями постоянного тока

Проектирование

- Механизм раздельного управления поворотами, механизм бортового поворота и механизм управления трёхколёсным шасси

Математика

- Вращение по часовой стрелке, против часовой стрелки
- Окружность колеса в сопоставлении с расстоянием, проходимым за один оборот колеса

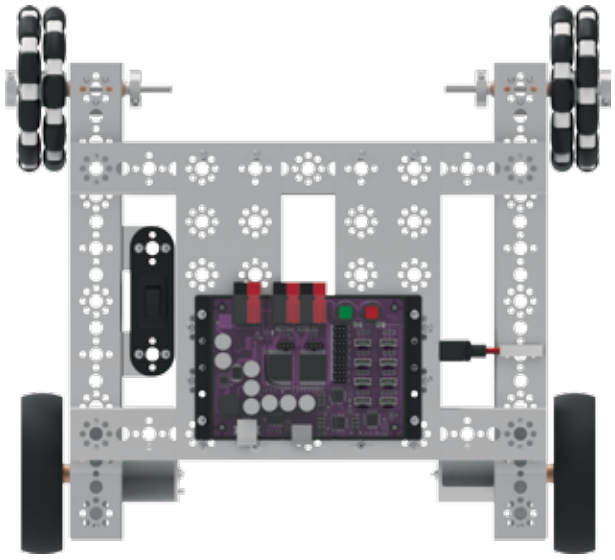
Подсказка: Без энкодеров угловые скорости электродвигателей постоянного тока, которые исполняют команды, задающие их мощность, могут быть разными в зависимости от уровня заряда аккумуляторной батареи.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 8: Движение по кругу

В восьмом упражнении нам понадобятся знания об электродвигателях, чтобы добавить роботу новое движение. Уметь двигаться по прямой важно, но этого мало — необходимо дополнить эту способность способностью выполнять повороты. В ходе упражнения робот-исполнитель будет ездить кругами вследствие регулирования мощности слаженно работающих электродвигателей.



Необходимые детали и принадлежности

- Робот-исполнитель PRIZM в сборе
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер

Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act8_Drive_Circle**. Откроется окно нового скетча под названием TaskBot_Act8_Drive_Circle (рисунок 21).

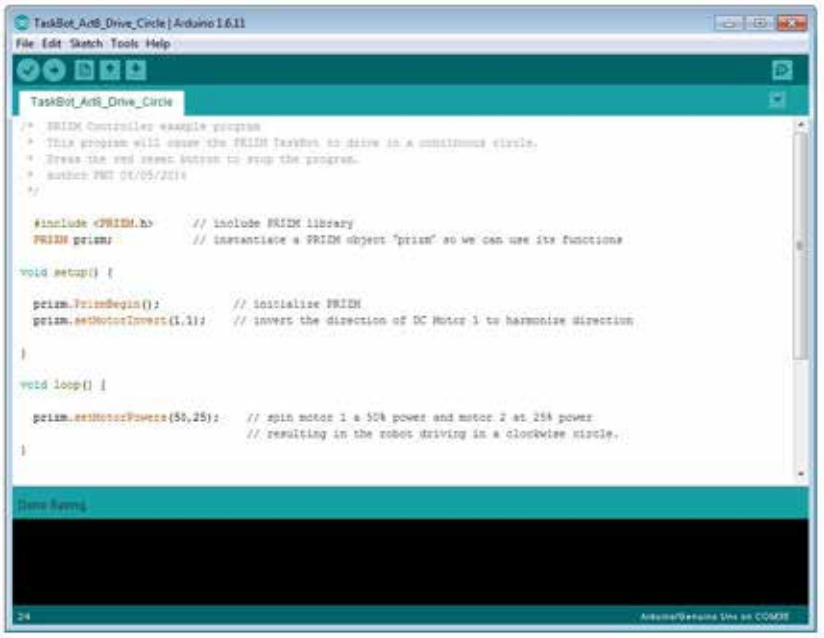


Рисунок 21

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда светодиод загорится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за направлением и продолжительностью движения робота-исполнителя.

Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Последующие действия

Хотя скетч не добавляет никаких новых функций, он должен углубить ваше понимание того, как слаженная работа электродвигателей приводит к выполнению заданного движения.

Все функции в этом скетче работают совместно, заставляя робота-исполнителя двигаться по кругу. Поскольку нам нужно, чтобы электродвигатели всегда работали вместе, функцию prizm.setMotorInvert достаточно использовать только при настройке. Всего одна функция prizm.setMotorPowers даёт обоим электродвигателям команду вращаться с разной скоростью. Одному электродвигателю задаётся 50 %, а другому 25 %, в результате получается круговое движение.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 137-142: функции электродвигателя постоянного тока

Связь с настоящей жизнью

Если вам случалось видеть поля, орошаемые дождевальной системой, то, скорее всего, она двигалась по кругу. У типичной дождевальной системы есть поворотный центр, к которому подсоединён источник воды, подаваемой по крылу из трубопроводов к разбрызгивающим отверстиям по всей длине крыла. Крыло опирается на колёса, расположенные приблизительно через каждые десять метров, а колёса приводятся в движение электродвигателями. Поворотный центр системы остаётся на месте, вращающиеся колёса перемещают всё крыло по кругу, по пути орошая окрестные поля.

Подсказка: Чтобы робот мог двигатель по кругу, понадобится от полутора до двух квадратных метров свободного места на полу.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Круговое движение
- Движение по касательной

Технология

- Управление электродвигателями и точность перемещения
- Решение задач при помощи программирования

Проектирование

- Конструкция машины, способной двигаться по криволинейным траекториям

Математика

- Техническое (математическое) определение круга
- Вычисление радиуса

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя двигаться по кругу другого размера. Вспомните, чему мы научились, выполняя предыдущие упражнения, и попробуйте увеличить или уменьшить диаметр круга, меняя параметры.

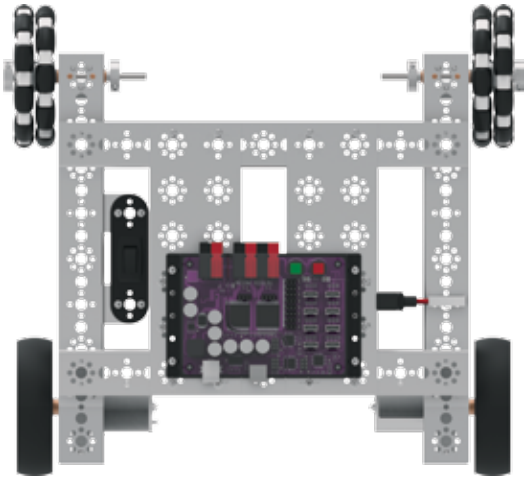
Поставьте себе задачу увеличить число движений. Что нужно сделать, чтобы заставить робототехническую модель двигаться по овалу или восьмёрке?

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 9: Движение по квадрату

Продолжим совершенствовать способность робота-исполнителя двигаться по разным траекториям и научим его выполнять повороты под углом 90 градусов. Способность поворачивать под прямым углом понадобится роботу-исполнителю при движении по квадрату.



Необходимые детали и принадлежности

- Робот-исполнитель PRIZM в сборе
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер

Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act9_Drive_Square_1**. Откроется окно нового скетча под названием TaskBot_Act9_Drive_Square_1 (рисунок 22).

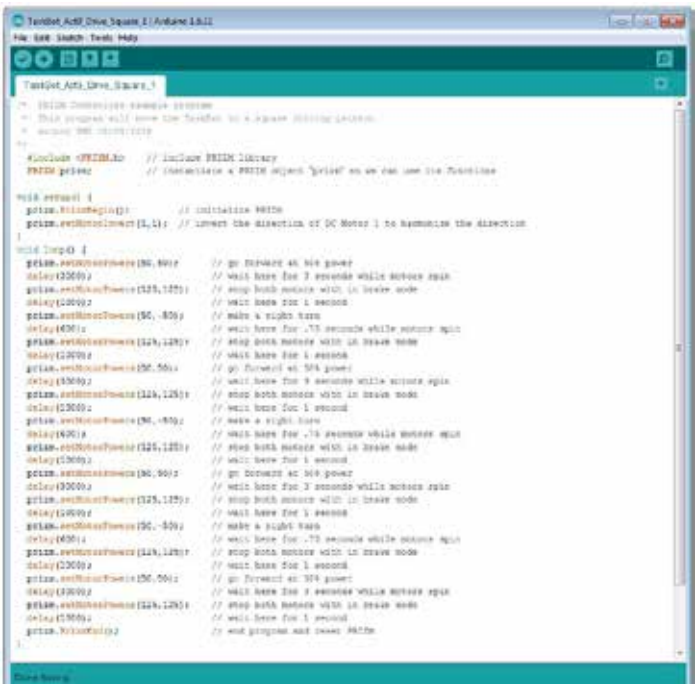


Рисунок 22

Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за направлением и продолжительностью движения робота-исполнителя. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Последующие действия

У скетча широкие возможности, но составлен он из функций, которые мы уже использовали. Мы только что объединили несколько простых последовательных действий в одно большое и сложное, в данном случае — движение по квадрату.

Применительно к этой программе важно понять, что мы используем счисление пути, чтобы запрограммировать движение робота по квадрату. Счисление пути попросту означает управление работой электродвигателя, исходя из времени. Так, чтобы повернуть направо, мы даём электродвигателям команду вращаться в противоположных направлениях с 50%-й мощностью в течение 600 миллисекунд.

Мы предполагаем, что при вращении электродвигателей в течение определённого времени на определённых оборотах машина повернёт вправо почти под прямым углом. Но предположения не всегда точны, потому что заряд аккумуляторной батареи нашего робота меняется, и любая пробуксовка колеса на рабочей поверхности приведёт к ошибочному результату. По сути мы счисляем путь для поворота вправо.

Все функции в этом скетче работают совместно, заставляя робота-исполнителя двигаться по квадрату. Поскольку нам нужно, чтобы электродвигатели всегда работали вместе, функцию prizm.setMotorInvert достаточно использовать только при настройке. Всего одна функция prizm.setMotorPowers даёт обоим электродвигателям команду вращаться с разной скоростью.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 137-142: функции электродвигателя постоянного тока

Подсказка: Эта программа относится к тем, на точность выполнения которых способен повлиять заряд аккумуляторной батареи. Почему? Задаваемая этой программой геометрическая форма обусловлена использованием ряда функций задержки и, поскольку частота вращения электродвигателей меняется из-за уровня заряда аккумуляторной батареи, форма квадрата может несколько нарушиться в результате разрядки аккумуляторных батарей. Это одно из следствий способа управления машинами, в основу которого положено "счисления пути".

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Взаимосвязь времени, расстояния и скорости

Технология

- Способ измерения времени в микропроцессоре
- Измерение радиуса поворота

Проектирование

- Разработка программы для движения по заданной траектории

Математика

- Взаимосвязь сторон и углов квадрата
- Измерение углов

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя двигаться по траектории в виде квадрата. Вспомните, чему мы научились, выполняя предыдущие упражнения, и попробуйте увеличить или уменьшить квадрат, меняя параметры.

Поставьте себе задачу составить программу для следования по траекториям более сложным, чем квадрат. Попробуйте провести робота по траекториям, повторяющим написание букв алфавита, вывести его из простого лабиринта, используя счисление пути.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

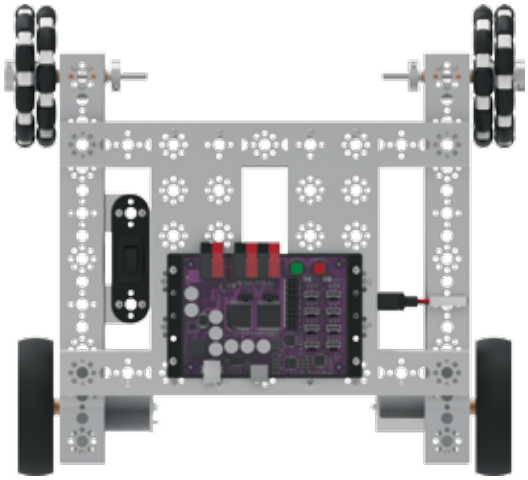
Примечание: Помните про это упражнение. Мы вернёмся к описанному действию в следующем упражнении, чтобы совершенствовать свои познания и умения в программировании.

Упражнение 10: Упрощение движения по квадрату

Помните задание с квадратом из упражнения № 9? Для следующего упражнения хотим показать вам более совершенный способ запрограммировать то же действие.

Необходимые детали и принадлежности

- Робот-исполнитель PRIZM в сборе
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act10_ Drive_Square_2**. Откроется окно нового скетча под названием TaskBot_Act10_Drive_Square_2 (рисунок 23).

Рисунок 23

Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за направлением и продолжительностью движения робота-исполнителя. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Последующие действия

В скетче используется новая структура программы в виде оператора цикла "for" (оператора цикла со счётчиком); он даёт новый способ выполнения функций при помощи пользовательских функций. Объединение двух этих нововведений упростит составление и чтение скетча.

Оператор цикла "for" используется для повторения блока кодов (блока команд), заключённого в фигурные скобки. Для подсчёта повторений и завершения цикла используется счётчик итераций. Оператор цикла "for" помогает ограничить число повторений цикла.

Пользовательские функции объявляют вне функций setup и loop. В скетче их используют чаще всего, когда требуется много раз повторить одно и то же действие.

Чтобы заставить робота-исполнителя ездить по квадрату в 9-м упражнении, мы составили перечень действий, разместив каждое на отдельной строке. В результате у нас получился громоздкий программный код, который выглядел сложнее, чем на самом деле. В сложном действии мы можем выделить конкретные простые действия, которые несколько раз повторялись. Этим выделенным действиям мы можем дать определение в вызываемой функции за пределами цикла(). После того, как вызываемые функции получат определение за пределами цикла(), мы сможем использовать их в цикле(), чтобы упростить код.

Для этого скетча мы определили, что вне цикла() находятся функции "forward()", то есть "передний ход()" и "rightTurn()", то есть "поворот вправо()". Определив эти две функции, мы теперь можем вызвать их в цикле(). Мы запускаем цикл() при помощи оператора цикла со счётчиком, чтобы задать число повторений цикла.

У оператора цикла со счётчиком три части. Первая — инициализация: int x=0. Она используется первой и только один раз.

Вторая — условие: x<=3. Условие проверяется при каждом повторении цикла(). Если условие истинно, то выполняется третья часть: x++ — приращение, после чего условие проверяется снова. Когда условие становится ложным, цикл() завершается.

Проще говоря, инициализация запускает цикл(). Условие определяет, сколько раз надлежит повторить цикл, а именно: три раза плюс инициализация в самом начале. Всего цикл() будет выполнен четыре раза. А приращение определяет, как отсчитывается каждое выполнение цикла(), соответствующее одному приращению.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, читайте следующий раздел в приложении: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 137-142: функции электродвигателя постоянного тока

Связь с настоящей жизнью

Учась выполнять свободные броски, вы настраиваете мозг и мышцы на повторение определённого набора управляемых действий, в результате которых баскетбольный мяч должен пролететь через кольцо. Воспроизводимость результата или способность раз за разом получать тот же или очень близкий результат, повторяя некое действие, имеет решающее значение во многих видах спорта, но не менее решающую роль играет в проектировании и программировании роботов. Определение способа (или программного кода), который заставит робота двигаться точно по квадрату, доказывает способность робота к точному повторению действия или повторению с небольшим отличием.

Подсказка: Функцию можно задать в одном месте скетча, а потом дать на неё ссылку или "вызвать" её в другом месте. Так образуются подпрограммы, благодаря которым скетч остаётся небольшим.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Взаимосвязь времени, расстояния и скорости

Технология

- Способ измерения времени в микропроцессоре
- Измерение радиуса поворота

Проектирование

- Разработка программы для движения по заданной траектории

Математика

- Взаимосвязь сторон и углов квадрата
- Измерение углов

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя двигаться по траектории в виде квадрата. Используйте для этого оператор цикла со счётчиком и вызываемые функции. Или в качестве дополнительного задания составьте программы управления движением по сложным траекториям в виде прямоугольников или шестиугольников.

Вспомните, чему мы научились, выполняя предыдущие упражнения. Теперь задача заключается в том, чтобы научиться строить собственные, не имеющие аналогов робототехнические модели, и находить новые и увлекательные способы применить усвоенные знания по программированию.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

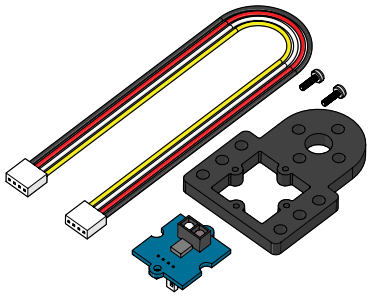
Вводное задание по конструированию: Добавляем роботу-исполнителю сообразительности!

Пока что наш робот замечательно выполняет команды, но принимать какие-то решения ещё не умеет. Благодаря начальным упражнениям мы знаем, что датчики позволяют роботам воспринимать окружающий мир. Можно наделить роботов способностью принимать решения, основываясь на предоставляемой датчиками информации, — и казаться разумными.

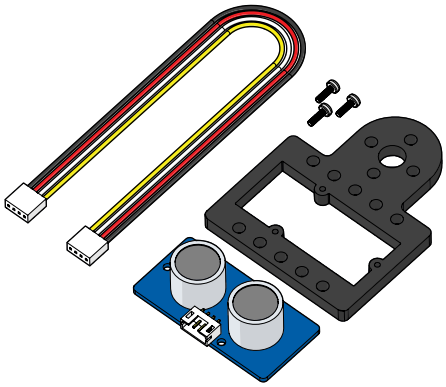
В следующих примерах с программными кодами мы прикрепим датчики к роботу-исполнителю PRIZM. Разберём два примера с датчиком линии и два примера с ультразвуковым датчиком. Указания по закреплению датчиков на роботе-исполнителе PRIZM см. ниже.

Шаг 1

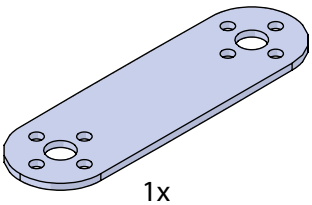
Необходимые детали и принадлежности



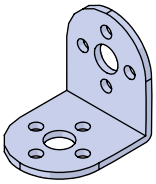
1x
Комплект датчика линии 43056



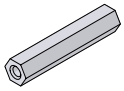
1x
Комплект ультразвукового датчика 43055



1x
Плоская скоба 39061



1x
Г-образная скоба 39062



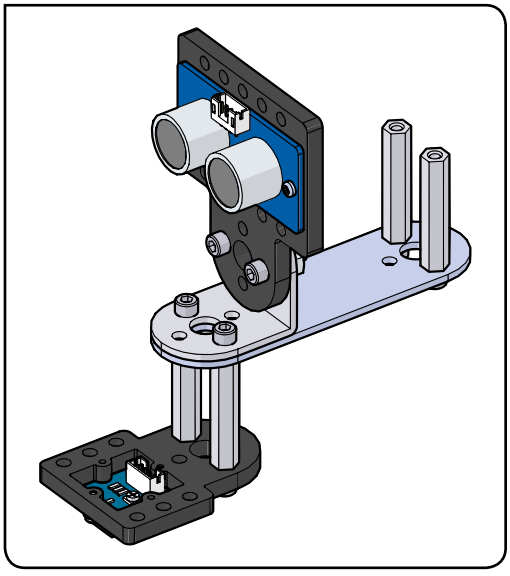
4x
Распорная стойка 6-32 x 32 мм 39107

6x
Винт с головкой под торцевой ключ 6-32 x 1/2" 39097

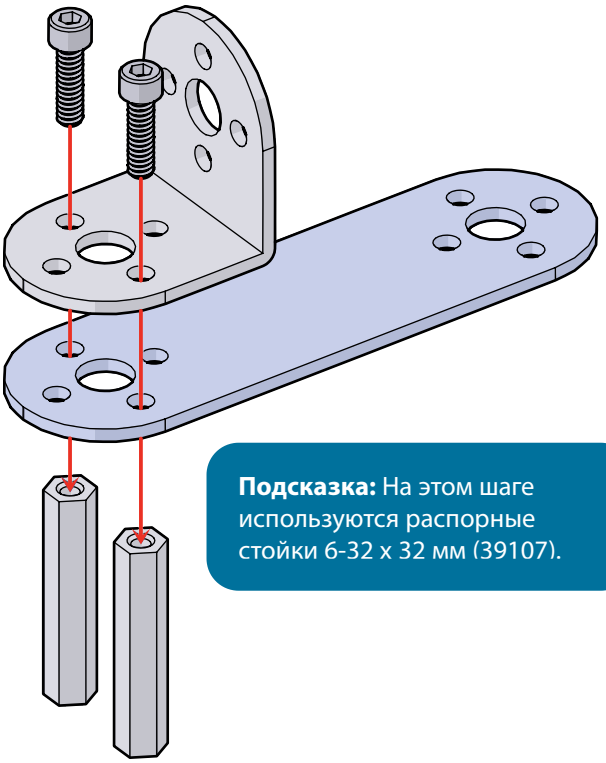
2x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

2x
Зубчатая гайка 39094

Собранная модель должна выглядеть так.

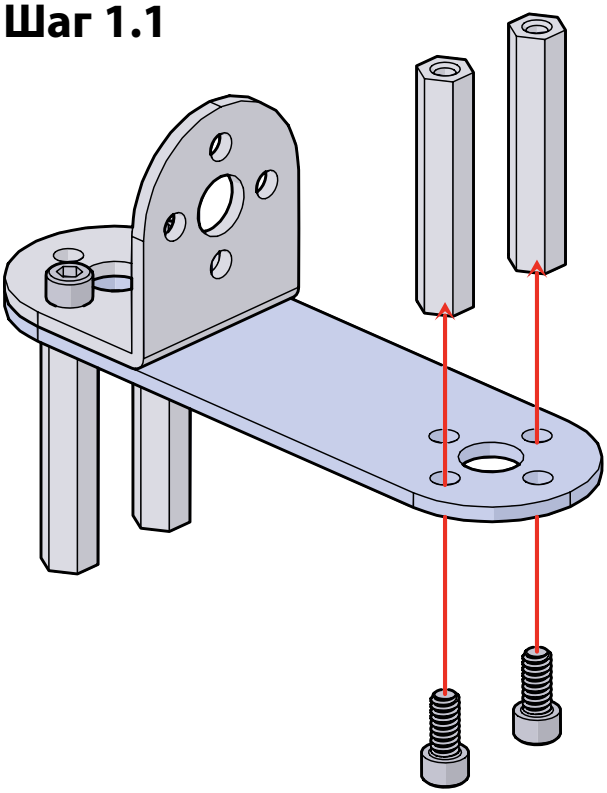


Шаг 1.0



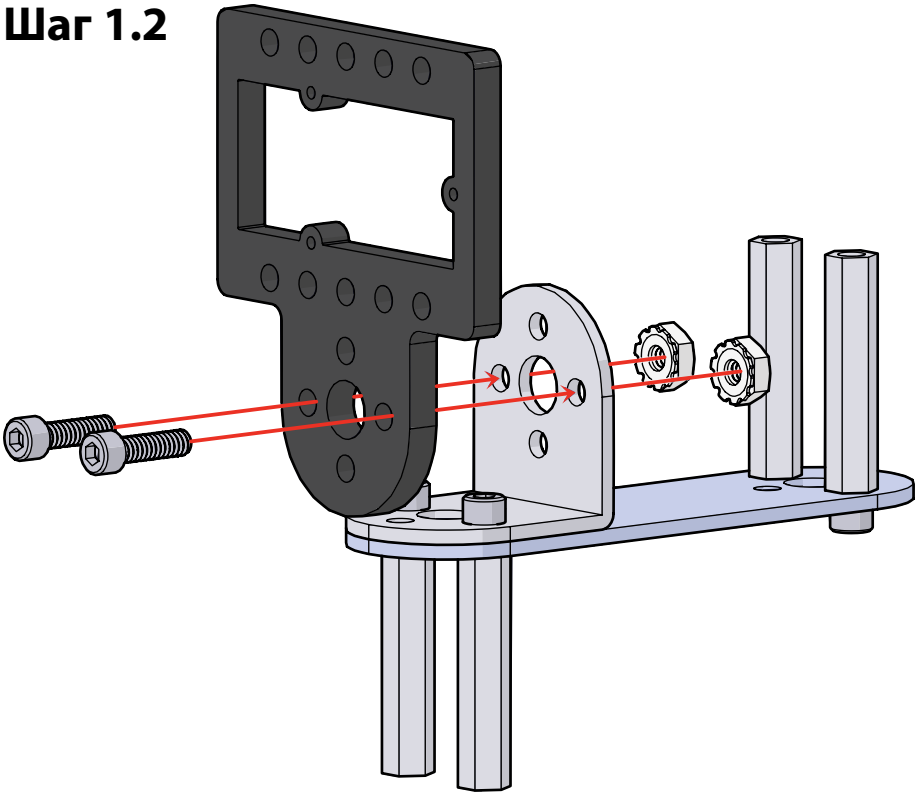
Подсказка: На этом шаге используются распорные стойки 6-32 x 32 мм (39107).

Шаг 1.1

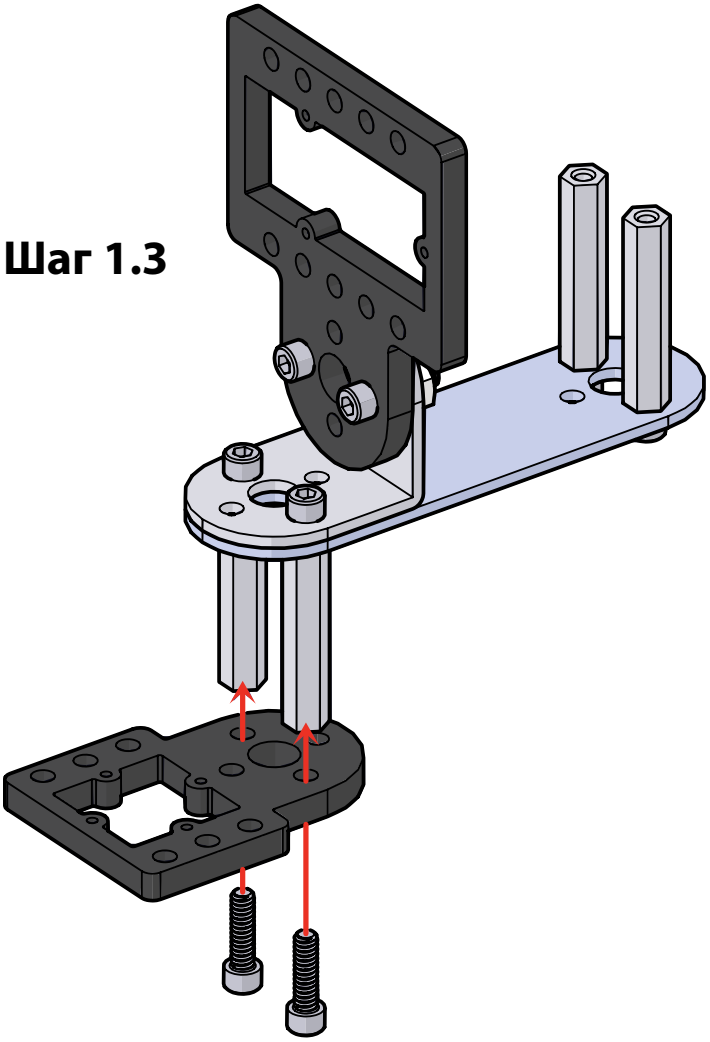


Подсказка: На этом шаге используются распорные стойки 6-32 x 32 мм (39107).

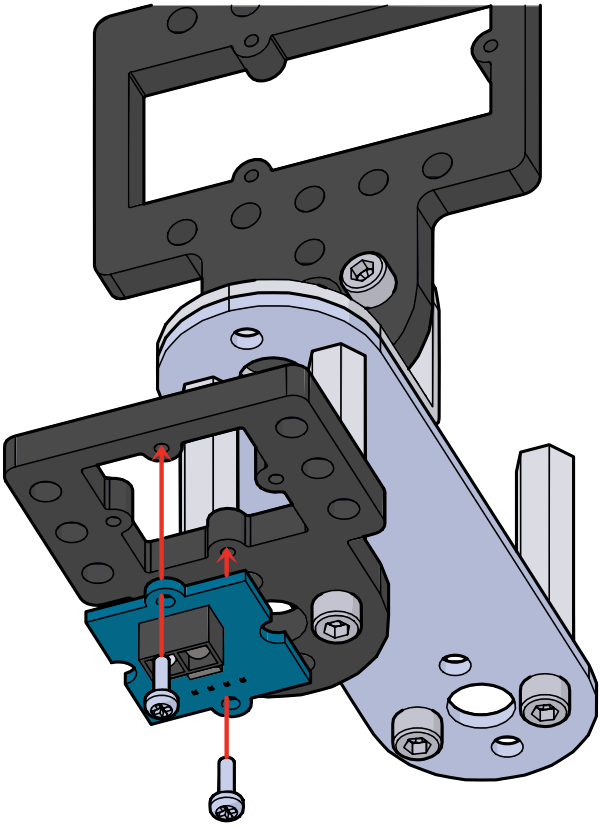
Шаг 1.2



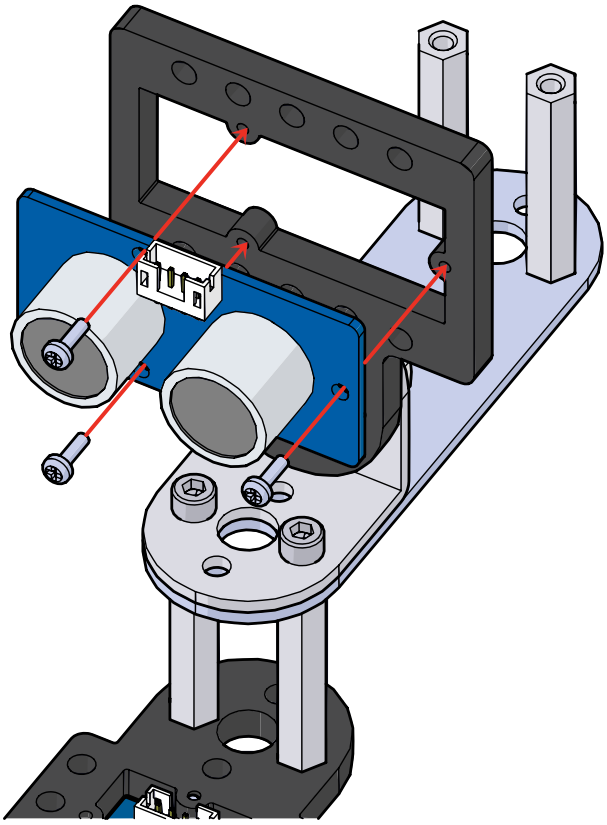
Шаг 1.3



Шаг 1.4



Шаг 1.5

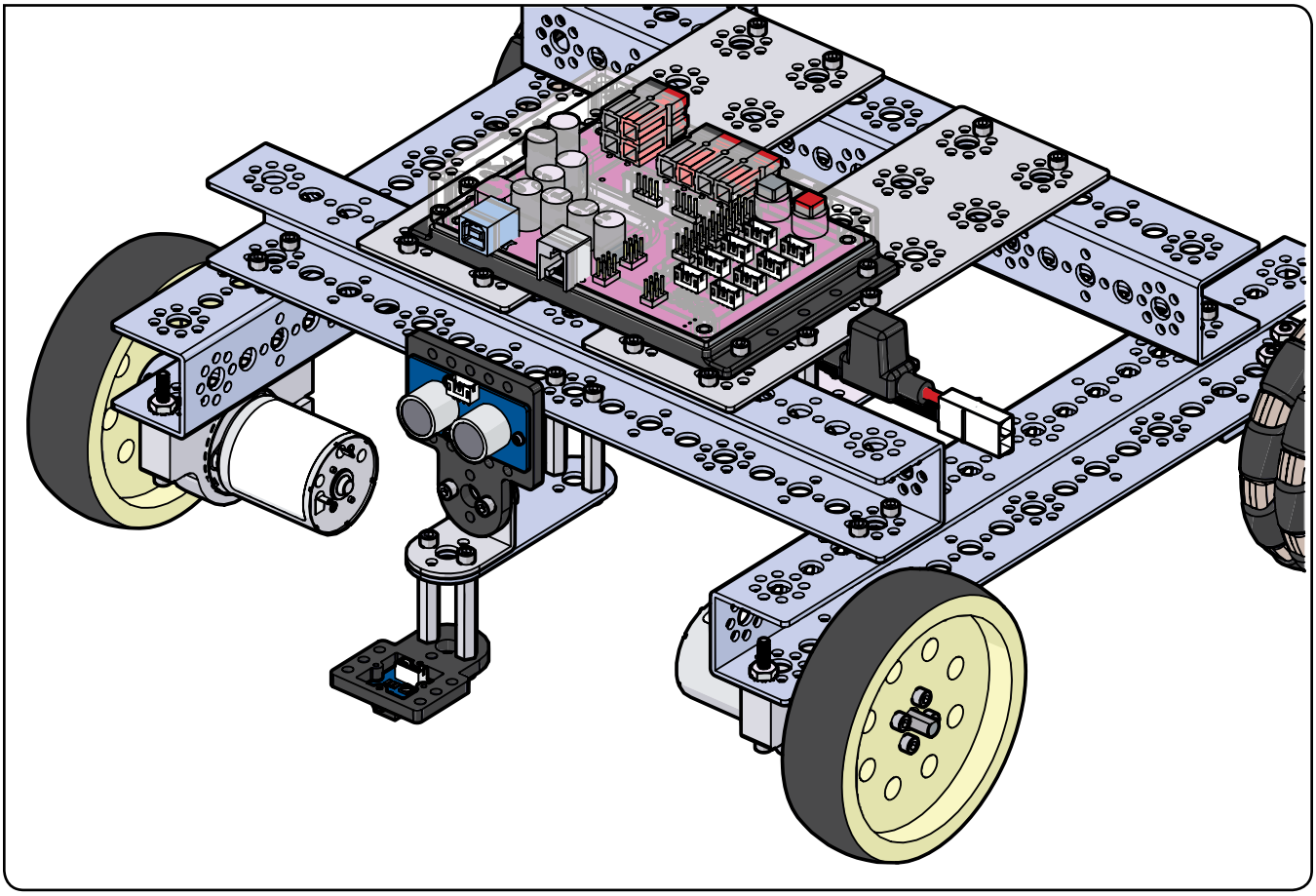


Шаг 2

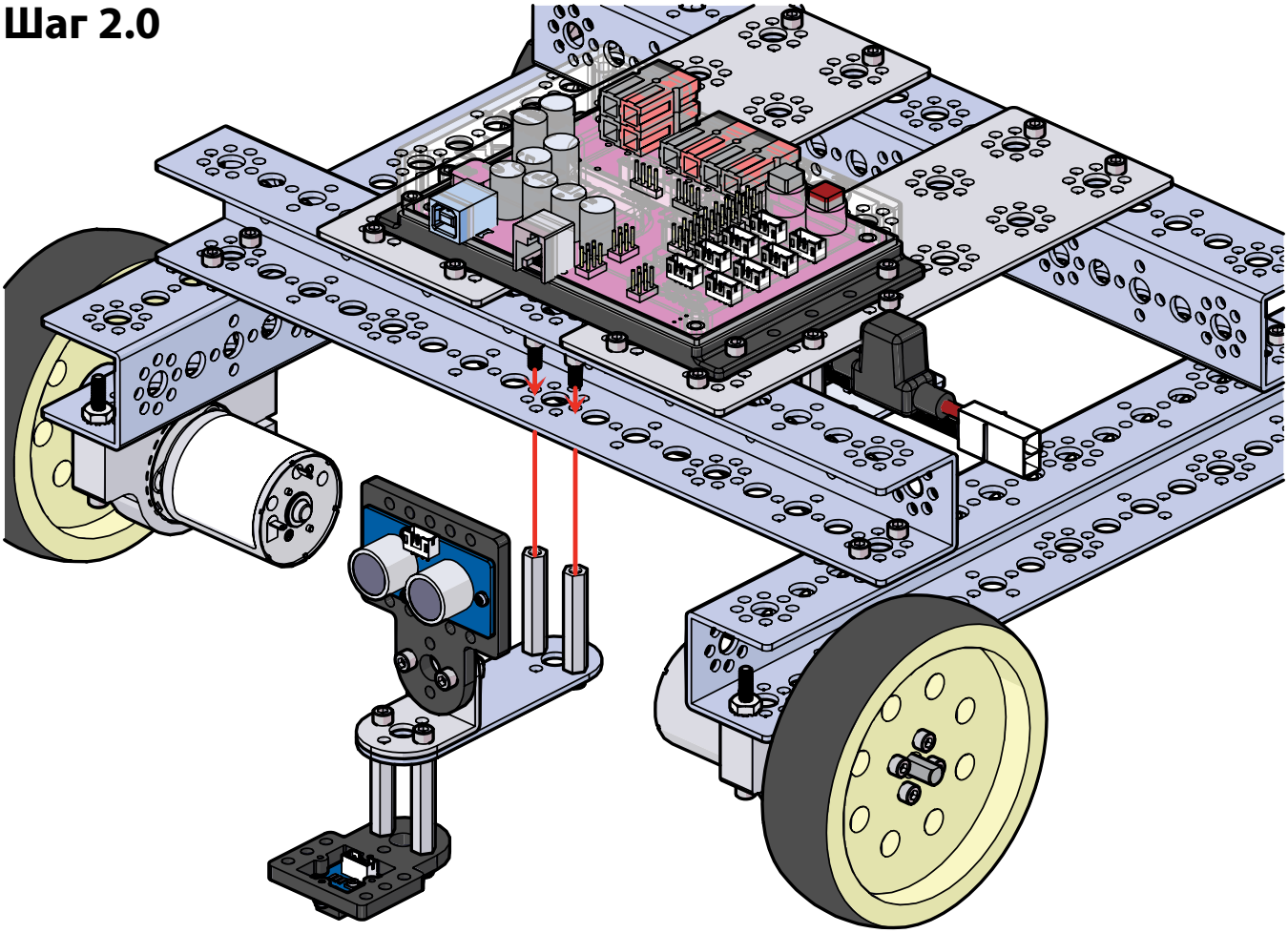
Необходимые детали и принадлежности

2x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

Собранная модель должна выглядеть так.



Шаг 2.0

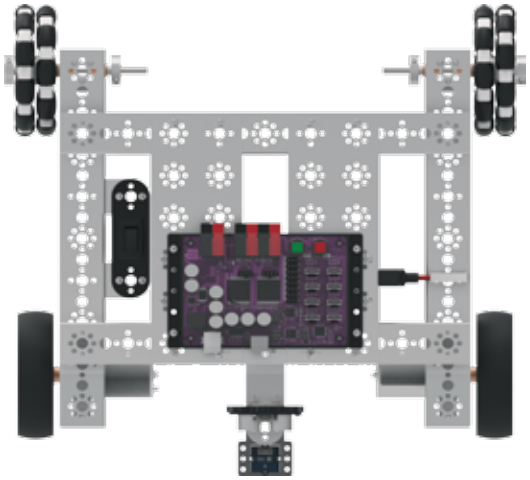


Упражнение 11: Подъезд к линии и остановка

Начав с простого, как и во всех наших упражнениях, мы добавим к простейшему умению робота двигаться передним ходом датчик линии. Тогда робот-исполнитель сможет останавливаться, обнаружив в окружающем мире заданные условия. В этом упражнении робот-исполнитель должен проехать вперёд и остановиться у линии.

Необходимые детали и принадлежности

- Светлая поверхность с контрастными темными элементами
- Полностью собранный робот-исполнитель PRIZM с датчиком
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act11_ Drive_To_Line**. Откроется окно нового скетча под названием TaskBot_Act11_Drive_To_Line (рисунок 24).

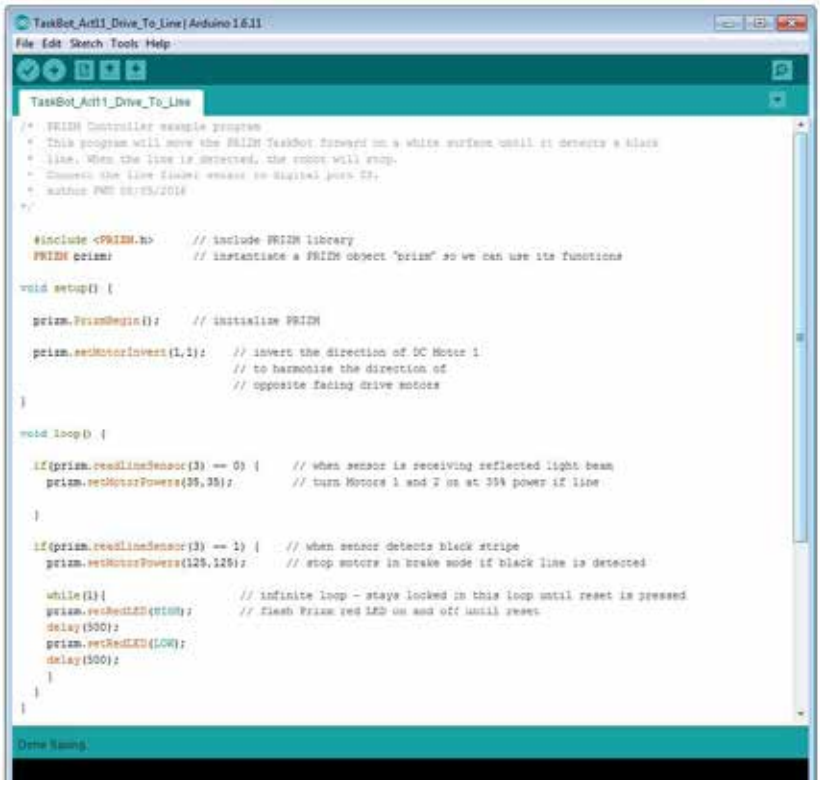


Рисунок 24

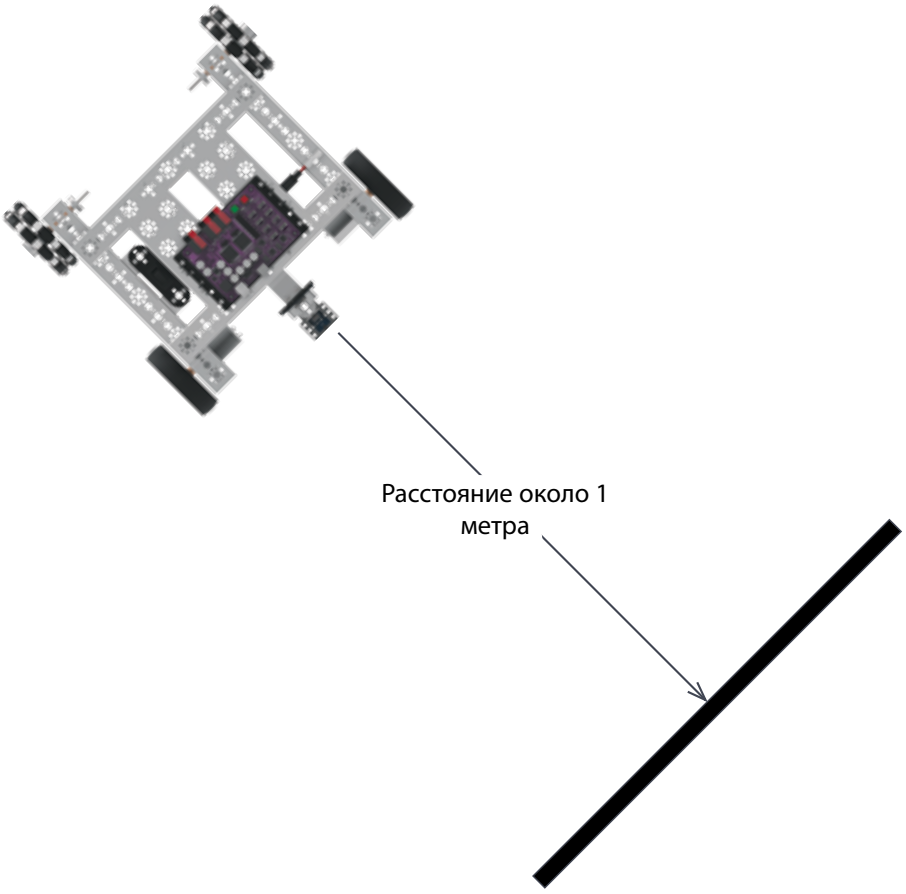
Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Датчик линии необходимо подсоединить к порту № 3 для цифровых датчиков.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Робота-исполнителя надо поставить на белую или светоотражающую поверхность, развернув в сторону чёрной линии или поверхности, которая не отражает свет, на расстоянии около метра.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за действиями робота. Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?



Поиск и устранение неисправностей: Убедитесь, что датчик линии вставлен в предусмотренный для него порт и правильно настроен на обнаружение линии. Возможно, понадобится слегка подправить высоту или чуть сдвинуть регулировочный винтик на тыльной стороне датчика. Чтобы проверить исправность датчика, вручную проведите модель вперёд-назад по белой поверхности и над чёрной линией. Пока датчик линии находится над белой поверхностью, на нём должен гореть красный светодиод, а когда датчик оказывается над чёрной линией, светодиод должен гаснуть.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Последующие действия

В скетче используется два условных оператора (такую структуру программы мы использовали в упражнении № 4), и впервые появляется оператор цикла с предварительным условием (цикл "while").

Цикл "while" учитывает проверяемое условие в круглых скобках и повторяет цикл до тех пор, пока это проверяемое условие в круглых скобках не станет ложным.

В этом скетче условные операторы учитывают условие, связанное с датчиком линии. Первый условный оператор даёт роботу-исполнителю команду понизить мощность электродвигателей до 35%, оказавшись над белой или светоотражающей поверхностью. У второго условного оператора две части. Первая командует роботу-исполнителю затормозить над чёрной линией или над поверхностью, не отражающей свет. Во второй части есть цикл с предусловием, который останавливает выполнение скетча и заставляет мигать красный светодиод до тех пор, пока скетч не будет перезапущен.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков
- Страницы 137-142: функции электродвигателя постоянного тока

Связь с настоящей жизнью

Это типичный случай для водителей, подъезжающих к пешеходному переходу. Ради безопасности пешеходов необходимо остановить машину у края пешеходного перехода или не доезжая до него. После некоторой тренировки это входит у водителей в привычку. Но роботы учатся иначе, поэтому им нужны автоматические приборы и датчики, чтобы знать, когда и где остановиться.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Формула скорости
- Постоянное ускорение при прямолинейном движении

Технология

- Циклы с предусловием (циклы "while")
- Считывание показаний датчика

Проектирование

- Применение правил решения задач

Математика

- Вычисление расстояния, времени и скорости
- Логические операторы (например, "если...то")

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя подъехать к чёрной линии и остановиться. Вспомните, чему мы научились, выполняя предыдущие упражнения.

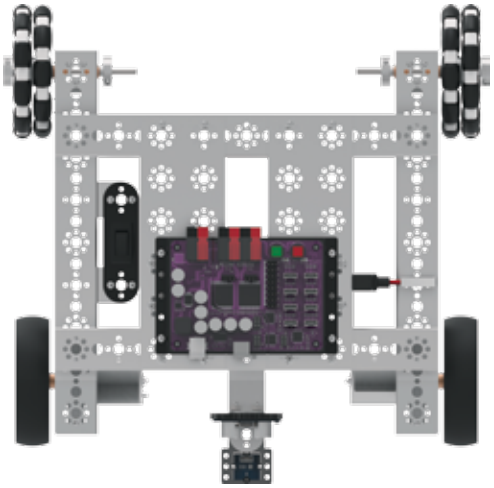
Поставьте себе задачу создать программы, которые заставят робота-исполнителя, подъехавшего к линии, двигаться иначе, например: отъехать от неё задним ходом или повернуть в другом направлении.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 12: Движение по линии

В этом упражнении мы возьмём то, чему научились в предыдущем упражнении, и внесём небольшое изменение, чтобы получить новое движение. Тогда робот-исполнитель сможет двигаться по линии.



Необходимые детали и принадлежности

- Светлая поверхность с контрастными темными элементами
- Полностью собранный робот-исполнитель PRIZM с датчиком
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер

Чёрная линия не уже 5 сантиметров

Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act12_ Follow_A_Line**. Откроется окно нового скетча под названием TaskBot_Act12_Follow_A_Line (рисунок 25).

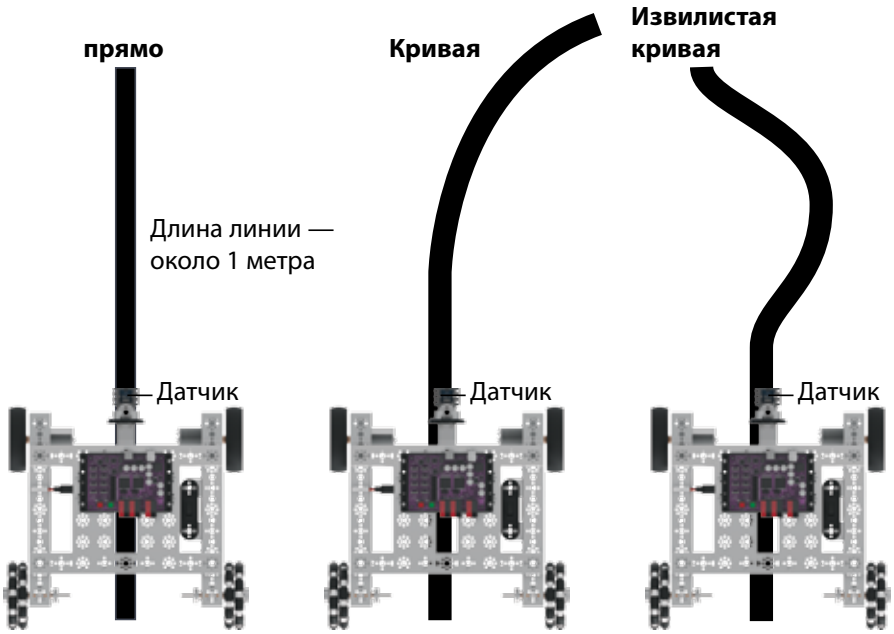
Рисунок 25

Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Датчик линии необходимо подсоединить к порту № 3 для цифровых датчиков.

Для выполнения этого программного кода понадобится белая или светоотражающая поверхность: на неё мы поместим чёрную полосу, которую предстоит пересечь роботу. Подойдёт белый, глянцевый картон или несколько листов белой бумаги, уложенных встык.

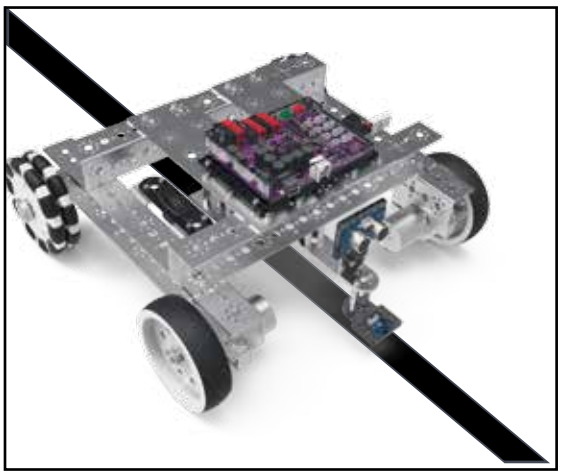
Чёрную полосу можно сделать из изоленты, наклеив её на картон или на соединённые встык листы бумаги. Траекторию движения можно сделать криволинейной или прямолинейной. У кривизны поворота нашего робота будет ограничение, поэтому не стремитесь поворачивать со слишком малым радиусом.



Поиск и устранение неисправностей: Убедитесь, что датчик линии вставлен в предусмотренный для него порт и правильно настроен на обнаружение линии. Возможно, понадобится слегка подправить высоту или чуть сдвинуть регулировочный винтик на тыльной стороне датчика. Чтобы проверить исправность датчика, вручную проведите модель вперёд-назад по белой поверхности и над чёрной линией. Пока датчик линии находится над белой поверхностью, на нём должен гореть красный светодиод, а когда датчик оказывается над чёрной линией, светодиод должен гаснуть.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Робот-исполнитель должен находиться на белой поверхности так, чтобы датчик линии был чуть сбоку от линии, по которой предстоит двигаться. Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за действиями робота.



Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Последующие действия

В скетче используется два условных оператора (оператора "if"), которые становятся нам всё привычнее. Мы добавили ещё одно действие, которое должно выполняться с учётом проверяемого условия для каждого условного оператора.

Первое действие, запускаемое условными операторами, связано с электродвигателями. Условный оператор приказывает роботу-исполнителю выполнить поворот, посылая на один из электродвигателей сигнал мощности, а на другой — сигнал торможения с учётом условия, за истинностью или ложностью которого следит датчик линии. Каждый из условных операторов противоположен другому. Это приводит к ряду последовательных поворотов, которые робот-исполнитель совершает, двигаясь по линии.

Второе действие, запускаемое условными операторами, связано с работой красного светодиода на контроллере PRIZM. Условный оператор приказывает светодиоду загореться или погаснуть с учётом условия, за истинностью или ложностью которого следит датчик линии. Это включение-выключение светодиода синхронизируется с вращением электродвигателей.

Действия, запускаемые каждым из условных операторов, согласованы, так что робот-исполнитель движется по линии и при этом выдаёт световые сигналы при помощи мигающего красного светодиода.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков
- Страницы 137-142: функции электродвигателя постоянного тока

Связь с настоящей жизнью

Немецкий концерн PUMA решил по-новому подойти к физическим тренировкам и выпустил на рынок темпоробот PUMA. Темпоробот движется по линии, нанесённой на беговую дорожку, задавая бегуну темп. Бегун может по смартфону сообщить роботу расстояние и темп бега. Потом оба выходят на дорожку, и там бегун пытается обогнать темпоробота.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Отражение и поглощение света
- Угол падения и отражения

Технология

- Пороговые значения
- Чёрный и белый цвета (аналоговые сигналы) в роли сигналов "вкл." и "выкл." (цифровых)

Проектирование

- Проектирование машин для работы на краю

Математика

- Применение логических операторов для решения задач

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя ехать по чёрной линии. Вспомните, чему мы научились, выполняя предыдущие упражнения.

Поставьте себе задачу заставить робота-исполнителя двигаться быстрее, но при этом точно по линии, или изменить программный код так, чтобы робот-исполнитель двигался по линии в обратном направлении.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

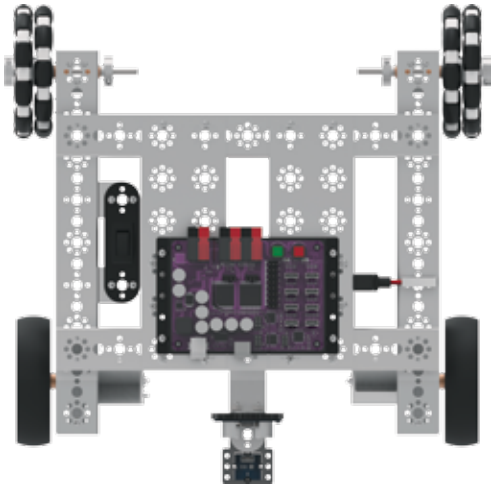
Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 13: Подъезд к стене и остановка

В этом упражнении мы продолжим развивать усвоенное применительно к ультразвуковому датчику. Он позволит роботу-исполнителю подъехать к стене и остановиться на заданном расстоянии от неё.



Необходимые детали и принадлежности

- Полностью собранный робот-исполнитель PRIZM с датчиком
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер

Открывание скетча

Прежде чем открыть наш следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act13_ Drive_To_Wall**. Откроется окно нового скетча под названием TaskBot_Act13_Drive_To_Wall (рисунок 26).

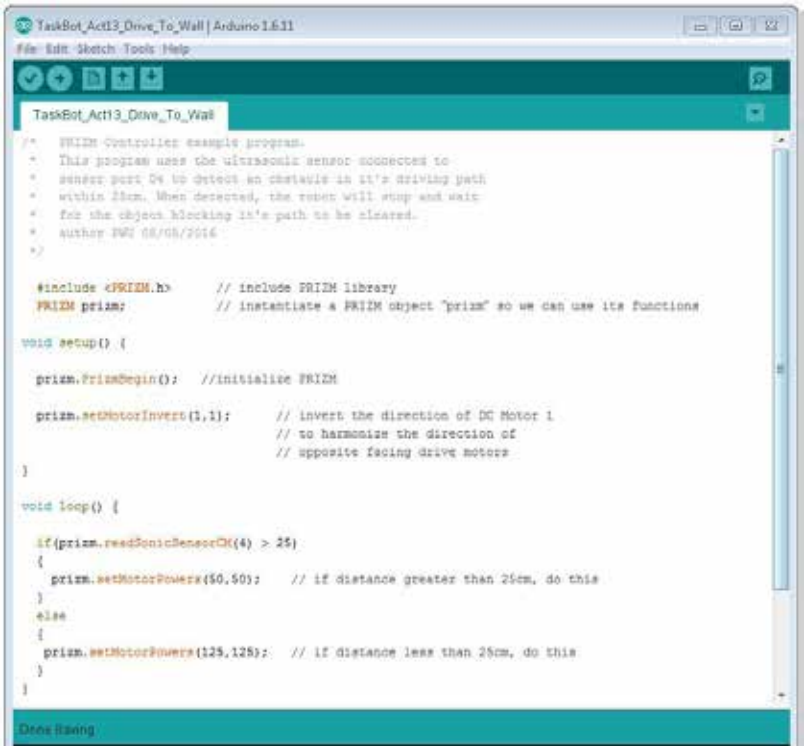


Рисунок 26

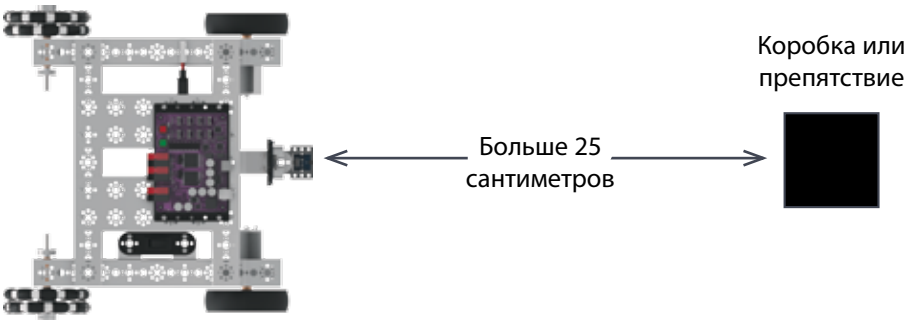
Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Ультразвуковой датчик необходимо подсоединить к порту № 4 для цифровых датчиков.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Робота-исполнителя следует развернуть к предмету или стене на расстоянии не менее 25 сантиметров. Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за действиями робота. Что происходит, если робота-исполнителя или предмет сдвинуть до завершения скетча?

Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?



Последующие действия

В скетче используется полная форма условного оператора "if/else" — новый элемент в структуре программы.

Для управления ходом выполнения программного кода полный условный оператор "if/else" лучше простейшей формы условного оператора "if". Он позволяет сгруппировать несколько проверок и провести их одновременно. То есть часть оператора со словом "if" утверждает, что если условие истинно, необходимо сделать одно. Часть оператора, содержащая слово "else", утверждает, что если условие ложно, необходимо сделать другое.

В этом скетче первая часть оператора "if/else" проверяет условие, за истинностью которого следит ультразвуковой датчик, и если датчик находится дальше 25 сантиметров от объекта, то электродвигателям задаётся мощность 50 %. Вторая часть оператора "if/else" использует ту же проверку, но если расстояние до объекта меньше 25 сантиметров, то даётся команда на исполнение другого действия, и электродвигатели получают сигнал затормозить.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков
- Страницы 137-142: функции электродвигателя постоянного тока

Поиск и устранение неисправностей: Убедитесь, что ультразвуковой датчик вставлен в предусмотренный для него порт и вставлен правильно. С этим скетчем следует использовать порт D4. Учтите, что датчик может не обнаружить предметы с недостаточной площадью поверхности или с неровной поверхностью, либо неверно определить расстояние до них. Для проверки исправности датчика можно всегда вернуться к упражнению № 5; загрузите и запустите относящийся к нему скетч, чтобы проверить прибор контроля последовательной передачи данных и выходной сигнал датчика. Если это понадобится, не забудьте либо подсоединить ультразвуковой датчик к другому порту для датчиков, который соответствовал бы скетчу, либо заменить пример скетча.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Связь с настоящей жизнью

Во многих более современных автомобилях есть датчики, подсказывающие водителю об опасном сближении с другим автомобилем или стеной при заезде на стоянку. При сближении автомобиля со стеной датчик включает звуковое предупреждение водителю, побуждая его снизить скорость и остановиться до столкновения со стеной.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Скорость звука
- Состав звуковых волн

Технология

- Калибровка (тарировка) датчиков

Проектирование

- Проектирование машин для обнаружения препятствий

Математика

- Расчёты расстояния на основе скорости звука

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя подъехать к стене или препятствию и остановиться. Вспомните, чему мы научились, выполняя предыдущие упражнения.

Поставьте себе задачу изменить направление, дальность или скорость хода. Можно также проверить, объект какого размера и формы ультразвуковой датчик обнаружит скорее и на каком расстоянии.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

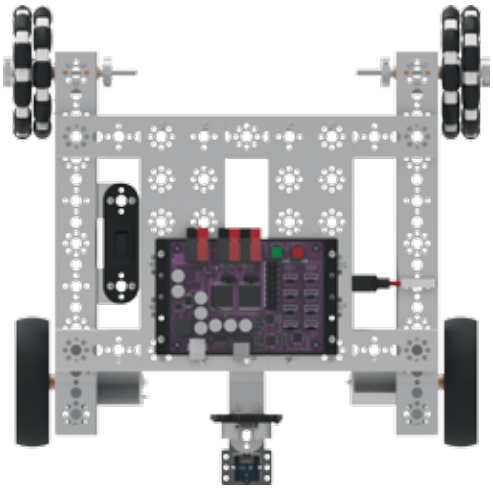
Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Упражнение 14: Объезд препятствий

В этом упражнении оператор "if/else" будет управлять дополнительными действиями. Благодаря этому робот-исполнитель сможет объезжать возникающие на его пути препятствия.

Необходимые детали и принадлежности

- Несколько препятствий, которые предстоит объехать
- Полностью собранный робот-исполнитель PRIZM с датчиком
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act14_Avoid_Obstacle**. Откроется окно нового скетча под названием TaskBot_Act14_Avoid_Obstacle (рисунок 27).

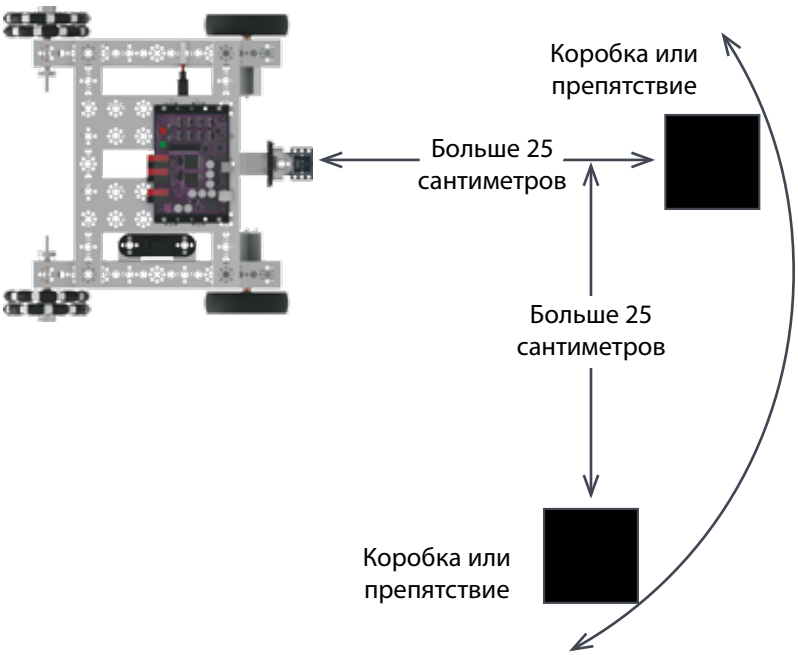
Рисунок 27

Запуск программы

Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Ультразвуковой датчик необходимо подсоединить к порту № 4 для цифровых датчиков.

Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Для выполнения этого программного кода установите перед нашим роботом какое-нибудь препятствие на расстоянии намного превышающем 25-сантиметровую дальность обнаружения. Для этой цели хорошо подойдут картонные коробки. Избегайте слишком тяжёлых предметов, чтобы робот не получил повреждений, если вдруг столкнётся с ними.



Чтобы начать упражнение, нажмите зелёную пусковую кнопку. Наш робот будет двигаться передним ходом, пока не обнаружит в 25 сантиметрах от себя препятствие прямо по курсу.

При обнаружении препятствия робот должен остановиться, отъехать назад, повернуть вправо и продолжить движение. Понаблюдайте за действиями робота.

Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Последующие действия

В скетче по-прежнему используется оператор "if/else" из предыдущего упражнения, но теперь он отвечает за несколько новых действий.

Каждая часть оператора "if/else" отвечает здесь за несколько действий. В части оператора со словом "if" робот-исполнитель движется передним ходом с 35%-й мощностью электродвигателей и горит зелёный светодиод, пока датчик выискивает препятствие. В части оператора со словом "else" робот-исполнитель обнаруживает препятствие, зелёный светодиод гаснет, а робот-исполнитель останавливается, отъезжает назад и поворачивает вправо. После этого цикл повторяется.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков
- Страницы 137-142: функции электродвигателя постоянного тока

Поиск и устранение неисправностей: Убедитесь, что ультразвуковой датчик вставлен в предусмотренный для него порт и вставлен правильно. С этим скетчем следует использовать порт D4. Учтите, что датчик может не обнаружить предметы с недостаточной площадью поверхности или с неровной поверхностью, либо неверно определить расстояние до них. Для проверки исправности датчика можно всегда вернуться к упражнению № 5; загрузите и запустите относящийся к нему скетч, чтобы проверить прибор контроля последовательной передачи данных и выходной сигнал датчика. Если это понадобится, не забудьте либо подсоединить ультразвуковой датчик к другому порту для датчиков, который соответствовал бы скетчу, либо заменить пример скетча.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Связь с настоящей жизнью

Автомобили начинают снабжать датчиками, помогающими ориентироваться в окружающем пространстве во время движения. У некоторых автомобилей теперь есть системы автоматического торможения, которые начинают торможение и замедляют автомобиль, если обнаруживают ситуацию, чреватую столкновением. Участвующая в этом электроника срабатывает быстрее человека. Выигрыш во времени очень способствует уменьшению возможных последствий столкновения или наезда.

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Физика отражения звука от плоской поверхности
- Физика отражения звука от округлой поверхности
- Физика отражения звука от неровной поверхности

Технология

- Толкование показаний ультразвукового датчика

Проектирование

- Проектирование машин для объезда препятствий

Математика

- Графическое представление данных
- Усреднение данных

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя объезжать препятствия. Вспомните, чему мы научились, выполняя предыдущие упражнения.

Поставьте себе задачу опробовать модель в работе с разными объектами или создать испытательную трассу, на которой необходимо будет обнаружить множество препятствий. Мы можем попробовать внести нужные изменения в образец кода, чтобы получить другую реакцию на препятствие и научить робота действовать иначе.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

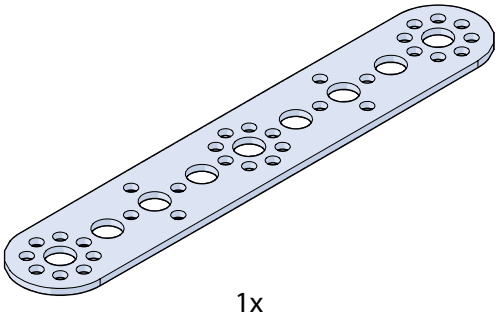
Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Вводное задание по конструированию: Наделяем робота-исполнителя способностью выражать отношение!

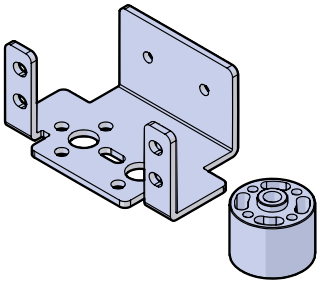
Перед составлением новых кодов мы добавим роботу-исполнителю PRIZM ещё один сервопривод. Это позволит объединить всё пройденное в начальных упражнениях. При помощи всевозможных электродвигателей и датчиков мы немного научим робота-исполнителя выражать отношение к происходящему.

С помощью сервопривода будет подаваться сигнал об обнаружении объекта на пути робота. Сервопривод должен будет поднять "руку" в виде рычага, показывая , что обнаружил некий объект. Для оснащения робота-исполнителя PRIZM сервоприводом и рукой в виде рычага выполните указанные ниже действия.

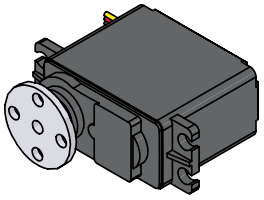
Необходимые детали и принадлежности



1x
Планка 160 мм x 27 мм
39272



1x
Одинарный кронштейн для стандартных сервоприводов 39060



1x
Стандартный сервопривод HS-485HB с поворотом вала на 180° 39197

6x
Винт со сферической головкой 3/8" 39111

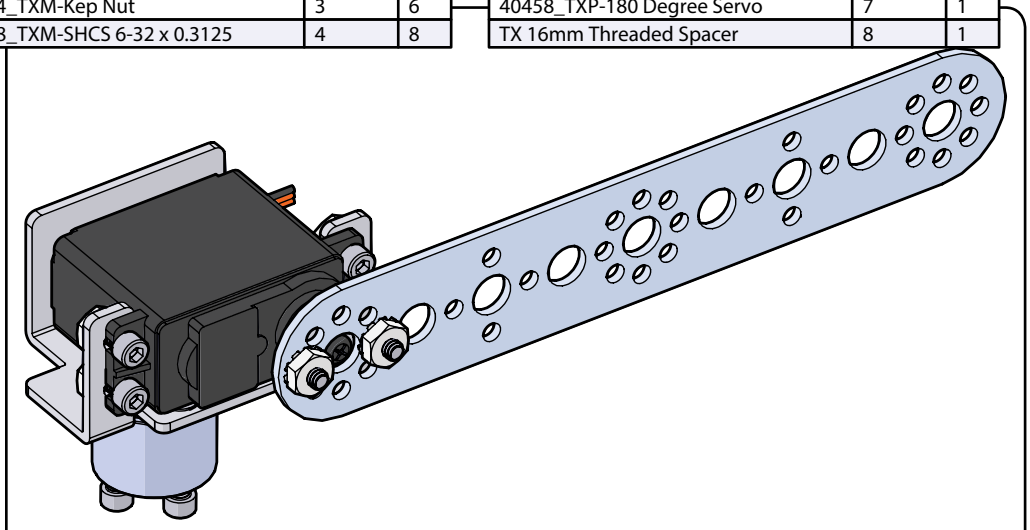
8x
Винт с головкой под торцевой ключ 6-32 x 5/16" 39098

6x
Зубчатая гайка 39094

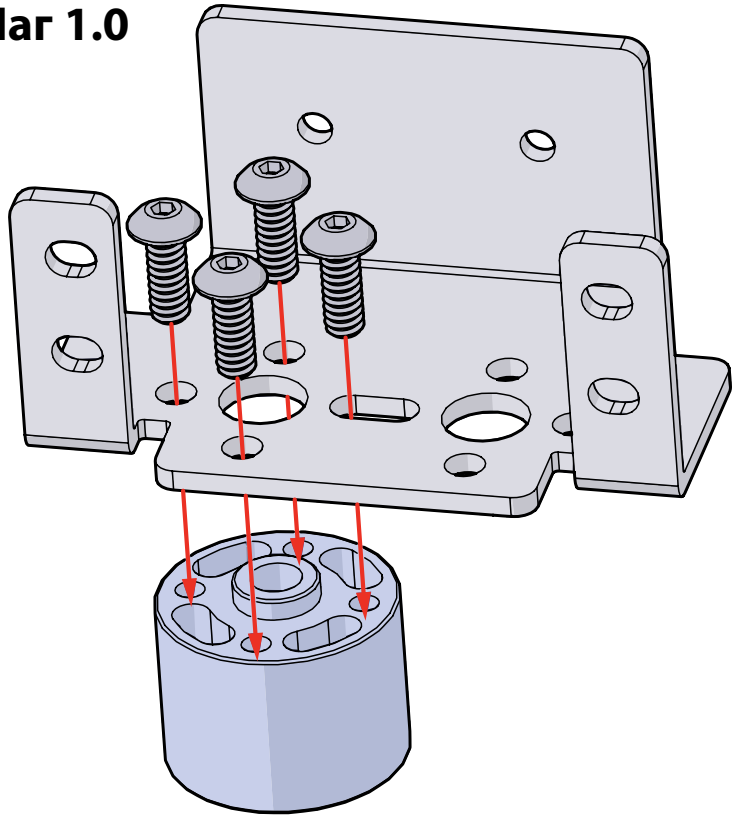
Description	BOM ID	Qty
39020_TXM-Servo Mount Horn	1	1
39060_TXM-Standard Servo Bracket	2	1
39094_TXM-Kep Nut	3	6
39098_TXM-SHCS 6-32 x 0.3125	4	8

Description	BOM ID	Qty
39111_TXM-BHCS 6-32 x 0.375	5	6
39272_TXM-Flat160mmx26mm	6	1
40458_TXP-180 Degree Servo	7	1
TX 16mm Threaded Spacer	8	1

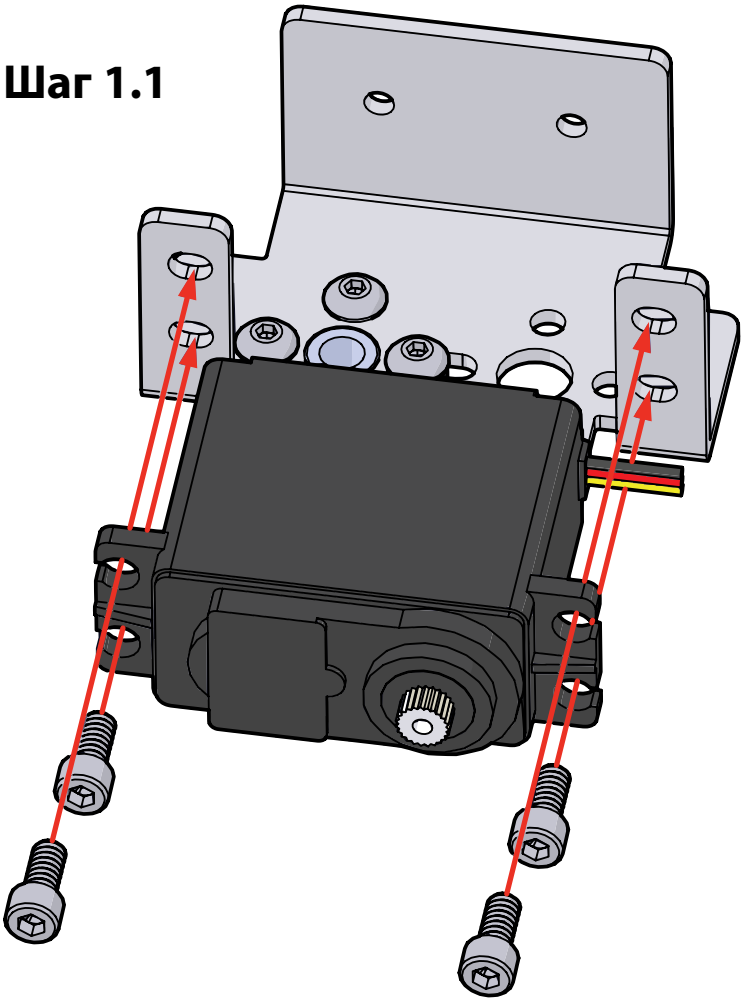
Description	BOM ID	Qty
TXP Servo Horn Attachment Screw	9	1



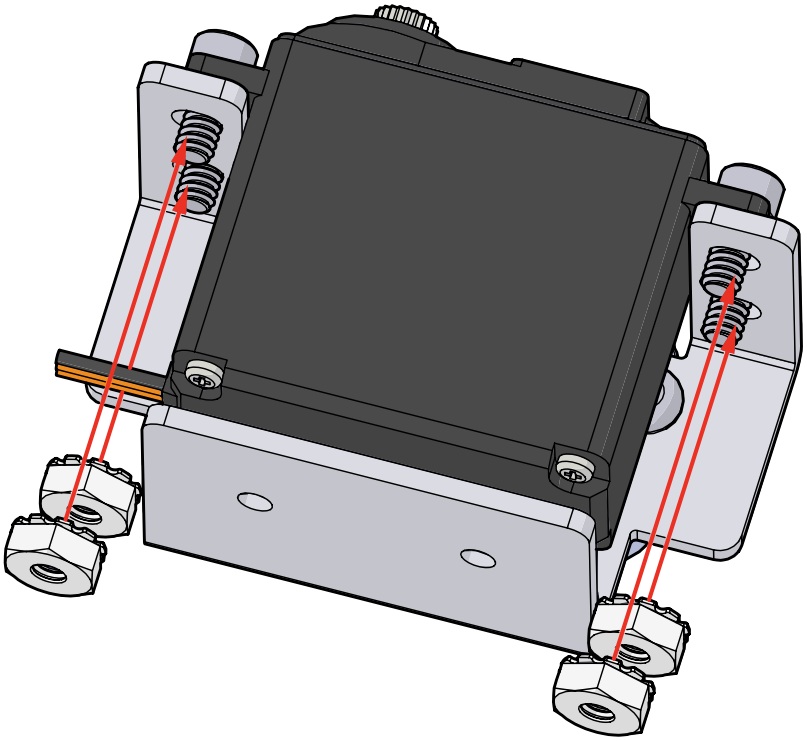
Шаг 1.0



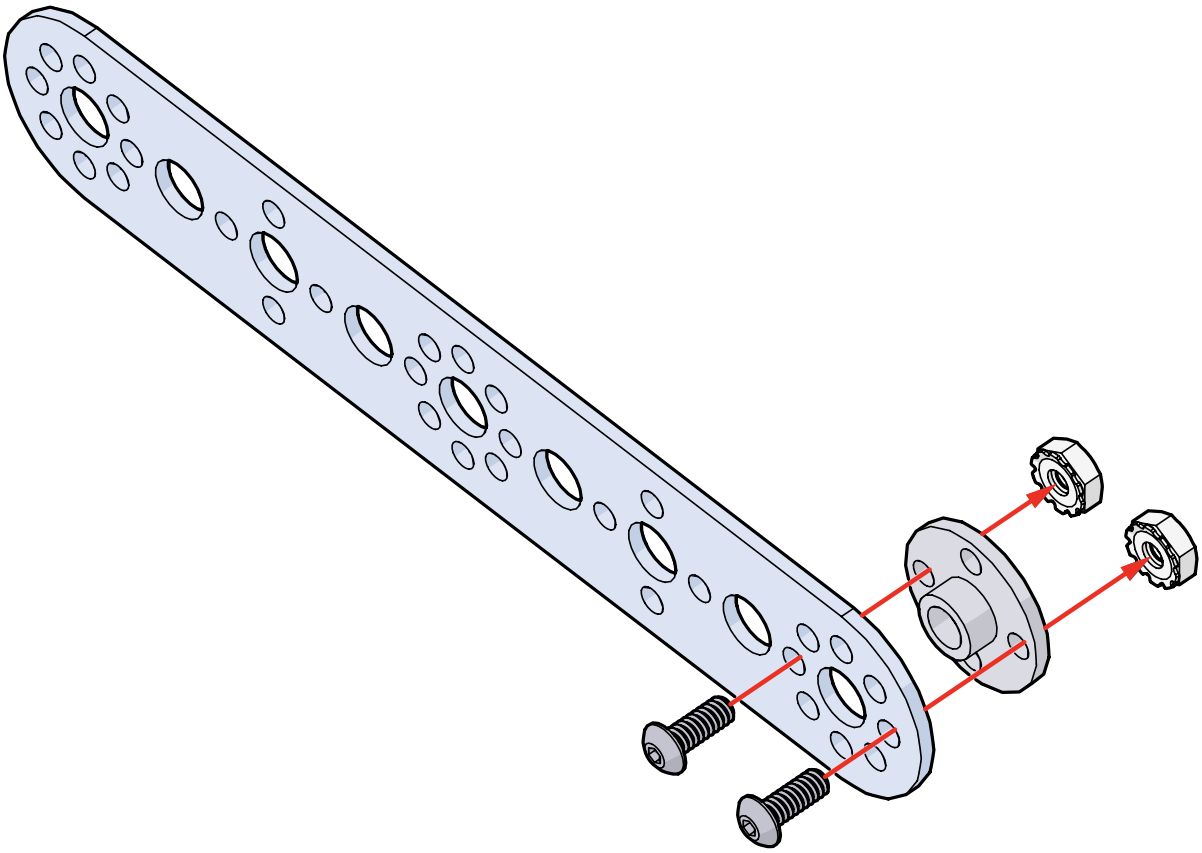
Шаг 1.1



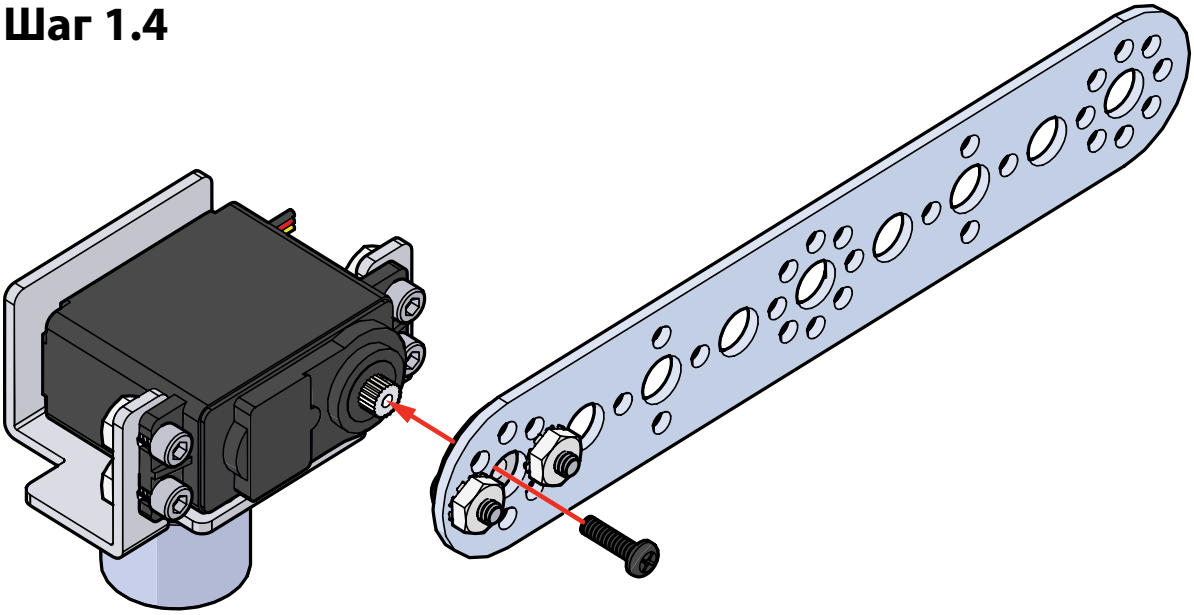
Шаг 1.2



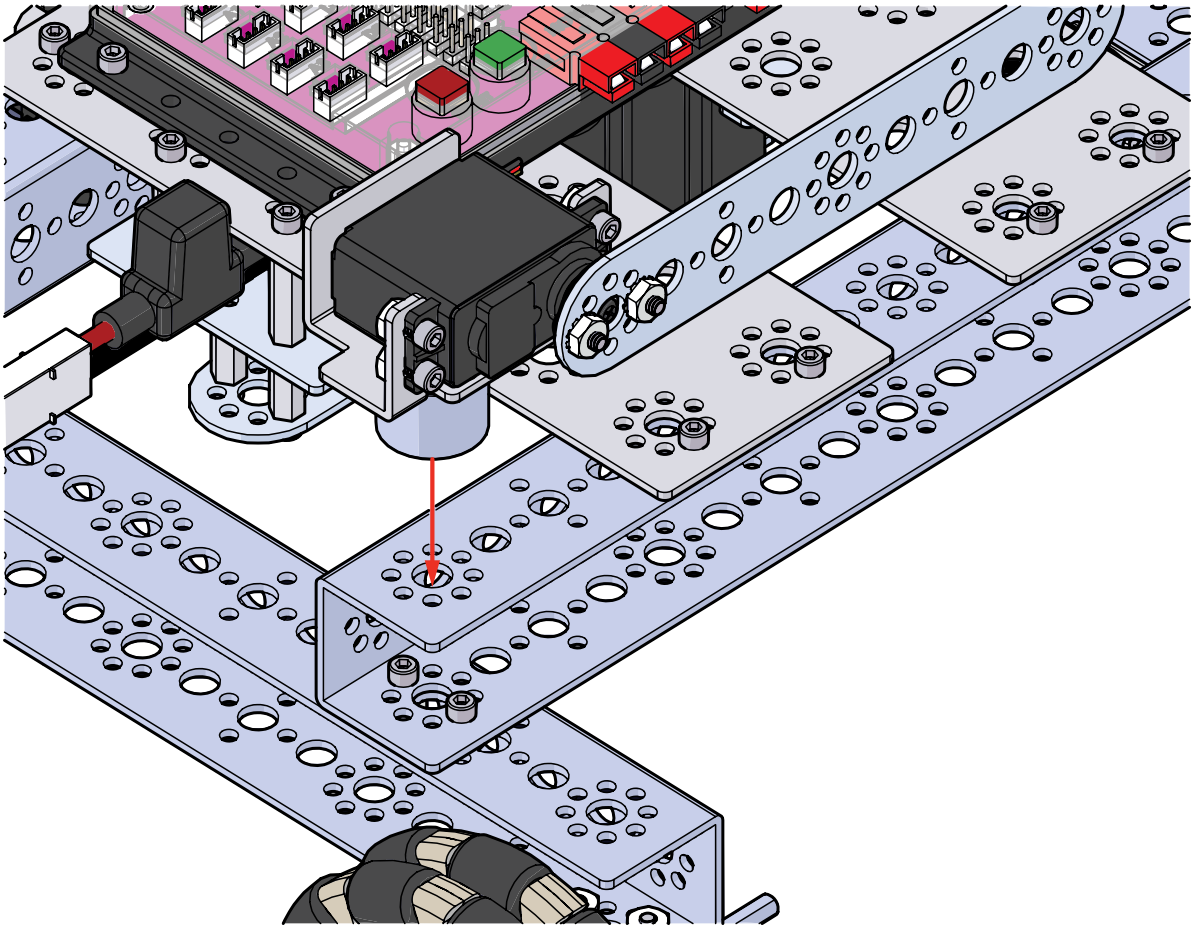
Шаг 1.3



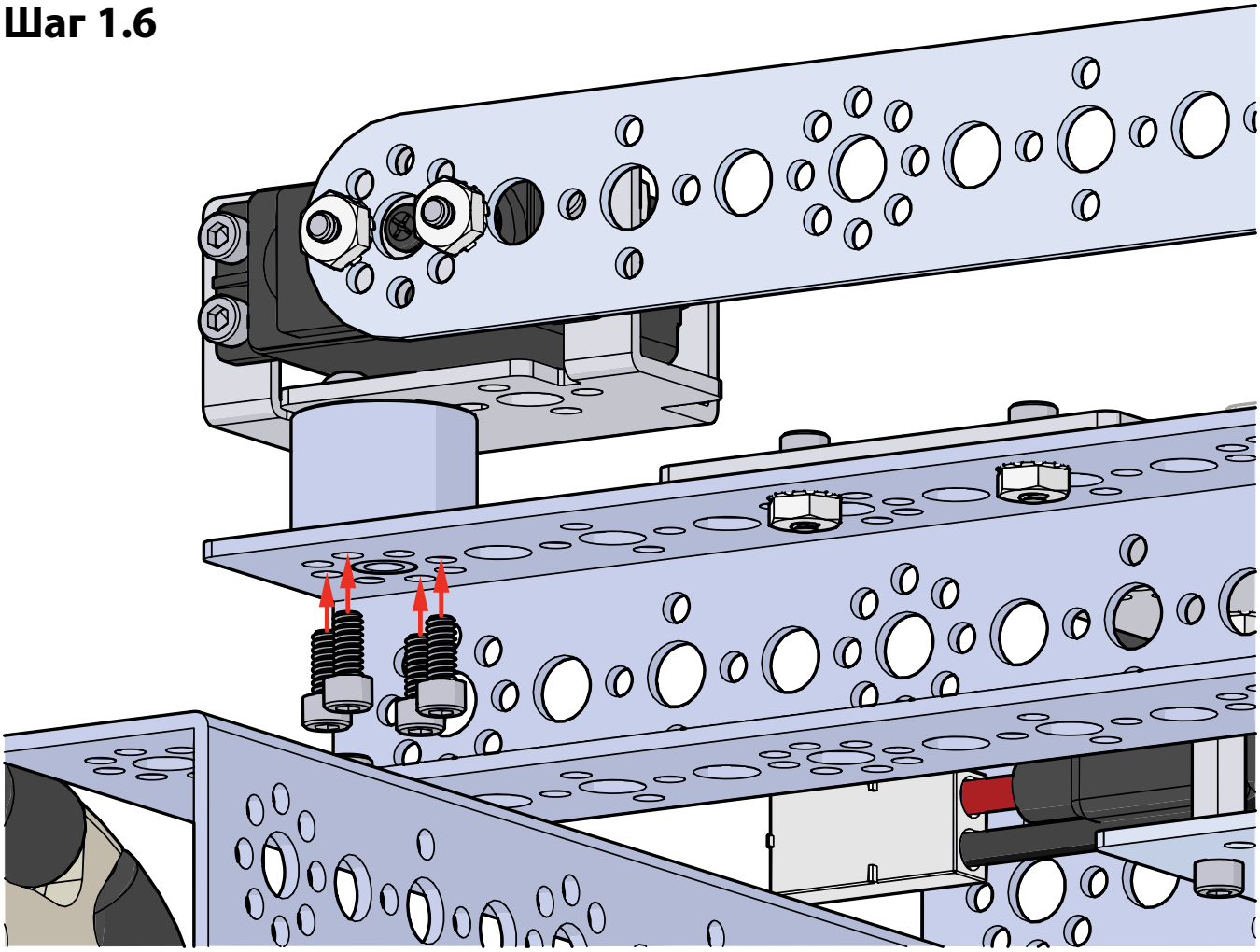
Шаг 1.4



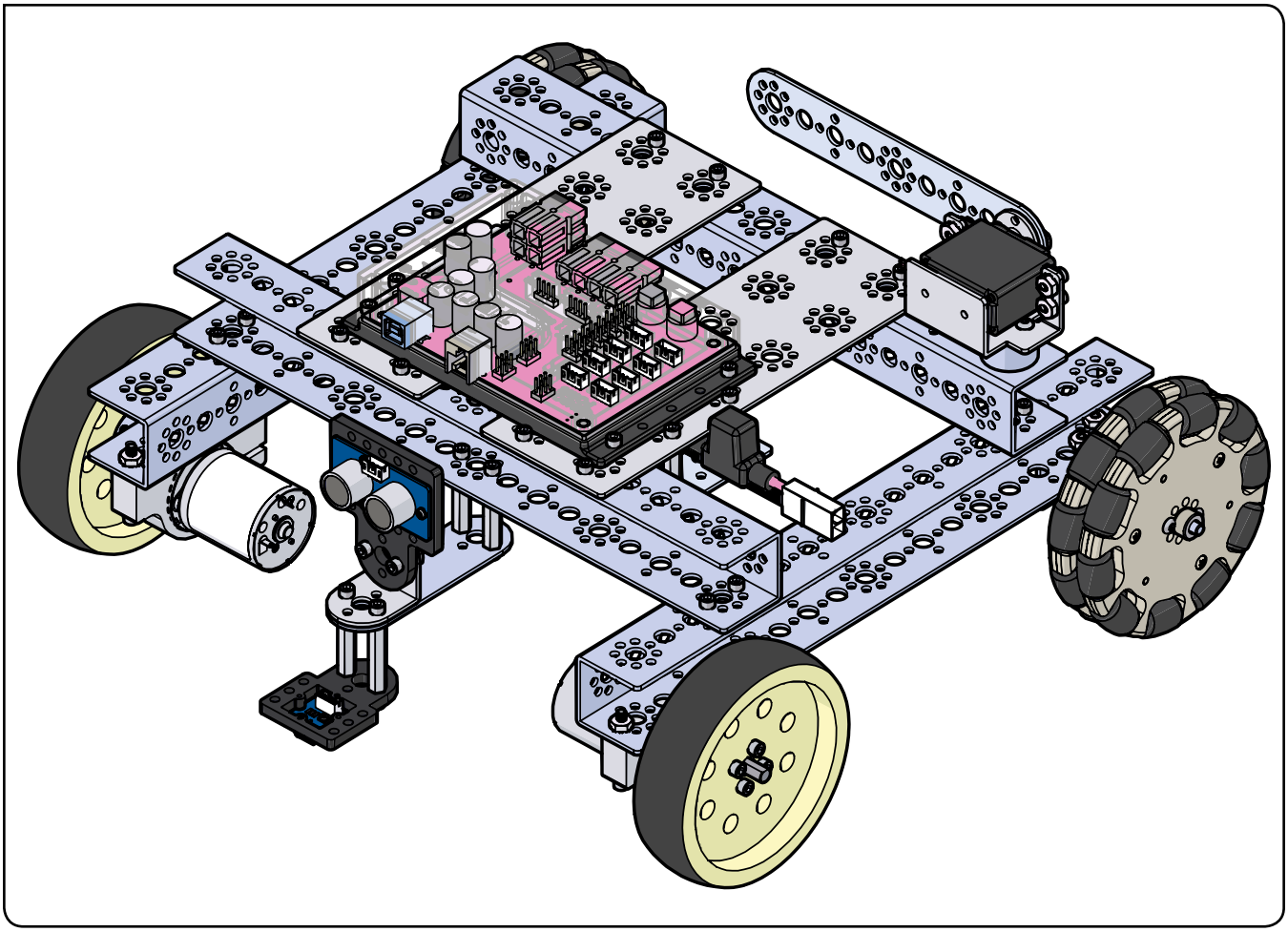
Шаг 1.5



Шаг 1.6



Собранная модель должна выглядеть так.

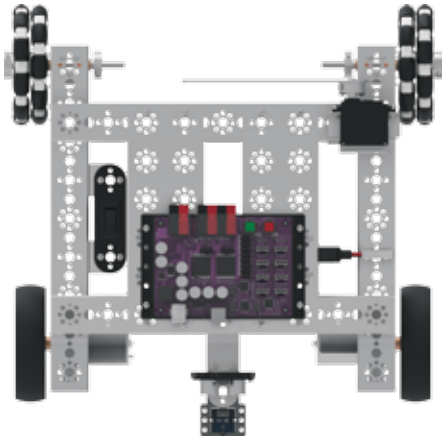


Упражнение 15: Использование нескольких датчиков

Это этап, объединяющий и завершающий все предыдущие упражнения. Робот-исполнитель будет двигаться по линии и выискивать препятствия. Обнаружив некое препятствие, робот-исполнитель остановится, поднимет руку и будет ждать, пока препятствие не уберут, после чего продолжит движение.

Необходимые детали и принадлежности

- Светлая поверхность с контрастными темными элементами
- Препятствие
- Полностью собранный робот-исполнитель PRIZM с датчиком
- Кабель USB
- Заряженная аккумуляторная батарея 12 В NiMH из конструктора TETRIX
- Компьютер



Открывание скетча

Прежде чем открыть следующий пример скетча, обязательно сохраните любые скетчи, которые захотите позднее использовать в качестве образца.

Откройте скетч, последовательно выбрав в меню пункты **File > Examples > TETRIX_PRIZM > TaskBot_Act15_ Combining_Sensors**. Откроется окно нового скетча под названием TaskBot_Act15_Combining_Sensors (рисунок 28).

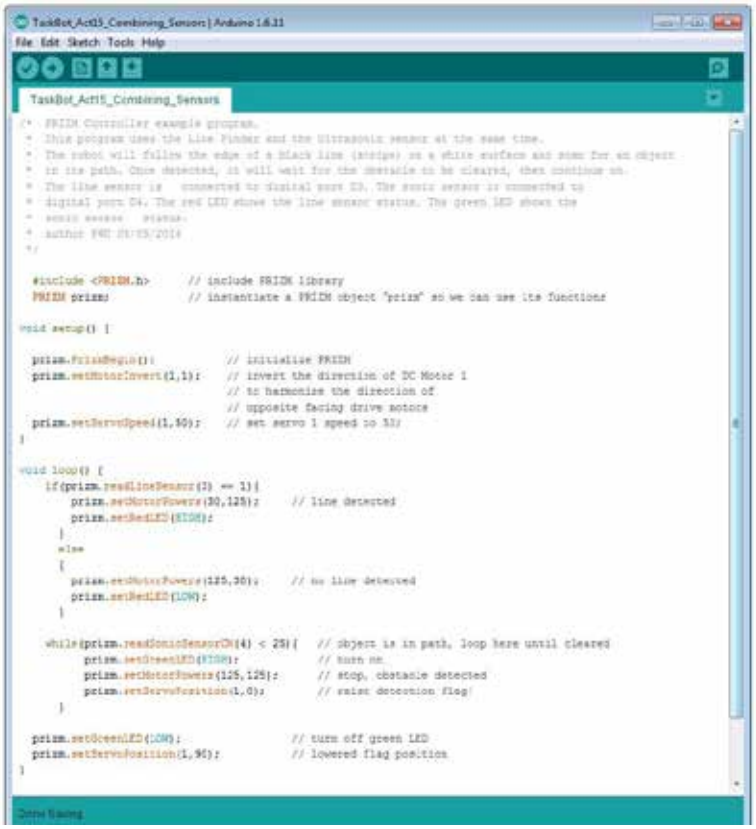
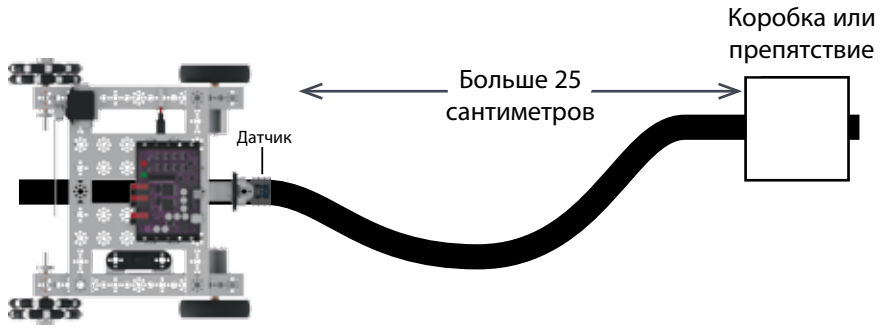


Рисунок 28

Запуск программы

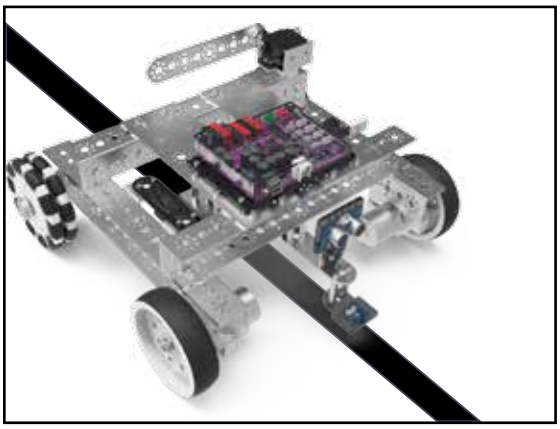
Перед тем как загрузить скетч в контроллер PRIZM, не забудьте проверить соединения. Датчик линии следует подсоединить к порту № 3 для цифровых датчиков, а ультразвуковой датчик — к порту № 4 для цифровых датчиков. Мы используем датчик линии для движения по линии, а ультразвуковой датчик для обнаружения препятствий. В этом примере скетча робот будет двигаться по чёрной линии на белой поверхности и следить за появлением объекта на своём пути.

При обнаружении объекта на расстоянии менее 25 сантиметров робот должен остановиться, чтобы не врезаться в препятствие. Робот поднимет свою руку и будет ждать, пока препятствие не уберут, после чего продолжит движение.



Подгрузите скетч. Должен загореться зелёный светодиод, означающий готовность кода к выполнению. Когда это случится, отсоедините кабель USB и поставьте робота-исполнителя на пол.

Робот-исполнитель должен находиться на белой поверхности так, чтобы датчик линии был чуть сбоку от линии, по которой предстоит двигаться.



Для выполнения этого программного кода установите перед нашим роботом какое-нибудь препятствие на расстоянии намного превышающем 25-сантиметровую дальность обнаружения. Для этой цели хорошо подойдут картонные коробки. Избегайте слишком тяжёлых предметов, чтобы робот не получил повреждений, если вдруг столкнётся с ними.

Нажмите зелёную пусковую кнопку, чтобы запустить выполнение скетча. Понаблюдайте за действиями робота. Можно также переставлять препятствия в разные места и наблюдать, как поведёт себя робот.

Нажмите красную кнопку сброса параметров, чтобы прекратить выполнение скетча. Исходя из заметок к скетчу, совпало ли движение с вашими ожиданиями?

Поиск и устранение неисправностей в датчике линии: Убедитесь, что датчик линии вставлен в предусмотренный для него порт и правильно настроен на обнаружение линии. Возможно, понадобится слегка подправить высоту или чуть сдвинуть регулировочный винтик на тыльной стороне датчика. Чтобы проверить исправность датчика, вручную проведите модель вперёд-назад по белой поверхности и над чёрной линией. Пока датчик линии находится над белой поверхностью, на нём должен гореть красный светодиод, а когда датчик оказывается над чёрной линией, светодиод должен гаснуть.

Поиск и устранение неисправностей в ультразвуковом датчике: Убедитесь, что ультразвуковой датчик вставлен в предусмотренный для него порт и вставлен правильно. С этим скетчем следует использовать порт D4. Учтите, что датчик может не обнаружить предметы с недостаточной площадью поверхности или с неровной поверхностью, либо неверно определить расстояние до них. Для проверки исправности датчика можно всегда вернуться к упражнению № 5; загрузите и запустите относящийся к нему скетч, чтобы проверить прибор контроля последовательной передачи данных и выходной сигнал датчика. Если это понадобится, не забудьте либо подсоединить ультразвуковой датчик к другому порту для датчиков, который соответствовал бы скетчу, либо заменить пример скетча.

Подсказка: Если контроллер PRIZM не выполняет подгруженный скетч, попробуйте несколько раз выключить-включить электропитание робота.

Последующие действия

В скетче использованы полный условный оператор "if/else" и цикл "while", которые мы уже проходили. Скетч предназначен не для добавления новых элементов. Он нужен для объединения элементов, взятых из предыдущих упражнений.

Оператор "if/else" в этом скетче отвечает за движение по линии, как и в упражнении № 6 с двумя операторами "if", но степень управления движением здесь выше. Цикл "while" следит за сигналами ультразвукового датчика и выдаёт команду на действие при обнаружении объекта в пределах 25 сантиметров. Эта новая команда останавливает робота-исполнителя и при помощи сервопривода поднимает его руку, не опуская её, пока объект не уберут с пути робота-исполнителя. После того как объект убирают, робот-исполнитель опускает свою руку и продолжает изначально заданное движение по линии.

Подробнее о создании скетчей и библиотечных функциях контроллера PRIZM, задействуемых в работе электродвигателей постоянного тока, см. **www.arduino.cc** и приложение: Библиотечные функции на языке Arduino для контроллера TETRIX PRIZM:

- Страницы 134-135: функции порта для датчиков
- Страницы 137-142: функции электродвигателя постоянного тока
- Страницы 144-145: функции сервопривода

Связь с настоящей жизнью

Легкодоступные в наши дни электронные устройства (смартфоны, приборы наблюдения за состоянием здоровья и т. п.) считывают показания датчиков, на основе которых мы решаем, куда пойти, что есть, какую физическую нагрузку себе дать, когда встать утром и многое другое. За объединение показаний всех этих датчиков и принятие соответствующих решений отвечают программные коды этих устройств. Программирование робототехнических моделей применяется схожим образом — всё дело в том, каких действий вы хотите от своего робота с учётом имеющихся у него датчиков!

Дальнейшее изучение в контексте научно-технических дисциплин

Физика

- Органы чувств у человека
- Работа человеческого мозга на основе данных, получаемых от органов чувств

Технология

- Определение степени предпочтительности показаний датчиков

Проектирование

- Объединение показаний датчиков для решения задач

Математика

- Термины, относящиеся к распределению данных (среднее, медиана, мода, размах, кластер, выброс, асимметрия)

Упражнение по программированию

Опираясь на этот пример, попытайтесь создать новый скетч, который заставит робота-исполнителя ехать по чёрной линии и останавливаться перед препятствиями. Вспомните, чему мы научились, выполняя предыдущие упражнения.

Поставьте себе задачу создать скетч, который заставит робота-исполнителя подъехать по линии к некоему объекту, и продолжить движение, не дожидаясь устранения этого объекта с пути.

Подсказка: Не забывайте, что для удобства можно распечатать памятку с библиотечными функциями на языке Arduino для контроллера TETRIX PRIZM, которая есть в приложении на странице 153, если понадобится быстро узнать, какие функции имеются в вашем распоряжении.

Подсказка: Хотите увидеть, как это работает? Тогда посмотрите серию наших видеоочерков про рабочее место робототехника, дополняющую *Руководство по программированию контроллера PRIZM*. Всю серию видеоочерков см. на сайте **video.tetrixrobotics.com** или на канале TETRIXrobotics в YouTube.

Подсказка: В окне скетча библиотечные функции контроллера PRIZM меняют цвет, если напечатаны или введены без ошибок. Соответственно, если в их написании есть ошибка, то цвет не поменяется. Программное обеспечение Arduino распознаёт функции контроллера PRIZM как ключевые слова и окрашивает их в оранжевый цвет, если не нарушены синтаксические правила.

Подсказка: Пример библиотеки кодов в помощь начинающим. Это упражнение можно найти в приложении. Если у вас есть цифровая копия этого руководства, можно просто скопировать и вставить в окно вашего скетча образец кода к каждому упражнению. Цифровые материалы для скачивания можно найти по адресу: **www.TETRIXrobotics.com/PRIZMdownloads**.

Собрали, запрограммировали, испытали, научились . . . Поехали!

Вы построили робототехническую модель, запрограммировали контроллер PRIZM, проверили и отладили свою конструкцию. Куда двигаться дальше?

Назначение этого руководства — помощь в усвоении необходимых основ конструирования робототехнической модели серии TETRIX MAX и её программирования при помощи контроллера PRIZM и *ПО Arduino (IDE)*. Но это лишь начало. Есть ещё немало дополнительных ресурсов и возможностей для конструирования более крупных робототехнических моделей, способных выполнять более сложные задания. Теперь, усвоив основы, вы ограничены только пределами своего воображения. Мы составили перечень наиболее предпочтительных ресурсов в помощь тем, кто готов взяться за дело.

Непременно загляните в приложение к этому руководству. Оно содержит обширный массив технической документации, в том числе технические характеристики контроллера PRIZM, обзоры составных частей и принципиальные схемы, описания, структурные схемы и таблицу с подсказками к библиотеке функций на языке Arduino для контроллера PRIZM. В приложении вы также найдёте объяснение того, как подключить дополнительные устройства управления, в частности энкодеры для электродвигателей (то есть датчики углового положения валов электродвигателей).

Подробнее о *ПО Arduino (IDE),* особенностях и секретах программирования:

Контроллер PRIZM построен на схеме Arduino, поэтому может работать с языками программирования, совместимыми с языком Arduino. Благодаря этому пользователи контроллеров PRIZM получают доступ в интернет-сообщество Arduino, которое создало массу вспомогательных материалов, помогающих новичкам быстро освоить программирование на этом языке. См. подробности на **www.arduino.cc**.

Дополнительно о подключении и поддержке датчиков:

Контроллер PRIZM снабжён портами для датчиков, совместимыми с модульными датчиками системы Grove. Сейчас в библиотеке Arduino для контроллера PRIZM предусмотрена поддержка двух таких датчиков: датчика линии и ультразвукового. Пользователи, которые хотели бы расширить круг датчиков, совместимых с их робототехническими моделями, найдут пример кода на языке Arduino к каждому из датчиков серии Grove здесь **wiki.seeedstudio.com/wiki/Category:Grove** or **wiki.seeedstudio.com/wiki/Grove_System**.

Чтобы увидеть пример программного кода Arduino, просто щёлкните по интересующему вас датчику.

Обратите внимание! Съемные переходники для других датчиков серии Grove, совместимых с конструкторами TETRIX, продаются отдельно и понадобятся для прикрепления датчиков к конструкции робототехнической модели. В последующие версии библиотеки Arduino для контроллера PRIZM будут включены средства поддержки дополнительных датчиков серии Grove.

Подробнее о конструкторе TETRIX:

Детали, описанные в этом руководстве, лишь часть конструктора серии TETRIX MAX. На сайте **www.TETRIXrobotics.com** вы подробнее узнаете о других конструктивных деталях, элементах механизмов движения и вспомогательных комплектующих для пополнения вашей коллекции; там же можно посмотреть вспомогательные видеоочерки и скачать ресурсы.

Технические характеристики контроллера к робототехническим моделям серии TETRIX® PRIZM™

Микроконтроллер:	ATmega328P с установленным загрузчиком операционной системы Arduino Optiboot
Память:	программируемое ЭПЗУ объёмом 32 кб (ATmega328P)
Питающее напряжение:	9...18 В пост. тока
Порты для управления электродвигателями постоянного тока:	2 соединителя с разъёмами Powerpole; мостовая схема управления при помощи широтно-импульсной модуляции; каждый канал рассчитан на постоянный ток силой 10 А и максимальную нагрузку по току до 20 А
Рекомендованный электродвигатель:	Электродвигатель постоянного тока на 12 В TETRIX® (39530)
Режимы управления электродвигателями постоянного тока:	Постоянная мощность (от -100 % до 100 %) Постоянная скорость при помощи ПИД-регулирования (от -720 до 720 градусов в секунду) ПИД-регулирование постоянной скорости для достижения и удержания заданного конечного положения с учётом показаний энкодера ПИД-регулирование постоянной скорости для достижения и удержания заданного конечного положения с учётом показаний энкодера в градусах Остановка в режиме торможения или выбега Отслеживание тока электродвигателя постоянного тока (все режимы)
Рекомендованный энкодер:	Комплект энкодера для электродвигателя серии TETRIX MAX (38000)
Порты под энкодеры для электродвигателей:	2 квадратурных, 5 В пост. тока, не более 50 мА; Технические характеристики: 360 отсчётов на оборот, 1440 импульсов на оборот; ENC 1 и ENC 2
Разъём USB:	USB типа B
Накопитель USB:	FTDI
Порты для стандартных сервоприводов:	Всего 6: каналы 1-6 для сервоприводов
Порты для сервоприводов продолжительного вращения (CR):	Всего 2: Каналы CR1 и CR2
Совокупное ограничение по току сервопривода:	постоянный ток не более 6 В, 6 А
Режимы управления сервоприводами:	Задание скорости сервопривода (0...100 %) Задание положение вала сервопривода (0...180 градусов) Задание состояния сервопривода продолжительного вращения (вращение по ЧС/против ЧС)
Отслеживание напряжения аккумуляторной батареи:	в пределах 0...18 В
Порты для цифровых датчиков (D2-D5):	Каждый можно настроить на ввод или вывод цифровых сигналов. D2 можно превратить в порт последовательной связи.
3 порта для аналоговых датчиков (A1-A3):	Каждый порт можно настроить на ввод аналоговых сигналов или ввод-вывод цифровых сигналов.
1 порт I2C (I2C):	скорость 100 кГц. Используется шина связи I2C, которая используется также для внутренних микроконтроллеров, управляющих электродвигателями постоянного тока и сервоприводами. Адреса I2C с 0x01 по 0x06 оставлены контроллером PRIZM в запасе.
Порт расширения под дополнительный контроллер электродвигателя (EXP):	К этому порту при помощи гирляндной схемы можно подключить дополнительные модули управления электродвигателями постоянного тока и сервоприводами TETRIX.
Порт для подсоединения аккумуляторной батареи:	подходит к разъёмам Powerpole; дополнительный порт для раздачи напряжения аккумуляторной батареи по гирляндной цепи контроллерам электродвигателей, подсоединяемых к порту расширения.
1 зелёная пусковая кнопка (START)	Программируемая кнопка
1 красная кнопка сброса параметров/остановки (RESET)	Непрограммируемая кнопка
1 красный светодиод:	Программируемый светодиод, служащий индикатором
1 зелёный светодиод:	Программируемый светодиод, служащий индикатором
1 синий светодиод:	Включается на время подачи в цепь электропитания
2 жёлтых светодиода:	Указывают на последовательную передачу данных через порт USB
1 красный и 1 зелёный светодиод электродвигателей постоянного тока	Обозначают наличие вращения электродвигателя постоянного тока и направление вращения, задаваемое по каждому каналу управления электродвигателями постоянного тока

Обзор функций контроллера к робототехническим моделям серии PRIZM

Контроллер PRIZM подсоединяют к компьютеру при помощи типового кабеля с разъёмом USB. Напряжение подается из внешней аккумуляторной батареи TETRIX, рассчитанной на 12 В, перезаряжаемой, никель-металлогидридной. В контроллере есть система привода сдвоенных сильноточных электродвигателей постоянного тока, каждый электродвигатель снабжён средствами поддержки квадратурного энкодера, благодаря чему осуществляется точное ПИД-регулирование скорости и положения валов электродвигателей постоянного тока. Кроме того, есть шесть портов для стандартных сервоприводов и два порта для сервоприводов продолжительного вращения. Для внешних датчиков есть четыре цифровых порта, два аналоговых и один порт I2C. Цифровой порт D2 тоже можно настроить на последовательную передачу данных при помощи соответствующей библиотеки в составе ПО Arduino. Все порты для цифровых датчиков можно настроить на ввод или вывод сигналов. Порты для аналоговых датчиков можно перевести в режим ввода аналоговых и вывода цифровых сигналов. Есть также два встроенных светодиода — по одному красного и зелёного цвета, — которые служат для подачи зрительных сигналов.

Функциями контроллера PRIZM управляет микроконтроллер ATmega328P; для подгрузки в него программного кода через разъём USB используется начальный загрузчик Optiboot. Есть ещё два процессорных микроконтроллера, которые управляют функциями электродвигателей: микроконтроллер, управляющий электродвигателями постоянного тока и обеспечивающий сопряжение с энкодерами, и микроконтроллер, управляющий сервоприводами. Эти микроконтроллеры обмениваются сигналами с главным процессорным микроконтроллером через шину связи I2C. В каждый микроконтроллер, управляющий электродвигателями, встроено ПО, которому подчиняются все сложные функции управления электродвигателями постоянного тока, энкодерами и сервоприводами, так что главный процессор этим не обременён и может свободно управлять выполнением программного кода. Зелёная пусковая кнопка (Start) служит для запуска программы, подгруженной в контроллер. Красная кнопка сброса параметров (Reset) служит для остановки программы. В результате нажатия на красную кнопку сброса параметров главный процессорный микроконтроллер, микроконтроллеры управления электродвигателями постоянного тока и сервоприводами возвращаются в исходное состояние, а в памяти обнуляются все сохранённые значения.

Обзор составных частей контроллера PRIZM™ и схемы расположения контактов

Порты для датчиков на контроллере PRIZM

В контроллере PRIZM используются схемы назначения контактов, совместимые с Arduino UNO. Датчики, к которым в библиотеке на языке Arduino для контроллера PRIZM есть программные коды, настраиваются автоматически благодаря библиотечным функциям. Поддержка разных типов датчиков будет добавляться по мере их выхода на рынок. Однако функции составления программных кодов на языке Arduino позволяют напрямую пользоваться всеми портами, если нужно с ними поработать. За исключением порта I2C все остальные можно настроить на ввод или вывод сигналов при помощи функции Arduino pinMode. Подробности см. в разделе справки по языкам (Language Reference) на **www.arduino.cc**.

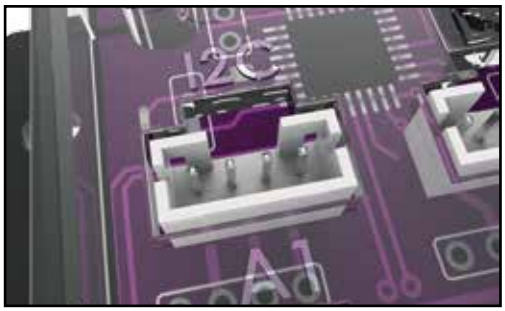


Рисунок 29: Порт для датчиков на контроллере PRIZM (Контакты слева направо: 1, 2, 3, 4.)

Таблица 1: Назначение контактов порта I2C

Контакт	Функция	Назначение контактов в *ПО Arduino (IDE)* ( )
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	SDA (последовательная передача данных через шину связи I2C)	Канал ввода ADC4 (A4)
Контакт 4	SCL (тактовый сигнал шины связи I2C)	Канал ввода ADC5 (A5)

Примечание: Порт I2C на контроллере PRIZM можно использовать только в режиме I2C. Переводить его в аналоговый или цифровой режим нельзя.

Таблица 2: Порт для аналоговых датчиков (A1)

Контакт	Функция	Назначение контактов в *ПО Arduino (IDE)* ( )
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод аналоговых или ввод-вывод цифровых сигналов	Ввод аналоговых сигналов (A1) | ввод-вывод цифровых сигналов (15)

Таблица 3: Порт для аналоговых датчиков (A2)

Контакт	Функция	Назначение контактов в *ПО Arduino (IDE)* ( )
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод аналоговых или ввод-вывод цифровых сигналов	Ввод аналоговых сигналов (A2) | ввод-вывод цифровых сигналов (16)

Примечание: Порты A1-A3 для аналоговых датчиков можно также настроить на ввод-вывод цифровых сигналов.

Таблица 4 Порт для аналоговых датчиков (A3)

Контакт	**Функция**	**Назначение контактов в *ПО Arduino (IDE)* ( )**
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод аналоговых или ввод-вывод цифровых сигналов	Ввод аналоговых сигналов (A3) | ввод-вывод цифровых сигналов (17)

Таблица 5: Порт для цифровых датчиков (D2)

Контакт	**Функция**	**Назначение контактов в *ПО Arduino (IDE)* ( )**
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Ввод-вывод цифровых сигналов	Ввод-вывод цифровых сигналов (9)
Контакт 4	Ввод-вывод цифровых сигналов	Ввод-вывод цифровых сигналов (2)

Таблица 6: Порт для цифровых датчиков (D3)

Контакт	**Функция**	**Назначение контактов в *ПО Arduino (IDE)* ( )**
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод-вывод цифровых сигналов	Ввод-вывод цифровых сигналов (3)

Таблица 7: Порт для цифровых датчиков (D4)

Контакт	**Функция**	**Назначение контактов в *ПО Arduino (IDE)* ( )**
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод-вывод цифровых сигналов	Ввод-вывод цифровых сигналов (4)

Таблица 8: Порт для цифровых датчиков (D5)

Контакт	**Функция**	**Назначение контактов в *ПО Arduino (IDE)* ( )**
Контакт 1	Заземление	Отсутствует
Контакт 2	+5 В, 100 мА	Отсутствует
Контакт 3	Неиспользуемый	Отсутствует
Контакт 4	Ввод-вывод цифровых сигналов	Ввод-вывод цифровых сигналов (5)

Примечание: Цифровой порт D2 можно также превратить в порт последовательной связи с программным управлением, используя для этого *Библиотеку функций последовательной передачи данных в ПО Arduino (IDE)*. Контакты D9 и D2 можно настроить на приём-передачу и обеспечить канал последовательной связи через порт.

Порты для сервоприводов на контроллере PRIZM

У контроллера PRIZM шесть портов с нумерацией от 1 до 6 для сервоприводов с управляемым положением вала. По каждому каналу для сервоприводов можно подать напряжения питания и сигналы управления для одного любительского сервопривода. Библиотека функций контроллера TETRIX PRIZM для ПО Arduino (IDE) делает всю сложную работу по управлению сервоприводами.

Примечание:
Рекомендуем следующие сервоприводы с сайта **www.TETRIXrobotics.com**: стандартный сервопривод HS-485HB с поворотом вала на 180 градусов (39197), четвертной сервопривод HS-785HB для лебёдки, с накладкой (39905) и четвертной сервопривод HS-755HB, с накладкой (39904).

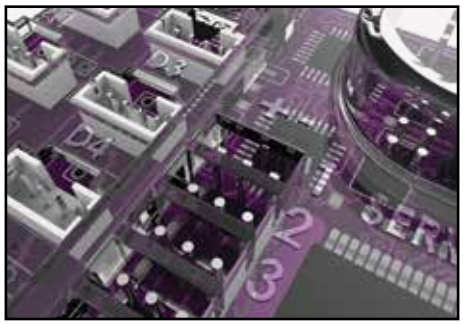


Рисунок 30: Порты для сервоприводов на контроллере PRIZM (Контакты слева направо: 1, 2, 3.) (См. маркировку SERVOS 1-6)

Контакт 1: Сигнал широтно-импульсной модуляции (ШИМ) сервопривода. У провода обычно жёлтый или белый цвет.

Контакт 2: Провод подачи напряжения на сервопривод. У провода красный цвет. Контроллер PRIZM выдаёт положительное напряжение +6 В.

Контакт 3: Провод заземления сервопривода. У провода чёрный цвет.

Подключение к контактам противоположной полярности не повредит контроллер PRIZM. Но если вставить его не так, как положено, сервопривод работать не будет. Контроллер PRIZM в общей сложности в состоянии подавать постоянный ток силой 6 А и напряжением 6 В на порты, предназначенные для сервоприводов. Потребление тока ограничено термоустройством (предохранительным размыкателем с самовозвратом в дежурное положение).

Порты для сервоприводов продолжительного вращения (CR) в контроллере PRIZM

У контроллера PRIZM есть два дополнительных порта для особых сервоприводов. К этим портам подсоединяют два сервопривода продолжительного вращения. Сервоприводы продолжительного вращения сконструированы так, что вращаются по часовой стрелке или против часовой стрелки. Их можно использовать в качестве уменьшенных, облегчённых редукторных электродвигателей постоянного тока. Порты CR1 и CR2 служат для подключения сервоприводов в таком исполнении и могут вращать их в продолжительном цикле в любом из двух указанных направлений по командам из библиотеки контроллера PRIZM.

Рисунок 31: Порты для сервоприводов продолжительного вращения (CR) в контроллере PRIZM (Контакты слева направо: 1, 2, 3.) (Маркировка: CR1 и CR2)

Контакт 1: Сигнал широтно-импульсной модуляции (ШИМ) сервопривода. У провода обычно жёлтый или белый цвет.
Контакт 2: Провод подачи напряжения на сервопривод. У провода красный цвет. Контроллер PRIZM выдаёт положительное напряжение +6 В.
Контакт 3: Провод заземления сервопривода. У провода чёрный цвет.
Подключение к контактам противоположной полярности не повредит контроллер PRIZM. Но если вставить его не так, как положено, сервопривод работать не будет. Контроллер PRIZM в общей сложности в состоянии подавать постоянный ток силой 6 А и напряжением 6 В на все порты для сервоприводов — в том числе порты CR1 и CR2. Потребление тока ограничено термоустройством (предохранительным размыкателем с самовозвратом в дежурное положение).

Порты для энкодеров на контроллере PRIZM

У контроллера PRIZM есть два входа под квадратурные энкодеры, позволяющие точно регулировать параметры в каналах управления электродвигателями постоянного тока № 1 и № 2. Энкодеры на электродвигателях постоянного тока TETRIX отслеживают угловое положение их валов; на основе этих данных программный код обеспечивает быстрое и точное регулирование угловой скорости и положения валов электродвигателей постоянного тока. Через порт ENC1 поступают сигналы энкодера для электродвигателя постоянного тока, подключенному к каналу № 1, а через порт ENC2 — сигналы энкодера в канал управления электродвигателем постоянного тока № 2. Мы добавили примеры программных кодов, в которых используются показания энкодеров, и все относящиеся к таким энкодерам библиотечные функции Arduino для контроллера PRIZM. Эти примеры программных кодов можно найти в выпадающем меню File > Examples > TETRIX_PRIZM.

Примечание: Энкодеры не входят ни в один из робототехнических наборов серии TETRIX MAX. При желании добавить этот элемент к своему набору можно купить комплект энкодера для электродвигателя серии TETRIX MAX (38000) на сайте **www.TETRIXrobotics.com**.

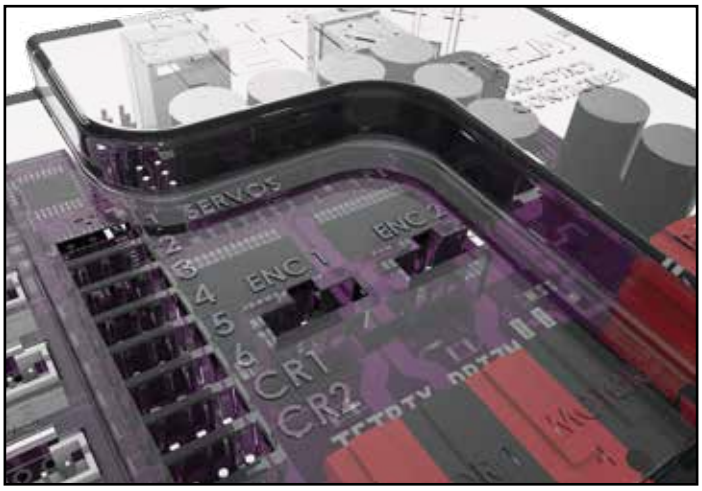


Рисунок 32: Порт под квадратурные энкодеры на контроллере PRIZM (Контакты слева направо: 1, 2, 3, 4.) (Обозначение: ENC1 и ENC2)

Контакт 1: источник постоянного напряжения +5 В

Контакт 2: сигнал с показаниями счётчика энкодера (A)

Контакт 3: заземление энкодера

Контакт 4: сигнал с показаниями счётчика энкодера (B)

Хотя гнездо разъёма имеет правильную полюсность, старайтесь вставлять штекер, не прилагая усилия, чтобы не вставить его наоборот. Если штекер с силой вставить наоборот, нарушится полюсность, что грозит повреждением энкодера и контроллера PRIZM.

Порт расширения на контроллере PRIZM

На контроллере PRIZM есть модульный разъём RJ-45 с меткой EXP, который соединён с встроенной шиной связи I2C. Этот порт служит для подсоединения дополнительных контроллеров электродвигателей постоянного тока и контроллеров сервоприводов TETRIX по гирляндной схеме. К порту расширения можно подсоединить до четырёх дополнительных контроллеров электродвигателей постоянного тока и/или сервоприводов в любом сочетании, и получить дополнительные каналы управления электродвигателями и сервоприводами. Каждый блок управления электродвигателем динамически настраивает собственный адрес I2C с учётом своего положения в гирляндной цепи. Первый блок должен использовать адрес "0x01", второй — "0x02", третий — "0x03", а четвёртый — "0x04".

Если либо к порту расширения, либо к порту для датчиков I2C подсоединены дополнительные датчики I2C или устройства, то этим устройствам нельзя использовать адреса I2C с 0x01 по 0x06. Указанные адреса предназначены для имеющихся внутри контроллера PRIZM процессорных микроконтроллеров, которые управляют электродвигателями и сервоприводами, а также для любых устройств управления электродвигателями или сервоприводами, дополнительно подсоединяемых по гирляндной цепи к порту расширения.

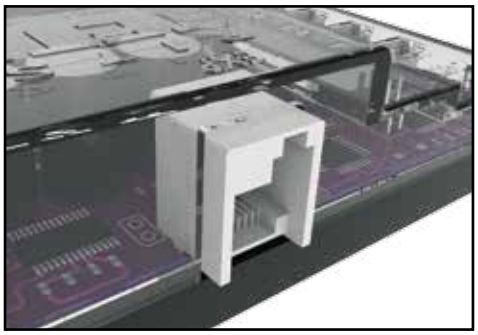


Рисунок 33: Порт расширения под дополнительные контроллеры электродвигателей на контроллере PRIZM
Смещённый модульный разъём RJ-45

Примечание: Порт не работает с устройствами, имеющими отношение к LEGO® MINDSTORMS®. Запрещается подсоединять к этому порту датчики, электродвигатели LEGO MINDSTORMS, контроллер EV3 или любое другое LEGO устройство.

Порт USB на контроллере PRIZM

Порт USB на контроллере PRIZM служит для связи между самим контроллером и компьютером под управлением ОС Windows, Mac или Linux. Его главное назначение — загрузка программного кода в память микроконтроллера. Может также обеспечивать обмен данными между контроллером PRIZM и компьютером с использованием протокола последовательной связи. Например, в *ПО Arduino (IDE)* есть прибор контроля последовательной передачи данных, в окне которого отображаются значения данных, полученных от контроллера PRIZM. Здесь можно отображать показания обычных датчиков или энкодеров, а при необходимости и другие виды данных о программах.

Возможность видеть значения данных зачастую очень помогает при отладке программного кода. Прибор контроля последовательной передачи данных можно использовать и для отправки информации в контроллер PRIZM. Примером этого может служить приведение контроллером PRIZM в движение электродвигателей постоянного тока или сервоприводов в ответ на нажатие кнопок на клавиатуре. Для задействования функций отслеживания данных или управления наторелые в программировании пользователи имеют возможность создать графические интерфейсы на платформах Java, Python или других платформах программирования.

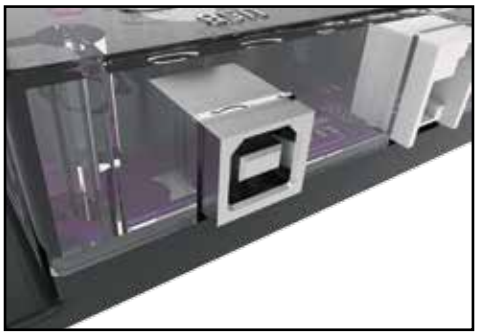


Рисунок 34: Порт USB на контроллере PRIZM, предназначенный для программирования и связи

Кнопка сброса параметров на контроллере PRIZM

Красная кнопка сброса параметров служит двум целям. Нажатие на неё прекращает выполнение любого программного кода и возвращает показания всех энкодеров и других датчиков в исходное состояние. Нажатие на неё точно так же перезапускает всю систему, как и выключение-включение питания.

Рисунок 35: Кнопка сброса параметров на контроллере PRIZM. Включается нажатием.

Пусковая кнопка на контроллере PRIZM

Зелёная пусковая кнопка (Start) служит для запуска программного кода, загруженного в контроллер PRIZM. Когда вы создаёте программный код посредством *ПО Arduino (IDE)*, функция PrizmBegin выполняет все команды, необходимые для запуска программы. Вызов этой функции приводит необходимые параметры в надлежащее состояние и задерживает выполнение программного кода до тех пор, пока не будет нажата зелёная пусковая кнопка.

Рисунок 36: Пусковая кнопка на контроллере PRIZM. Включается нажатием.

Порты для электродвигателей постоянного тока на контроллере PRIZM

На контроллере PRIZM два порта для подсоединения электродвигателей постоянного тока. Они обозначены надписями Motor 1 и Motor 2. Каждый канал используется для управления угловой скоростью и направлением вращения электродвигателей постоянного тока TETRIX при помощи команд, определение которых есть в *ПО Arduino (IDE)* контроллера PRIZM. На каждый канал управления электродвигателями подаётся постоянный ток силой 10 А и напряжением 12 В. Каждый канал управления электродвигателем снабжён красно-чёрным разъёмом, в котором красная часть имеет положительный потенциал, а чёрная — отрицательный. Силовые кабели к электродвигателям постоянного тока TETRIX снабжены парными цветными разъемами, которые следует вставлять в часть разъёма того же цвета, чтобы всё работало.

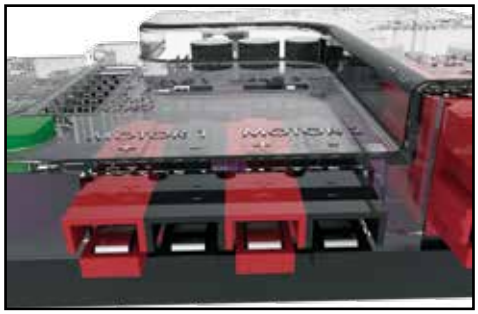


Рисунок 37: Порты для электродвигателей постоянного тока на контроллере PRIZM

Порт для подсоединения аккумуляторной батареи на контроллере PRIZM

Для питания контроллера PRIZM служит перезаряжаемая никель-металлогидридная аккумуляторная батарея на 12 В серии TETRIX. К контроллеру PRIZM прилагается комплект выключателя, предназначенный для подсоединения аккумуляторной батареи к соответствующему порту, выполненному в стилистике PRIZM. Аккумуляторную батарею можно подсоединить либо к верхнему, либо к нижнему ряду разъёмов. Порт расширения предназначен для подсоединения дополнительных контроллеров электродвигателей к контроллеру PRIZM и раздачи им напряжения по гирляндной цепи.

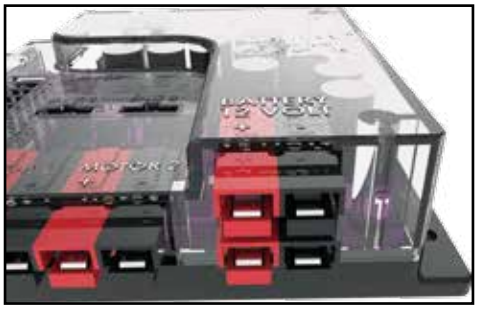


Рисунок 38: Порт ввода/вывода напряжения аккумуляторной батареи контроллера PRIZM

Библиотечные функции на языке Arduino для контроллера серии TETRIX® PRIZM™

Операторы включения

Оператор включения помогает добавить в скетч, написанный на языке Arduino, функциональные возможности библиотеки для контроллера серии TETRIX PRIZM. Библиотека для контроллера PRIZM представляет собой собрание функций, которые упрощают программирование контроллера PRIZM. Представьте себе библиотеку в виде ёмкости, заполненной мини-программами, каждая со своим уникальным именем, обозначающим исполняемую функцию, которое можно использовать в программном коде для выполнения сложных заданий. Чтобы включить библиотеку в скетч, написанный на языке Arduino, используйте оператор включения (#include) в начале программного кода. К примеру, чтобы добавить библиотеку для контроллера PRIZM в скетч на языке Arduino, следует вставить этот оператор вверху программы.

#include <PRIZM.h>

Оператор приказывает компилятору Arduino включить в оставляемый скетч на языке Arduino все программные коды, содержащиеся в библиотеке для контроллера PRIZM.

Объявления объекта

Объявление объекта — важный оператор при использовании контроллера PRIZM и библиотеки для контроллера PRIZM. Чтобы воспользоваться функциями из библиотеки для контроллера PRIZM, сначала нам необходимо объявить имя объекта библиотеки, который после этого вставляется в виде приставки перед каждой библиотечной функцией. Для этого можно сразу за оператором включения (#include) вставить следующий оператор.

PRIZM *myname*;

Параметр *myname* произволен — в соответствии с нашим выбором или тем, что для нас будет иметь смысл, он может стать чем угодно в пределах создаваемой нами области применения кода. В этом руководстве по программированию ради единообразия параметр "*myname" получит имя "*prizm", а это значит, что объявление объекта, используемое в примерах скетча программного кода, будет выглядеть так:

PRIZM prizm;

Вышеуказанный оператор создаёт в библиотеке объект под именем "prizm", используемый в коде при вызове функции из того класса функций контроллера PRIZM, в котором этот объект был создан. В нашем коде при всяком вызове какой-либо библиотечной функции контроллера PRIZM придётся снабдить приставкой каждую функцию с определённым именем объекта. В случае вышеупомянутого оператора объектом будет “prizm”. *У всех приведённых ниже библиотечных функций есть приставка в виде параметра "myname"*. Не забудьте, используя эти функции в нашем коде, заменить имя параметра "*myname" тем, которое мы дали объекту в библиотеке контроллера* PRIZM.

И, как уже отмечалось, в примерах программного кода в этом руководстве используется имя объекта "prizm". Кроме того, следует пояснить, что любая формулировка, перед которой стоит значок "//" представляет собой всего лишь заметку или пояснение и не включается в программный код при составлении скетча. Есть полезный приём: описывать и пояснять каждое действие, связанное с программным кодом. Это показывает, что вы понимаете, что делаете.

Функции инициализации

При помощи функций инициализации контроллера TETRIX PRIZM начинают или, при необходимости, завершают программу.

***myname*.PrizmBegin();**

Эта функция всегда вызывается в разделе настройки() скетча на языке Arduino. Функция инициализирует значимые параметры, и разрешает использование пусковой кнопки на контроллере PRIZM. При создании скетчей для контроллера PRIZM её использование обязательно.

Примеры:

```
prizm.PrizmBegin();                    // начать работу контроллера PRIZM
```

***myname*.PrizmEnd();**

Эта функция заканчивает или прекращает программу, возвращает значимые параметры в исходное состояние и отправляет программу к исходной точке.

Примеры:

```
prizm.PrizmEnd();                      // в случае вызова прекратит выполнение программы
```

Порты для датчиков на контроллере TETRIX PRIZM служат для подключения разных видов датчиков и обмена цифровыми и аналоговыми сигналами. Для подключения цифровых датчиков к контроллеру PRIZM служат порты D2-D5. Цифровой порт D2 можно также превратить в порт последовательной связи с программным управлением, используя для этого библиотеку соответствующего ПО на языке Arduino. Все цифровые порты можно настроить на ввод или вывод сигналов. Аналоговые датчики подсоединяются к контроллеру PRIZM через порты A1-A3. Порты для аналоговых датчиков можно настроить на ввод аналоговых и вывод цифровых сигналов. Есть также один порт для датчика I2C. Порт I2C соединён с коммуникационной шиной I2C контроллера PRIZM. Адреса I2C (0x01-0x06) сохраняются внутри контроллера PRIZM, их нельзя использовать для наружного датчика. Средства поддержки датчиков будут добавляться в контроллер PRIZM постоянно. О средствах поддержки и наличии датчиков см. **www.TETRIXrobotics.com**.

***myname*.readLineSensor(номер порта);**

Функция считывает состояние датчика линии и возвращает значение "1" или "0". Параметр *порт#* определяет, к какому цифровому порту подсоединять датчик. Датчик чаще всего используется для обнаружения разницы между чёрной и белой поверхностями. Настроить робота на движение по краю тёмной линии на белом фоне или по краю белой линии на тёмном фоне — очень распространённый вариант программирования. С этой целью датчик ищет отражение инфракрасного луча от поверхности, к которой обращён. Если датчик обращён к светлой или светоотражающей поверхности, то датчик обнаружит отражающийся от неё луч света. Если датчик обращён к тёмной поверхности или находится слишком далеко от обследуемой поверхности, тогда датчик не обнаружит отражения луча света. У датчика линии есть регулировочный потенциометрический винтик на тыльной стороне — с его помощью регулируют светочувствительность датчика. Если датчик обнаружит поверхность, которая не отражает свет (отражение луча не улавливается, значит, у фона или линии тёмный цвет), тогда функция вернёт значение "1", то есть "линия обнаружена". Если датчик улавливает отражённый луч (поверхность белая или отражает свет), тогда функция вернёт значение "0", то есть "линия не обнаружена". Развёрнутый пример применительно к датчику линии см. ниже или в библиотеке для контроллера PRIZM.

Примеры:

```
int x;                                 // задать целочисленную переменную с именем x
x = prizm.readLineSensor(2);           // считать состояние искателя линии, подключённого к цифровому
                                       // порту (D2), в переменную x
```

или

```
Serial.print(prizm.readLineSensor(2)); // считать состояние искателя линии, подключённого к цифровому
                                       // порту (D2), и распечатать его на
                                       // экране прибора контроля последовательной
                                       // передачи данных Arduino
```

***myname*.readSonicSensorCM(порт#);**

Функция считывает и возвращает расстояние до объекта, находящегося перед ультразвуковым датчиком. Параметр *порт#* определяет, к какому порту для цифровых датчиков подсоединяется датчик. Ультразвуковой датчик представляет собой модуль бесконтактного измерения расстояния, с помощью которого наш робот может обрести зрение, обнаруживать на своём пути объекты и объезжать их. Частота датчика модулируется на уровне 42 кГц, а диапазон измерения составляет от 3 до 400 см. Датчик испускает короткую серию звуковых импульсов и ждёт, когда они вернутся, отразившись от некоего объекта в пространстве. По времени, прошедшему между испусканием и возвращением серии звуковых импульсов, определяется расстояние до объекта.

Примеры:

```
int x;                                  // задать целочисленную переменную x
x = prizm.readSonicSensorCM(2);         // считать и сохранить в переменной x расстояние в
                                        // сантиметрах до объекта, обнаруженного
                                        // перед ультразвуковым датчиком, подсоединённым к
                                        // цифровому порту (D2)
```

или

```
Serial.print(prizm.readSonicSensorCM(2)); // считать расстояние в сантиметрах до
                                          // объекта, обнаруженного перед
                                          // ультразвуковым датчиком, подсоединённым к цифровому
                                          // порту (D2) и распечатать его на экране прибора
                                          // контроля последовательной передачи данных Arduino
```

***myname*.readSonicSensorIN(порт#);**

Функция считывает и возвращает расстояние в дюймах до объекта перед ультразвуковым датчиком. Параметр *порт#* определяет, к какому порту для цифровых датчиков подсоединяется датчик. Ультразвуковой датчик представляет собой модуль бесконтактного измерения расстояния, с помощью которого наш робот может обрести зрение, обнаруживать на своём пути объекты и объезжать их. Частота датчика модулируется на уровне 42 кГц, а диапазон измерения составляет от 2 до 150 дюймов. Датчик испускает короткую серию звуковых импульсов и ждёт, когда они вернутся, отразившись от некоего объекта в пространстве. По времени, прошедшему между испусканием и возвращением серии звуковых импульсов, определяется расстояние до объекта.

Примеры:

```
int x;                                  // задать целочисленную переменную x
x = prizm.readSonicSensorIN(порт#);     // считать и сохранить в переменной x расстояние
                                        // в дюймах до объекта, обнаруженного
                                        // перед ультразвуковым датчиком,
                                        // подсоединённым к порту для цифровых датчиков (D2)
```

или

```
Serial.print(prizm.readSonicSensorIN(2)); // считать расстояние в дюймах до
                                          // объекта, обнаруженного перед
                                          // перед ультразвуковым датчиком, подсоединённым
                                          // к цифровому порту (D2) и распечатать на
                                          // экране прибора контроля последовательной
                                          // передачи данных Arduino
```

Напряжение аккумуляторной батареи

Аккумуляторная батарея TETRIX выдаёт номинальное постоянное напряжение 12 В при электроёмкости 3 000 мАч. Полностью заряженная аккумуляторная батарея способна выдать пиковое напряжение до 15 В. Аккумуляторная батарея считается разряженной, когда напряжение опускается ниже 12,1 В. Батарея также снабжена сменным плавким предохранителем на 20 А. На всех устройствах с аккумуляторными батареями TETRIX должны использоваться фирменные предохранители, которые гарантируют безопасность. Следить за напряжением аккумуляторной батареи можно при помощи программного кода, вызывая функцию считывания напряжения батареи.

***myname*.readBatteryVoltage();**

Функция считывает и возвращает напряжение аккумуляторной батареи TETRIX, подсоединённой к соответствующему порту контроллера PRIZM. Возвращаемое значение представляет собой целое число, делимое на 100 для масштабирования фактического напряжения батареи. Так, значение 918 соответствует фактическому напряжению 9,18 В.

Примеры:

```
int x;                                  // задать целочисленное значение x

x = prizm.readBatteryVoltage();         // считать напряжение батареи в переменную x
```

или

```
Serial.print(prizm.readBatteryVoltage());   // считать напряжение аккумуляторной батареи и распечатать его
                                            // на экране прибора контроля последовательной передачи данных
Arduino
```

Использование пусковой кнопки

***myname*.readStartButton();**

Зелёную пусковую кнопку можно использовать не только для запуска исполнения программного кода. После использования для запуска программы её можно в любое время заново считать для выполнения других заданий. Так программа становится более управляемой. Например, два разных действия робота разнесены по двум разным частям программного кода, и при помощи пусковой кнопки запускается первая часть, а после повторного нажатия происходит быстрый переход ко второй части кода. После первого нажатия на кнопку функция должна вернуть значение "1", а после повторного нажатия — "0".

Примеры:

```
int x;                                  // задать целочисленное значение с именем x

x = prizm.readStartButton();            // считать состояние пусковой кнопки в переменную x
```

или

```
While (prizm.readStartButton() == 0){   // ждать здесь в цикле "while", пока пусковая
                                        // кнопка не будет нажата (возвращает "1")
}
```

Встроенные светодиоды

У контроллера PRIZM есть два сигнальных светодиода, работой которых можно управлять в коде. Красный и зелёный светодиоды загораются и гаснут, получая по отдельности команды HIGH или LOW.

***myname*.setGreenLED(HIGH);** // включает встроенный светодиодный индикатор зелёного цвета

***myname*.setGreenLED(LOW);** // выключает встроенный светодиодный индикатор зелёного цвета

Примеры:

prizm.setGreenLED(HIGH); // включает зелёный светодиод контроллера PRIZM

prizm.setGreenLED(LOW); // выключает зелёный светодиод контроллера PRIZM

***myname*.setRedLED(HIGH);** // включает встроенный светодиодный индикатор красного цвета

***myname*.setRedLED(LOW);** // выключает встроенный светодиодный индикатор красного цвета

Примеры:

prizm.setRedLED(HIGH); // включает красный светодиод контроллера PRIZM

prizm.setRedLED(LOW); // выключает красный светодиод контроллера PRIZM

Функции для электродвигателей постоянного тока

Эти функции служат для управления электродвигателями постоянного тока серии TETRIX PRIZM, подсоединёнными к выходным портам Motor 1 (электродвигатель № 1) и Motor 2 (электродвигатель № 2). Есть три разных вида управления. Регулятор мощности задаёт уровень мощности, а также направление вращения электрической машины. Чтобы функции скорости и приведения в заданное конечное положение работали, квадратурный энкодер TETRIX должен быть установлен на электродвигателе в качестве датчика обратной связи: тогда возможно точное регулирование скорости и задаваемого положения по алгоритму ПИД-регулирования.

***myname*.setMotorPower(номер электродвигателя, мощность);**

Функция служит для регулирования уровня мощности и направления вращения электродвигателей постоянного тока TETRIX, подсоединяемых к портам Motor 1 или Motor 2, обеспечивающим каналы управления. Параметр *номер электродвигателя* выбирает канал управления электродвигателем, а параметр *мощность* задаёт уровень мощности. Параметр номер электродвигателя может быть либо номером, либо переменной 1 или 2. Параметр *мощность* определяет величину мощности, с которой следует вращать вал электродвигателя, и устанавливается в пределах от 0 до 100. Это процентный показатель, то есть "0" означает полное отключение, а "100" означает включение на 100%-ю или полную мощность. Направление вращения задаётся знаком перед параметром мощности.

Примеры:

prizm.setMotorPower(1, 100); // вращать электродвигатель № 1 по ЧС с мощностью 100 %

prizm.setMotorPower(1, -100); // вращать электродвигатель № 1 против ЧС с мощностью 100 %

Кроме того, для остановки электродвигателя есть два режима: остановка выбегом и остановка торможением. Если задана мощность "0", то электродвигатель остановится после выбега. Если задана мощность "125", то электродвигатель остановится после торможения.

Примеры:

prizm.setMotorPower(2, 0); // остановить электродвигатель № 2 выбегом; 0 = выбег

или

prizm.setMotorPower(2, 125); // остановить электродвигатель № 2 торможением; 125 = торможение

***myname*.setMotorPowers(мощность 1-го электродвигателя, мощность мотора 2-го электродвигателя);**

Стоит задать параметры *power1* и *power2* в виде процентов от уровня мощности, как оба канала немедленно отреагируют. Запомните, что знак (+/-) перед уровнем мощности определяет направления вращения.

Примеры:

prizm.setMotorPowers(50, -50); // вращать электродвигатель № 1 по ЧС с мощностью 50 %, а
// электродвигатель № 2 против ЧС с мощностью 50 %

***myname*.setMotorSpeed(номер электродвигателя, скорость);**

Функция служит для точного регулирования угловой скорости электродвигателя постоянного тока TETRIX при помощи квадратурного энкодера с ПИД-регулированием. На электродвигателе постоянного тока должен быть установлен энкодер, подсоединённый к соответствующему гнезду с меткой "ENC1" или "ENC2" на контроллере PRIZM; чтобы функция работала, энкодер должен быть совместим с подчинённым ему электродвигателем. Энкодер возвращает показания счётчика в контроллер PRIZM, который считывает эти показания и на их основе выполняет сложный алгоритм регулирования с приставкой "ПИД", означающей *"пропорциональный-интегральный-дифференциальный"*. ПИД-алгоритм обеспечивает очень устойчивое и точное управление объектами и будет пытаться сохранить постоянной угловую скорость электродвигателя даже при изменении нагрузки на него. Благодаря ПИД-алгоритму колёса и механизмы, приводимые в движение электродвигателями постоянного тока TETRIX, можно чётко управлять, устанавливая точно в заданное положение. Параметр *электродвигатель#* может иметь значение "1" или "2". Параметр *угловая скорость* представляет собой задаваемую скорость в градусах в секунду (-1440), с которой должен вращаться наш электродвигатель. Возможный диапазон значений: от минимального, приблизительно равного 10, до максимального, равного 720 °/с. Направление вращения электродвигателя задаётся знаком перед параметром скорости. Значения за указанными пределами могут вызвать нарушение в работе устройств.

Примеры:

prizm.setMotorSpeed(1, 360); // вращать электродвигатель № 1 по ЧС со скоростью 360 °/с

или

prizm.setMotorSpeed(1, -360); // вращать электродвигатель № 1 против ЧС со скоростью 360
// °/с

***myname*.setMotorSpeeds(скорость1, скорость2);**

Функция использует ПИД-регулирование и не отличается от предыдущей, только что объяснённой, функции почти ничем, кроме того, что задаёт параметры *speed1* и *speed2* для каналов электродвигателя № 1 и электродвигателя № 2 одновременно. Оба подчинённых электродвигателя должны быть снабжены энкодером TETRIX, подсоединённым к соответствующим гнёздам с метками ENC1 и ENC2 на контроллере PRIZM.

Примеры:

prizm.setMotorSpeeds(360, -180); // вращать электродвигатель № 1 по ЧС со скоростью 360 °/с, а также
// вращать электродвигатель № 2 против часовой стрелки
// со скоростью 180 °/с

***myname*.setMotorTarget(электродвигатель#, скорость, заданное конечное значение);**

Помимо регулирования скорости, эта функция добавляет ещё одну характеристику, используя для управления скоростью и приведением объекта управления в заданное положение ПИД-алгоритм. Поскольку электродвигатель управляется согласно настройкам параметра *электродвигатель#*, он должен быть оснащён энкодером TETRIX, подсоединённым к соответствующему порту подчинённого электродвигателя. Эта функция будет вращать подчинённый ей электродвигатель постоянного тока так, чтобы сообщаемое энкодером значение сравнялось с заданным конечным значением. Электродвигатель будет вращаться в заданном направлении с угловой скоростью, указанной в параметре *скорость* (в градусах в секунду). Дойдя до места, соответствующего показаниям счётчика энкодера, указанному в параметре *заданное конечное положение*, электродвигатель

задерживается в этом положении, как это делает сервопривод, до получения новой команды. Эта функция игнорирует знак плюса/минуса перед параметром *скорость*. Взамен направление вращения задаётся указанным в команде энкодера конечным значением параметра *заданное конечное положение*. Пока выполняется эта функция, можно считывать функцию readMotorBusy(), которая покажет, что электродвигатель занят продвижением к заданному ему конечному положению. Возвращение функцией сигнала "HIGH" или "1" означает, что канал управления подчинённым электродвигателем занят (движением к заданному конечному положению). По достижении заданного конечного положения функция readMotorBusy() должна вернуть сигнал "LOW" или "0". Разрешающая способность энкодера: 1440 отсчётов на оборот вала электродвигателя или 1/4 градуса на отсчёт.

Примеры:

```
prizm.setMotorTarget(1, 360, 1440);        // вращать электродвигатель № 1 по ЧС со скоростью 360 °/с,
                                           // остановить и удерживать в занятом положении, когда показания
                                           // энкодера составят 1440 (один оборот)

prizm.setMotorTarget(2, 180, -1440);       // вращать электродвигатель № 1 против ЧС со скоростью 180
                                           // °/с; остановить и удерживать в занятом положении, когда
                                           // показания счётчика энкодера равны -1440
```

***tуpame*.setMotorTargets(скорость1, *заданное конечное положение*1, скорость2, *заданное конечное положение*2);**

Эта функция осуществляет ПИД-регулирование скорости и положения по каналам управления обоими электродвигателями постоянного тока. Эта функция одновременно задаёт параметры режима обоих электродвигателей (1 и 2), чтобы обеспечить одновременный и равномерный запуск. Одновременность запуска электродвигателей имеет в некоторых случаях решающее значение. Конечные значения скорости и остановки для электродвигателя № 1 задаются параметрами *скорость1* и *заданное конечное положение1*. Конечные значения скорости и остановки для электродвигателя № 2 задаются параметрами *скорость2* и *заданное конечное положение2*. Эта функция использует ПИД-регулирование для каналов обоих электродвигателей постоянного тока. Энкодеры должны быть установлены на обоих электродвигателях и подсоединены к соответствующим портам ENC1 и ENC2 на контроллере PRIZM. Эта функция будет вращать оба электродвигателя постоянного тока так, чтобы сообщаемые энкодером значения сравнялись с заданными конечными значениями. Электродвигатели будут вращаться в направлении заданных конечных положений со скоростями, указанными в параметрах *скорость1* и *скорость2* (в градусах в секунду). Достигнув конечного положения, заданного в параметрах *заданное конечное положение1* и *заданное конечное положение2*, каждый электродвигатель задержится в достигнутом положении, как это делает сервопривод, до получения новой команды. Эта функция игнорирует знак плюса/минуса перед параметрами скорость1 и скорость2. Направление вращения задаётся указанными в команде энкодера конечными значениями параметров *заданное конечное положение1* и *заданное конечное положение2*. Пока выполняется эта функция, можно считывать функцию readMotorBusy(), которая покажет, что электродвигатель занят продвижением к заданному ему конечному положению. Возвращение этой функцией сигнала "HIGH" или "1" означает, что канал управления подчинённым электродвигателем занят (движением к заданному конечному положению). По достижении заданного конечного положения функция readMotorBusy() должна вернуть сигнал "LOW" или "0". Разрешающая способность энкодера: 1440 отсчётов на оборот вала электродвигателя или 1/4 градуса на отсчёт.

Примеры:

```
prizm.setMotorTargets(360, 1440, 180, -1440);      // эта команда будет вращать электродвигатель № 1
                                                   // по часовой стрелке со скоростью 360 °/с, который
                                                   // остановится и будет удерживать занятое положение, получив
                                                   // от энкодера показания счётчика, равные 1440. Электродвигатель № 2
                                                   // будет вращаться против часовой стрелки со скоростью 180
```

```
                                    // °/с, остановится и задержится в занятом положении
                                    // при получении от энкодера показаний счётчика, равных -1440.
```

***myname*.setMotorDegree(электродвигатель#, скорость, градусы);**

Эта функция требует, чтобы подчинённый электродвигатель постоянного тока был снабжён энкодером — тогда она сможет применить ПИД-алгоритм для регулирования скорости и положения в соответствии с заданным конечным положением. Эта функция идентична функциям задания "конечного положения", объяснение которых уже дано, за исключением того, что конечные положения задаются в градусах, а не в отсчётах энкодера. Управление моторами по градусам больше основано на интуиции, чем на действительные показания счётчика энкодера. Параметр *электродвигатель#* задаёт функцию, которая будет управлять либо электродвигателем № 1, либо электродвигателем № 2. Параметр *скорость* представляет собой скорость, измеряемую в градусах в секунду, которую должен поддерживать подчинённый электродвигатель. Параметр *градусы* задаёт в градусах угловое положение, в котором вал электродвигателя должен остаться после вращения и остановки. После вызова эта функция будет вращать подчинённый ей электродвигатель постоянного тока, чтобы считываемое энкодером значение в градусах сравнялось с заданным значением в градусах. Вал электродвигателя будет вращаться, стремясь занять заданное конечное положение, с угловой скоростью, указанной в параметре *скорость* (в градусах в секунду). Дойдя до заданного конечного положения, вал этого электродвигателя задержится в нём, как это делает сервопривод, до получения новой команды. Эта функция игнорирует значок плюса/минуса перед значением параметра *скорость*. Направление вращения задаётся содержащимся в команде энкодера конечным положением в *градусах*. Пока выполняется эта функция, можно считывать функцию readMotorBusy(), которая покажет, что электродвигатель занят продвижением к заданному ему конечному положению. Возвращение этой функцией сигнала "1" означает, что канал управления подчинённым электродвигателем занят (движением к заданному конечному положению). По достижении заданного конечного положения функция readMotorBusy() должна вернуть "0". Разрешающая способность энкодера в градусах в составляет 1 градус.

Примеры:

```
prizm.setMotorDegree(1, 360, 180);  // повернуть канал электродвигателя постоянного тока № 1 по ЧС
                                    // со скоростью 360 градусов в секунду до заданного конечного
                                    // положения на 180 градусов (1/2 оборота) и остановить
                                    // по достижении заданного конечного положения остановить
                                    // электродвигатель и удерживать его в занятом положении
                                    // на метке 180 градусов
prizm.setMotorDegree(2, 180, -360); // повернуть канал электродвигателя постоянного тока № 1 против ЧС
                                    // со скоростью 180 градусов в секунду до
                                    // заданного конечного положения -360 градусов или
                                    // (1 оборот) и остановить; когда заданное конечное положение будет
                                    // достигнуто, остановить электродвигатель и удержать в занятом
                                    // положении на метке -360 градусов
```

***myname*.setMotorDegrees(скорость1, градусы1, скорость2, градусы2);**

Эта функция одновременно задаёт конечные скорость и градусы каналам управления обоих электродвигателей. Эта функция требует, чтобы оба подчинённых электродвигателя постоянного тока были снабжены энкодерами — тогда она сможет применить ПИД-алгоритм для управления скоростью и положением в соответствии с заданными конечными значениями. Эта функция идентична функциям задания "конечных положений", объяснение которых уже дано, за исключением того, что конечные положения задаются в градусах, а не в отсчётах энкодера. Управление моторами по градусам больше основано на интуиции, чем на действительные показания счётчика энкодера. Параметр *скорость1* задаёт скорость вращения электродвигателю № 1 в градусах в секунду. Параметр *градусы1* представляет собой положение в градусах, в котором вал электродвигателя должен остаться после вращения и остановки. Параметр *скорость2* задаёт скорость вращения электродвигателю № 2 в градусах в секунду. Параметр *градусы2* задаёт в градусах угловое положение, в котором вал электродвигателя № 2 должен остаться после вращения и остановки. В случае вызова эта функция будет вращать подчинённый ей канал

электродвигателя постоянного тока, чтобы считываемое энкодером значение в градусах сравнялось с заданным значением в градусах. Электродвигатели будут вращаться в направлении заданных конечных положений со скоростями, указанными в параметрах *скорость1* и *скорость2* (в градусах в секунду). Дойдя до заданного конечного положения, вал электродвигателя задержится в нём, как это делает сервопривод, до получения новой команды. Эта функция игнорирует знак плюса/минуса перед параметрами скорость1 и скорость2. Направление вращения задаётся указанными в команде энкодера конечными значениями параметров *градусы1* и *градусы2*. Пока выполняется эта функция, можно считывать функцию readMotorBusy(), которая покажет, что электродвигатель занят продвижением к заданному конечному положению. Возвращение этой функцией сигнала "1" означает, что канал управления подчинённым электродвигателем занят (движением к заданному конечному положению). По достижении заданного конечного положения функция readMotorBusy() должна вернуть "0". Разрешающая способность энкодера в градусах в составляет 1 градус.

Примеры:

```
prizm.setMotorDegrees(360, 360, 180, -360);  // повернуть электродвигатель № 1 по ЧС со скоростью 360
                                              // градусов в секунду и остановить по достижении
                                              // определяемом энкодером конечном положении 360
                                              // градусов; повернуть электродвигатель № 2 против часовой стрелки
                                              // со скоростью 180 градусов в
                                              // секунду и остановить в определяемом энкодером заданном
                                              //  конечном положении -360 градусов; каждый
                                              // электродвигатель должен задержаться в достигнутом положении,
                                              // когда оно совпадёт с заданным ему конечным положением
```

***myname*.setMotorInvert(электродвигатель#, обратное действие);**

При помощи этой функции можно сделать обратной схему привязки прямого/обратного направления для каналов обоих электродвигателей постоянного тока и поступающих в них сигналов энкодеров. Функция предназначена для согласования прямого и обратного вращения электродвигателей на противоположных бортах шасси робототехнической модели с бортовым поворотом. Параметр *электродвигатель#* задаёт канал электродвигателя 1 или 2. Значение параметра обратного действия задаёт условие; *обратное действие* = 0 означает отсутствие обратного действия, а *обратное действие* = 1 заставляет электродвигатель и энкодер выполнить действие обратное тому, которое задано каналом управления. По умолчанию при включении контроллера действует режим отсутствия инвертирования. Вызвать эту функцию придётся только один раз, в разделе настройки скетча на языке Arduino. Но вызвать её и изменить в программном коде можно в любое время.

Примеры:

```
prizm.setMotorInvert(1, 1);          // вращать в обратную сторону электродвигатель № 1
prizm.setMotorPower(1, 50);          // обычно эта функция заставляет электродвигатель № 1
                                     // вращаться с мощностью 50 % по часовой стрелке
                                     // но, поскольку мы вызвали функцию обратного действия, то
                                     // на самом деле электродвигатель № 1 будет вращаться
                                     // против часовой стрелки (-50); та же схема привязки направления
                                     // действует в отношении всех функций мощности электродвигателя,
                                     // скорости и функций задания конечного положения, когда
                                     // вызывается функция обратного действия
```

***myname*.readMotorBusy(электродвигатель#);**

Функция должна вернуть значение, равное "0" или "1" в зависимости от состояния канала электродвигателя постоянного тока, находящегося в режиме движения к заданному конечному положению. То есть любой электродвигатель постоянного тока, управляемый при помощи энкодера и вращающийся согласно показаниям счётчика энкодера или заданному конечному положению в градусах, сообщит о своём состоянии после опроса при помощи этой функции. Параметр *электродвигатель#* задаёт каналу электродвигателя значение 1 или 2. Функция должна вернуть значение "1", если электродвигатель движется к заданному ему конечному положению. Функция должна вернуть значение "0", если электродвигатель достиг заданного ему конечного положения. Возможность запросить в коде состояние электродвигателя, движущегося к заданному конечному положению, может быть очень полезным, так как помогает дождаться, пока электродвигатель достигнет заданного ему конечного положения и лишь потом перейти к следующей команде.

Примеры:

```
int x;                                  // определить целочисленную переменную x

x = prizm.readMotorBusy(1);             // считать возвращённый сигнал состояния электродвигателя № 1 в x
```

или

```
x = prizm.readMotorBusy(2);             // считать возвращённый сигнал состояния электродвигателя № 2 в x
```

или

```
Serial.print(prizm.readMotorBusy(1));   // печатать состояние электродвигателя № 1 на
                                        // экране прибора контроля последовательной передачи данных
Arduino
```

***myname*.readMotorCurrent(электродвигатель#);**

Эта функция считывает величину постоянного тока, потребляемого электродвигателем постоянного тока TETRIX, подсоединённого к порту Motor 1 или Motor 2. Параметр *электродвигатель#* задаёт каналу электродвигателя значение 1 или 2. Величина потребляемого электродвигателем тока будет зависеть от величины нагрузки. Более тяжёлая нагрузка на электродвигатель приведёт к большему потреблению тока. Во избежание повреждения электродвигателя важно не подвергать его продолжительным высоким нагрузкам. Из-за высокого потребления тока электродвигатель постоянного тока нагревается и со временем может выйти из строя. При помощи этой функции можно ограничить потребление тока электродвигателем. Образец полного кода для ограничения потребления тока см. в примере программы, включённой в библиотеку контроллера PRIZM.

Примеры:

```
int x;                                  // создать целочисленную переменную x

x = prizm.readMotorCurrent(1);          // считать ток электродвигателя № 1 и сохранить его в x
```

или

```
x = prizm.readMotorCurrent(2);          // считать ток электродвигателя № 2 и напечатать его на экране
                                        // прибора контроля последовательной передачи данных; значение,
                                        // которое эта функция возвращает, выражается целым числом
                                        // и представляет собой ток в миллиамперах; например, значение
                                        // 1500 равно 1,5 A
```

Функции для энкодера

Квадратурные энкодеры для электродвигателей TETRIX позволяют точно управлять скоростью и положением электродвигателей постоянного тока TETRIX. Контроллер PRIZM будет считывать показания счётчика энкодера и выполнять внутренний ПИД-алгоритм. Заложенный к контроллер PRIZM алгоритм ПИД-регулирования настроен на обслуживание электродвигателей постоянного тока TETRIX в сочетании с квадратурными энкодерами TETRIX. Квадратурные энкодеры TETRIX (38000) производят 360 отсчётов на оборот (отсч/об) и 1440 импульсов на оборот (имп/об). Это означает угловое разрешение в 1/4 градуса для вала электродвигателя постоянного тока.

***myname*.readEncoderCount(энк#);**

Эта функция считывает и возвращает показания счётчика квадратурного энкодера TETRIX, подсоединённого к соответствующему порту ENC1 или ENC2. Показания счётчика возвращаются в виде целого числа типа long. Параметр *энк#* может иметь значение "1" или "2". Значение, равное "1", сообщит показания счётчика энкодера 1 (ENC1). Значение *энк#*, равное "2", сообщит показания счётчика энкодера 2 (ENC2). Для справки: 1440 отсчётов = 1 оборот вала электродвигателя.

Примеры:

```
long x;                                  // задать переменную x в виде длинного целого
x = prizm.readEncoderCount(1);           // считать и сохранить показания счётчика ENC1 в виде длинной
                                         // переменной x
```

или

```
Serial.print(prizm.readEncoderCount(2);  // печатать показания счётчика ENC2 на
                                         // экране прибора контроля последовательной передачи данных Arduino
```

***myname*.readEncoderDegrees(энк#);**

Эта функция считывает и возвращает показания квадратурного энкодера TETRIX, подсоединённого к соответствующему порту ENC1 или ENC2, в градусах. Возвращается значение в виде длинного целого. Параметр *энк#* может иметь значение "1" или "2". Значение "1" сообщит угловое положение ENC1 в градусах. Значение "2" сообщит угловое положение ENC2 в градусах. Использование функции градусов для слежения за вращением электродвигателя может потребовать большей опоры на интуицию, чем на приблизительные показания счётчика энкодера. Хотя приблизительное число отсчётов должно дать разрешение в 1/4 градуса, разрешающая способность функции градусов ограничена разрешением в 1 градус.

Примеры:

```
long x;                                  // задать переменную x в виде длинного целого
x = prizm.readEncoderDegrees(1);         // задать переменную x в виде длинного целого; считать и
                                         // сохранить значение градусов для ENC1 в длинной переменной x
```

или

```
Serial.print(prizm.readEncoderDegrees(2)); // напечатать значение градусов для ENC2 на
                                         // экране прибора контроля последовательной передачи данных Arduino
```

***myname*.resetEncoder(энк#);**

В случае вызова эта функция обнуляет показания счётчика энкодера, заданные в параметре энк# (1 или 2).

Примеры:

```
prizm.resetEncoder(1);                   // обнулить счётчик энкодера № 1 (ENC1)
prizm.resetEncoder(2);                   // обнулить счётчик энкодера № 2 (ENC2)
```

***myname*.resetEncoders();**

В случае вызова эта функция обнуляет счётчики энкодеров №№ 1 и 2.

Примеры:

```
prizm.resetEncoders();                   // обнулить оба энкодера (1 и 2)
```

Функции для сервоприводов

Выпускается два вида сервоприводов TETRIX. Стандартному сервоприводу можно дать команду повернуться на любой угол в диапазоне от 0 до 180 градусов. Дойдя до заданного угла, он встанет, не реагируя на попытки сдвинуть вал с места. У контроллера PRIZM есть шесть портов с нумерацией от 1 до 6, предназначенных для сервоприводов. Через каждый порт можно управлять одним стандартным сервоприводом с шестью независимыми каналами управления сервоприводами. Другой вид сервопривода — сервопривод продолжительного вращения (обозначается CR). Сервопривод продолжительного вращения вращается по часовой стрелке или против часовой стрелки аналогично электродвигателю постоянного тока. У контроллера PRIZM есть два специальных порта, обозначенные CR1 и CR2, предназначенные для управления вращением сервоприводов CR в любом из двух направлений.

Сервопривод продолжительного вращения может пригодиться, если под электродвигатель постоянного тока отводится небольшое пространство или эксплуатационная нагрузка на него невелика. Подсоединяя сервоприводы к соответствующему порту на контроллере PRIZM, следите за тем, чтобы красному проводу сервопривода соответствовал знак "+", а чёрному проводу знак "-".

***myname*.setServoSpeed(сервопривод#, скорость);**

Эта функция задаёт скорость стандартного сервопривода, пока его вал поворачивается в указанное командой положение. Параметр *сервопривод#* выбирает канал управления сервоприводом. Параметр *скорость* задаёт угловую скорость. Вызвать эту функцию придётся только один раз — в начале программы. Но если потребуется изменить скорости, вызвать её в программном коде можно в любое время. Скорость можно менять в пределах от 0 до 100, что соответствует скоростям от 0 до 100 %. Если функцию не вызывать, скорость сервопривода по умолчанию всегда равна 100.

Примеры:

```
prizm.setServoSpeed(1, 25);                    // задать 1-му каналу сервопривода скорость 25 %
```

***myname*.setServoSpeeds(скорость1, скорость2, скорость3, скорость4, скорость5, скорость6);**

Эта функция позволяет настроить все шесть каналов сервопривода одним оператором функции. Параметры *скорость(#)* можно менять в пределах от 0 до 100. Все скорости будут разными. Учтите, что при использовании функций setServoSpeeds необходимо задать значения всем шести портам, даже если вы не используете все шесть.

Примеры:

```
prizm.setServoSpeeds(25, 25, 25, 25, 25, 25);     // задать всем шести каналам сервопривода скорость 25 %
```

***myname*.setServoPosition(сервопривод#, положение);**

Эта функция вращает сервопривод в заданное конечное положение. Параметр *сервопривод#* относится к сервоприводу, которому предстоит изменить положение. Параметр *положение* указывает, к какому конечному положению следует двигаться. Положение может быть каким угодно в пределах от 0 до 180. Параметр *положение* отображает конечное угловое положение сервопривода в градусах. Старайтесь не допускать вращения сервопривода дальше установленного механического ограничителя. Этот ограничитель может в незначительной степени отличаться у разных сервоприводов. При вращении сервопривода к крайним точкам, соответствующим значениям 0 или 180, прислушивайтесь к его звуку: не должно быть жужжания, которое означает, что сервопривод пытается преодолеть механический ограничитель. Это может повредить сервопривод и разрядить аккумуляторную батарею.

Примеры:

```
prizm.setServoPosition(1, 180);                    // повернуть сервопривод № 1 в положение 180 градусов
```

***myname*.setServoPositions(положение1, положение2, положение3, положение4, положение5, положение6);**

Эта функция обеспечивает одновременное вращение всех шести каналов сервоприводов в конечные положения. Параметр *положение(#)* задаёт конечное положение каждого канала в порядке от 1 до 6. Конечное положение может быть любым значением от 0 до 180, выраженным в градусах. Все заданные конечные положения могут быть разными. Учтите, что при использовании функций setServoPositions необходимо задать значения всем шести портам, даже если вы не используете все шесть.

Примеры:

```
prizm.setServoPositions(90,90,90,90,90,90);      // повернуть все шесть сервоприводов в положение 90 градусов
```

***myname*.readServoPosition(сервопривод#);**

Функция считывает и возвращает текущее положение канала сервопривода. Параметр сервопривод# может относиться к любому каналу с 1-го по 6-й. Возвращается значение в виде целого числа от 0 до 180.

Примеры:

```
int x;                                  // задать целочисленную переменную с именем x
x = prizm.readServoPosition(1);         // считать текущее угловое положение управляющего
                                        // сервоприводом канала 1 и сохранить его в виде переменной x
```

или

```
Serial.print(prizm.readServoPosition(2));   // считать положение управляющего сервоприводом канала 2 и
                                            // напечатать его на экране прибора контроля последовательной
                                            // передачи данных Arduino
```

***myname*.setCRServoState(сервоприводCR#, состояние);**

Функция управляет включением-выключением сервопривода продолжительного вращения (CR). Параметр *сервоприводCR#* относится к каналу электродвигателя постоянного вращения, которым предстоит управлять. Параметр *состояние* задаёт состояние включения-выключения и направление вращения. Параметр *состояние* может быть 100 для вращения по часовой стрелке, -100 для вращение против часовой стрелки и 0 для выключения или остановки.

Примеры:

```
prizm.setCRServoState(1, 100);      // включить CR1 и вращать в продолжительном режиме по часовой
                                    // стрелке

prizm.setCRServoState(1, 0);        // выключить CR1 и остановить

prizm.setCRServoState(1, -100);     // включить CR1 и вращать в продолжительном режиме
                                    // против часовой стрелки
```

Таблица библиотечных функций на языке Arduino для контроллера серии TETRIX® PRIZM™

Описание	Функция	Пример кодирования
Начать работу контроллера Prizm Вызывается в цикле настройки() кода на языке Arduino. Начинает работу контроллера PRIZM	**PrizmBegin();** Тип данных: Отсутствуют	**PrizmBegin**(); *Сбросить параметры и инициализировать контроллер PRIZM.*
Закончить работу контроллера Prizm После вызова сразу же завершает программу и возвращает контроллер PRIZM в исходное состояние.	**PrizmEnd();** Тип данных: Отсутствуют	**PrizmEnd**(); *Завершить программу контроллера PRIZM и вернуть контроллер в исходное состояние.*
Настроить красный светодиод Даёт красному светодиодному указателю контроллера PRIZM команду на включение или выключение.	**setRedLED**(состояние); Тип данных: *состояние* = целое число Диапазон данных: *состояние* = 1 или 0 или *состояние* = HIGH или LOW	**setRedLED**(HIGH); *или* **setRedLED**(1); *Включить красный светодиод.* **setRedLED**(LOW); *или* **setRedLED**(0); *Выключить красный светодиод.*
Настроить зелёный светодиод Даёт красному светодиодному указателю контроллера PRIZM команду на включение или выключение.	**setGreenLED**(состояние); Тип данных: *состояние* = целое число Диапазон данных: *состояние* = 1 или 0 или *состояние* = HIGH или LOW	**setGreenLED**(HIGH); *или* **setGreenLED**(1); *Включить зелёный светодиод.* **setGreenLED**(LOW); *или* **setGreenLED**(0); *Выключить зелёный светодиод.*
Задать мощность электродвигателя постоянного тока Задаёт уровень мощности и направление вращения электродвигателя постоянного тока TETRIX, подсоединённого к соответствующим портам на контроллере PRIZM. Размах уровня мощности составляет 0...100. Направление вращения задаётся знаком (+/-) перед уровнем мощности. Уровень мощности 0 = остановить в режиме выбега. Уровень мощности 125 = остановить в режиме торможения.	**setMotorPower**(электродвигатель#, мощность); Тип данных: *электродвигатель#* = целое число *мощность2* = целое число Диапазон данных: *электродвигатель#* = 1 или 2 *мощность* = -100...100 или *мощность* = 125 (режим торможения)	**setMotorPower**(1, 50); *Вращать электродвигатель № 1 по ЧС с мощностью 50 %* **setMotorPower**(2, -50%); *Вращать электродвигатель № 2 против ЧС с мощностью 50 %* **setMotorPower**(1, 0); *Выключить электродвигатель № 1 в режиме выбега.* **setMotorPower**(2, 125); *Выключить электродвигатель № 2 в режиме торможения.*
Задать мощность электродвигателей постоянного тока Одновременно устанавливается уровень мощности и направление вращения **обоих** электродвигателей постоянного тока TETRIX, подсоединённых к соответствующим портам на контроллере PRIZM. Параметры обоих каналов управления электродвигателями PRIZM № 1 и № 2 задаются единственным оператором. Размах уровня мощности составляет 0...100. Направление вращения задаётся знаком (+/-) перед уровнем мощности. Уровень мощности 0 = остановить в режиме выбега. Уровень мощности 125 = остановить в режиме торможения.	**setMotorPowers**(мощность 1-го электродвигателя, мощность мотора 2-го электродвигателя); Тип данных: мощность 1-го электродвигателя = целое число мощность 2-го электродвигателя = целое число Диапазон данных: *мощность1* = -100...100 *мощность2* = -100...100 или мощность 1-го электродвигателя = 125 (режим торможения) мощность 2-го электродвигателя = 125 (режим торможения)	**setMotorPowers**(50, 50); *Вращать электродвигатели №№ 1 и 2 по ЧС с мощностью 50 %.* **setMotorPowers**(-50, 50); *Вращать электродвигатель № 1 против ЧС, а электродвигатель № 2 по ЧС с мощностью 50 %.* **setMotorPowers**(0, 0); *Выключить электродвигатели №№ 1 и 2 в режиме выбега.* **setMotorPowers**(125, 125); *Выключить электродвигатели №№ 1 и 2 в режиме торможения.*

Описание	Функция	Пример кодирования
Задать угловую скорость электродвигателя постоянного тока При помощи ПИД-регулирования электродвигателю постоянного тока TETRIX, снабжённому энкодером для электродвигателей постоянного тока, задаётся постоянная скорость. Диапазон значений параметра *скорость* составляет 0...720 градусов в секунду (°/с). Знак (+/-) перед параметром *скорость* задаёт направление вращения.	**setMotorSpeed**(электродвигатель#, скорость); Тип данных: *электродвигатель#* = целое число *скорость* = целое число Диапазон данных: *электродвигатель#* = 1 или 2 *скорость* = -720...720	**setMotorSpeed**(1, 360); *Вращать электродвигатель № 1 по часовой стрелке с постоянной угловой скоростью 360 °/с.* **setMotorSpeed**(1, -360); *Вращать электродвигатель № 1 против часовой стрелки с постоянной угловой скоростью 360 °/с.*
Задать угловые скорости электродвигателей постоянного тока При помощи ПИД-регулирования скорости одновременно задаются постоянные скорости **обоим** каналам электродвигателей постоянного тока TETRIX, оснащённых энкодерами для электродвигателей TETRIX. Параметры обоих каналов управления электродвигателями PRIZM № 1 и № 2 задаются единственным оператором. Диапазон значений параметра *скорость* составляет 0...720 градусов в секунду (°/с). Знак (+/-) перед параметром *скорость* задаёт направление вращения.	**setMotorSpeeds**(скорость1, скорость2); Тип данных: *скорость*1 = целое *скорость*2 = целое Диапазон данных: *скорость1* = -720...720 *скорость2* = -720...720	**setMotorSpeed**(360, 360); *Вращать электродвигатель № 1 и электродвигатель № 2 по часовой стрелке с постоянной скоростью 360 °/с.* **setMotorSpeed**(360, -360); *Вращать электродвигатель № 1 по часовой стрелке, а электродвигатель № 2 против часовой стрелки с постоянной скоростью 360 °/с.* **setMotorSpeed**(360, -180); *Вращать электродвигатель № 1 по часовой стрелке, а электродвигатель № 2 против часовой стрелки с постоянной скоростью 180 °/с.*
Задать конечное положение электродвигателя постоянного тока При помощи ПИД-регулирования скорости и положения задаётся постоянная скорость и определяемое показаниями счётчика энкодера конечное положения удержания электродвигателя постоянного тока TETRIX, снабжённого энкодером. Диапазон значений параметра *скорость* составляет 0...720 градусов в секунду (°/с). Конечное положение, определяемое показаниями счётчика энкодера, представляет собой длинное целое число от -2 147 483 648 до 2 147 483 647. Каждый отсчёт энкодера = разрешение в 1/4 градуса.	**setMotorTarget**(электродвигатель#, скорость, заданное конечное положение); Тип данных: *электродвигатель#* = целое число *скорость* = целое число *заданное конечное положение* = длинное целое Диапазон данных: *электродвигатель#* = 1 или 2 *скорость* = 0...720 *заданное конечное положение* = -2147483648... 2147483647	**setMotorTarget**(1, 360, 1440); *Вращать электродвигатель № 1 с постоянной угловой скоростью 360 °/с до тех пор, пока показания энкодера № 1 не станут равны 1440. По достижении заданных показаний счётчика энкодера удержать в занимаемом положении, как при использовании сервопривода.* **setMotorTarget**(2, 180, -1440); *Вращать электродвигатель № 2 с постоянной угловой скоростью 180 °/с до тех пор, пока показания энкодера № 2 не станут равны -1440 (1 оборот). По достижении заданных показаний счётчика энкодера удержать в занимаемом положении, как при использовании сервопривода.*

Описание	Функция	Пример кодирования
Задать конечные положения электродвигателей постоянного тока При помощи ПИД-регулирования скорости и положения одновременно задаются постоянная скорость и конечные показатели счётчика энкодера, соответствующие положениям удержания для каналов **обоих** электродвигателей постоянного тока TETRIX, каждый из которых снабжён энкодером. Параметры обоих каналов управления электродвигателями PRIZM № 1 и № 2 задаются единственным оператором. Диапазон значений параметра угловой скорости составляет 0...720 градусов в секунду (°/с). Конечное положение, определяемое показаниями счётчика энкодера, представляет собой длинное целое число от -2 147 483 648 до 2 147 483 647. Каждый отсчёт энкодера = разрешение в 1/4 градуса.	**setMotorTargets**(скорость1, заданное конечное положение1, скорость2, заданное конечное положение2); Тип данных: *скорость*1 = целое *заданное конечное положение1* = длинное целое *скорость*2 = целое *заданное конечное положение2* = длинное целое Диапазон данных: *скорость1* = 0...720 *заданное конечное положение1* = -2147483648... 2147483647 *скорость2* = 0...720 *заданное конечное положение2* = -2147483648... 2147483647	**setMotorTargets**(360, 1440, 360, 1440); *Вращать электродвигатели № 1 и № 2 с постоянной угловой скоростью 360 °/с до тех пор, пока показания счётчика энкодера каждого электродвигателя не станут равны 1440. По достижении электродвигателем заданных показаний счётчика энкодера удержать в занимаемом положении, как при использовании сервопривода.* **setMotorTargets**(360, 1440, 180, 2880); *Вращать электродвигатель № 1 с постоянной угловой скоростью 360 °/с до тех пор, пока показания энкодера № 1 не станут равны 1440. Вращать электродвигатель № 2 с постоянной угловой скоростью 180 °/с до тех пор, пока показания энкодера № 2 не станут равны 2 880. Каждый электродвигатель по достижении заданного конечного положения, определяемого энкодером, задержится в занятом положении, как это делает сервопривод.* ***Примечание:*** *Один отсчёт энкодера равен разрешению в 1/4 градуса. Например, 1 оборот вала электродвигателя равен 1440 отсчётам энкодера (1440/4 = 360).*
Задать градус электродвигателя При помощи ПИД-регулирования скорости и положения задаются постоянная скорость и градус, соответствующие конечному положению удержания электродвигателя постоянного тока TETRIX, снабжённого энкодером. Диапазон значений параметра угловой скорости составляет 0...720 градусов в секунду (°/с). Градус конечного положения, определяемый энкодером, представляет собой длинное целое от -536 870 912 до 536 870 911 с разрешением в 1 градус.	**setMotorDegree**(электродвигатель#, скорость, градусы); Тип данных: *электродвигатель#* = целое число *скорость* = целое число *градусы* = длинное целое Диапазон данных: *электродвигатель#* = 1 или 2 *скорость* = 0...720 *градусы* = -536870912...536870911	**setMotorDegree**(1, 180, 360); *Вращать электродвигатель № 1 с постоянной угловой скоростью 180 °/с до тех пор, пока показания счётчика энкодера № 1 не станут равны 360. По достижении заданного угла в градусах, определяемого энкодером, удержать в занимаемом положении, как при использовании сервопривода.* **setMotorDegree**(2, 90, 180); *Вращать электродвигатель № 2 с постоянной угловой скоростью 90 °/с до тех пор, пока показания счётчика энкодера № 2 не станут равны 180. По достижении заданного угла в градусах, определяемого энкодером, удержать в занимаемом положении, как при использовании сервопривода.*

Описание	Функция	Пример кодирования
Задать угловую скорость электродвигателя в градусах При помощи ПИД-регулирования скорости и положения задаются постоянные скорости и измеряемые в градусах конечные положения удержания электродвигателя постоянного тока TETRIX, снабжённого энкодером. Параметры обоих каналов управления электродвигателями PRIZM № 1 и № 2 задаются единственным оператором. Диапазон значений параметра угловой скорости составляет 0...720 градусов в секунду (°/с). Заданное конечное положение в градусах, определяемое энкодером, представляет собой длинное целое от -536 870 912 до 536 870 911 с разрешением в 1 градус.	**setMotorDegrees**(скорость1, градусы1, скорость2, градусы2); Тип данных: *скорость*1 = целое *градусы1* = длинное целое *скорость*2 = целое *градусы2* = длинное целое Диапазон данных: *скорость1* = 0...720 *градусы1* = -536870912...536870911 *скорость2* = 0...720 *градусы2* = -536870912...536870911	**setMotorDegrees**(180, 360, 180, 360); *Вращать электродвигатели № 1 и № 2 с постоянной угловой скоростью 180 °/с до тех пор, пока показания энкодера каждого электродвигателя не станут равны 360. По достижении электродвигателем заданного угла в градусах, удержать в занимаемом положении, как при использовании сервопривода.* **setMotorDegrees**(360, 720, 90, 360); *Вращать электродвигатель № 1 с постоянной угловой скоростью 360 °/с до тех пор, пока показания энкодера № 1 не станут равны 720. Вращать электродвигатель № 2 с постоянной угловой скоростью 90 °/с до тех пор, пока показания энкодера № 2 не станут равны 360. Каждый электродвигатель по достижении заданного конечного положения, определяемого энкодером, задержится в занятом положении, как это делает сервопривод.*
Задать обратное направление электродвигателю Делает обратной схему правого/левого вращения в канале управления электродвигателем постоянного тока. Функция предназначена для согласования прямого и обратного вращения электродвигателей на противоположных бортах шасси робототехнической модели с бортовым поворотом. Замена заданных действий обратными в одном канале управления электродвигателями может потребовать более интуитивного кодирования противолежащих электродвигателей постоянного тока, работающих в паре. Параметр *обратное действие* со значением 1 = обратное действие. Параметр *обратное действие* со значением 0 = отсутствие обратного действия. По умолчанию установлено отсутствие обратного действия.	**setMotorInvert**(электродвигатель#, обратное действие); Тип данных: *электродвигатель#* = целое число *обратное действие* = целое число Диапазон данных: *электродвигатель#* = 1 или 2 *обратное действие* = 0 или 1	**setMotorInvert**(1, 1); *Заменить схему направления вращения электродвигателя № 1 на обратную.* **setMotorInvert**(2, 1); *Заменить схему направления вращения электродвигателя № 2 на обратную.* **setMotorInvert**(1, 0); *Не менять схему направления вращения электродвигателя № 1 на обратную.* **setMotorInvert**(2, 0); *Не менять схему направления вращения электродвигателя № 2 на обратную.* ***Примечание:*** *По умолчанию контроллер PRIZM настроен на отсутствие обратного действия после включения питания или сброса параметров.*
Считать состояние занятости электродвигателя Проверить состояние электродвигателя постоянного тока, который работает в режиме ПИД-регулирования положения, можно по флажку занятости. В состоянии занятости электродвигателя будет возвращено значение "1", если он движется к заданному конечному положению (в градусах или в отсчётах энкодера). Когда он достигнет заданного конечного положения и задержится в нём, возвращение значения "0" сообщит, что он занят.	**readMotorBusy**(электродвигатель#); Тип данных: *электродвигатель#* = целое число Диапазон данных: *электродвигатель#* = 1 или 2 Тип возвращаемых данных: *значение* = целое число	**readMotorBusy**(1); *Вернуть состояние занятости электродвигателя № 1.* **readMotorBusy**(2); *Вернуть состояние занятости электродвигателя № 2.*

Описание	Функция	Пример кодирования
Считать ток, потребляемый электродвигателем постоянного тока Считывает ток, потребляемый каждым электродвигателем постоянного тока TETRIX, подсоединённым к портам Motor 1 и Motor 2 контроллера PRIZM. Возвращаемое целочисленное значение представляет собой ток нагрузки электродвигателя в миллиамперах.	**readMotorCurrent**(электродвигатель#); Тип данных: *электродвигатель#* = целое число Диапазон данных: *электродвигатель#* = 1 или 2 Тип возвращаемых данных: *значение* = целое число	**readMotorCurrent**(1); *Считать ток нагрузки электродвигателя, подсоединённого к каналу Motor 1 контроллера PRIZM.* **readMotorCurrent**(2); *Считать ток нагрузки электродвигателя, подсоединённого к каналу Motor 2 контроллера PRIZM.* *Примеры: 1500 = 1,5 A*
Считать показания счётчика энкодера Считывается число отсчётов энкодера. Контроллер PRIZM использует число импульсов, выданных энкодером, для управления по ПИД-алгоритму электродвигателем постоянного тока TETRIX, подсоединённого к соответствующим портам. Контроллер PRIZM, отсчитывая импульсы в заданном отрезке времени, точно регулирует скорость и положение. Каждая 1/4 градуса равна одному импульсу или отсчёту, либо 1 угловой градус равен 4 отсчётам энкодера. Считав текущее показание счётчика, можно определить положение вала электродвигателя. Общие показания счётчика могут колебаться от -2 147 483 648 до 2 147 483 647. Вращение по часовой стрелке увеличивает показания счётчика, а вращение против часовой стрелки уменьшает показания счётчика. Значения энкодера обнуляются в случае включения питания и сброса параметров.	**readEncoderCount**(энк#); Тип данных: *энк#* = целое число Диапазон данных: *энк#* = 1 или 2 Тип возвращаемых данных: *значение* = длинное целое	**readEncoderCount**(1); *Считать текущее показание счётчика энкодера 1 (порт ENC 1).* **readEncoderCount**(2); *Считать текущее показание счётчика энкодера 2 (порт ENC 2).*
Считать показания энкодера в градусах Считывается число отсчитанных энкодером градусов. Контроллер PRIZM использует число импульсов, выданных энкодером, для управления по ПИД-алгоритму электродвигателем постоянного тока TETRIX, подсоединённого к соответствующим портам. Контроллер PRIZM, отсчитывая импульсы в заданном отрезке времени, точно регулирует скорость и положение. Эта функция аналогична функции счётчика энкодера, но вместо возврата приблизительных показаний счётчика энкодера она возвращает положение вала электродвигателя в градусах. Общие показания счётчика могут колебаться от -536 870 912 до 536 870 911. Вращение по часовой стрелке увеличивает показания счётчика, а вращение против часовой стрелки уменьшает показания счётчика. Значения энкодера обнуляются в случае включения питания и сброса параметров.	**readEncoderDegrees**(энк#); Тип данных: *энк#* = целое число Диапазон данных: *энк#* = 1 или 2 Тип возвращаемых данных: *значение* = длинное целое	**readEncoderDegrees**(1); *Считать текущее показание счётчика энкодера 1 в градусах (порт ENC 1).* **readEncoderDegrees**(2); *Считать текущее показание счётчика энкодера 2 в градусах (порт ENC 2).*

Описание	Функция	Пример кодирования
Обнулить счётчик каждого энкодера Вызов этой функции приведёт к обнулению показаний счётчика энкодера.	**resetEncoder**(энк#); Тип данных: *энк#* = целое число Диапазон данных: enc# = 1 или 2	**resetEncoder**(1); *Обнулить показания счётчика энкодера 1.* **resetEncoder**(2); *Обнулить показания счётчика энкодера 2.*
Обнулить счётчики обоих энкодеров (1 и 2) Вызов этой функции приведёт к обнулению показаний счётчиков энкодера 1 и энкодера 2.	**resetEncoders**(); Тип данных: Отсутствуют	**resetEncoders**(); *Обнулить показания счётчика энкодера 1 и энкодера 2.*
Считать выходной сигнал искателя линии Считывает цифровой выходной сигнал датчика линии, подсоединённого к порту для датчиков контроллера PRIZM. Считываемое значение равно 0, если уловлено отражение света (обнаружена поверхность светлого цвета), и 1, если отражение света не уловлено (обнаружена поверхностью тёмного цвета, например, линия).	**readLineSensor**(порт#); Тип данных: *порт#* = целое число Диапазон данных: *порт#* (см. примечание в смежной колонке.) Тип возвращаемых данных: *значение* = целое число (0 или 1)	**readLineSensor**(2); *Считать цифровое значение датчика линии на порте D2 для цифровых датчиков.* ***Примечание:*** *Датчик линии можно подсоединить к любому цифровому порту D2–D5 или аналоговому порту A1–A3, настроенному на приём цифровых сигналов. Подробнее о портах для датчиков и схемах разводки контактов см. технический раздел, посвящённый портам для датчиков.*
Считать показания ультразвукового датчика в сантиметрах Считывает расстояние в сантиметрах до объекта перед ультразвуковым датчиком. Частота датчика модулируется на уровне 42 кГц, а диапазон измерения составляет от 3 до 400 сантиметров. Считываемое значение представляет собой целое число.	**readSonicSensorCM**(порт#); Тип данных: *порт#* = целое число Диапазон данных: *порт#* (см. примечание в смежной колонке.) Тип возвращаемых данных: *значение* = целое число (3...400) Возможна небольшая вариация в максимальных и минимальных значениях.	**readSonicSensorCM**(3); *Считать расстояние в сантиметрах до объекта перед ультразвуковым датчиком, подсоединённым к порту D3 для цифровых датчиков.* ***Примечание:*** *Ультразвуковой датчик можно подсоединить к любому цифровому порту D2–D5 или аналоговым портам A1–A3, настроенным на приём цифровых сигналов. Подробнее о портах для датчиков и схемах разводки контактов см. технический раздел, посвящённый портам для датчиков.*
Считать показания ультразвукового датчика в дюймах Считывает расстояние в дюймах до объекта перед ультразвуковым датчиком. Частота датчика модулируется на уровне 42 кГц, а диапазон измерения составляет от 2 до 150 дюймов. Считываемое значение представляет собой целое число.	**readSonicSensorIN**(порт#); Тип данных: *порт#* = целое число Диапазон данных: *порт#* (см. примечание в смежной колонке.) Тип возвращаемых данных: *значение* = целое число (2...150) Возможна небольшая вариация в максимальных и минимальных значениях.	**readSonicSensorIN**(4); *Считать расстояние в дюймах до объекта перед ультразвуковым датчиком, подсоединённым к порту D4 для цифровых датчиков.* ***Примечание:*** *Ультразвуковой датчик можно подсоединить к любому цифровому порту D2–D5 или аналоговым портам A1–A3, настроенным на приём цифровых сигналов.*
Считать напряжение аккумуляторной батареи Считывает напряжение аккумуляторной батареи TETRIX, питающей контроллер PRIZM. Считываемое значение представляет собой целое число.	**readBatteryVoltage**(); Тип данных: Отсутствуют Тип возвращаемых данных: *значение* = целое число	**readBatteryVoltage**(); *Считать напряжение аккумуляторной батареи TETRIX, питающей контроллер PRIZM.* *Примеры: Значение 918 равно 9,18 В.*

Описание	Функция	Пример кодирования
Считать состояние пусковой кнопки Считывает состояние зелёной пусковой кнопки контроллера PRIZM. Возвращение значения "1" указывает на то, что кнопка нажата. Возвращение значения "0" указывает на то, что кнопка не нажата.	**readStartButton**(); Тип данных: Отсутствуют Тип возвращаемых данных: *значение* = целое число (0 или 1)	**readStartButton**(); *Считать пусковую кнопку. Значение "1" означает, что кнопка нажата. Значение "0" означает, что кнопка не нажата.*
Считать положение сервопривода Считывает соответствующее самой последней команде положение вала сервопривода, подсоединённого к портам 1-6 для сервоприводов на контроллере PRIZM. Возвращаемое значение составит 0...180.	**readServoPosition**(сервопривод#); Тип данных: *сервопривод#* = целое число Диапазон данных: *сервопривод#* = 1...6 Тип возвращаемых данных: *значение* = целое число (0...180)	**readServoPosition**(1); *Считать соответствующее самой последней команде положение вала сервопривода № 1.* **readServoPosition**(2); *Считать соответствующее самой последней команде положение вала сервопривода № 2.*
Задать состояние сервопривода продолжительного вращения (CR) Задаётся включение-выключение и направление вращения сервопривода продолжительного вращения, подсоединённого к портам CR1 и CR2 контроллера PRIZM. Параметр *состояния* -100 приравнивается к вращению против часовой стрелки. Параметр состояния 100 приравнивается к вращению по часовой стрелке. Параметр состояния 0 приравнивается к выключению или остановке. Параметр *сервоприводCR#* может быть командой 1 или 2 либо на порт CR1, либо на порт CR2.	**setCRServoState**(сервоприводCR#, состояние); Тип данных: *сервоприводCR#* = целое число *состояние* = целое число Диапазон данных: *сервоприводCR#* = 1 или 2 *состояние* = -100, 0, 100	**setCRServoState**(1, 100); *Вращать сервопривод CR1 в продолжительном режиме по часовой стрелке.* **setCRServoState**(1, 0); *Остановить сервопривод CR1.* **setCRServoState**(1, -100); *Вращать сервопривод продолжительного вращения CR1 в продолжительном режиме против часовой стрелки.*

Памятка по библиотечным функциям на языке Arduino для контроллера серии TETRIX® PRIZM™

Ниже — справочный указатель всех операторов функций в библиотеке для контроллера к робототехническим моделям серии TETRIX PRIZM, используемых в среде разработки на языке Arduino.

Подсказка: Используя эти функции в нашем коде, обязательно ставьте перед каждой функцией имя объекта PRIZM, которое мы выбираем, используя оператор *myname* для PRIZM.

PrizmBegin();

PrizmEnd();

setRedLED(HIGH/LOW);

setGreenLED(HIGH/LOW);

setMotorPower(электродвигатель#, мощность);

setMotorPowers(мощность 1-го электродвигателя, мощность мотора 2-го электродвигателя);

setMotorSpeed(электродвигатель#, скорость);

setMotorSpeeds(скорость1, скорость2);

setMotorTarget(электродвигатель#, скорость, заданное конечное положение);

setMotorTargets(скорость1, заданное конечное положение1, скорость2, заданное конечное положение2);

setMotorDegree(электродвигатель#, скорость, градусы);

setMotorDegrees(скорость1, градусы1, скорость2, градусы2);

setMotorInvert(электродвигатель#, обратное действие);

readMotorBusy(электродвигатель#);

readMotorCurrent(электродвигатель#);

readEncoderCount(энк#);

readEncoderDegrees(энк#);

resetEncoder(энк#);

resetEncoders();

readLineSensor(порт#);

readSonicSensorCM(порт#);

readSonicSensorIN(порт#);

readBatteryVoltage();

readStartButton();

setServoSpeed(сервопривод#, скорость);

setServoSpeeds(скорость1, скорость2, скорость3, скорость4, скорость5, скорость6);

setServoPosition(сервопривод#, положение);

setServoPositions(положение1, положение2, положение3, положение4, положение5, положение6);

readServoPosition(сервопривод#);

setCRServoState(сервоприводCR#, состояние);

Библиотека с образцами программ для контроллера TETRIX® PRIZM™

Пример 1: GettingStarted_Act1_Blink_RedLED

```
/*  пример программы для контроллера PRIZM
 *   Заставить красный светодиод контроллера PRIZM мигать с 1-секундным интервалом.
 *   автор PWU от 08/05/2016
*/

  #include <PRIZM.h>              // добавить библиотеку контроллера PRIZM

  PRIZM prizm;                    // создать экземпляр объекта PRIZM "prizm",
                                  // чтобы можно было использовать его функции

void setup() {

   prizm.PrizmBegin();            // начать работу контроллера PRIZM

}

void loop() {                     // повторять этот код циклически

  prizm.setRedLED(HIGH);          // включить красный светодиод
  delay(1000);                    // ждать здесь 1000 мс (1 секунду)
  prizm.setRedLED(LOW);           // выключить красный светодиод
  delay(1000);                    // ждать здесь 1000 мс (1 секунду)
}
```

Пример 2: GettingStarted_Act2_Move_DCMotor

```
/*  пример программы для контроллера PRIZM
 *  Вращать канал 1 электродвигателя постоянного тока 5 секунд, после чего остановить выбегом.
 *  После остановки ждать 2 секунды, после чего вращать в противоположном направлении.
 *  Повторять, пока не будет нажата красная кнопка сброса параметров.
 *  автор PWU от 08/05/2016
*/

  #include <PRIZM.h>              // добавить в скетч библиотеку контроллера PRIZM
  PRIZM prizm;                    // создать экземпляр объекта PRIZM "prizm",
                                  // чтобы можно было использовать его функции

void setup() {

  prizm.PrizmBegin();             // начать работу контроллера PRIZM
}

void loop() {                     // повторять циклически

  prizm.setMotorPower(1,25);      // вращать электродвигатель № 1 по ЧС с мощностью 25 %
  delay(5000);                    // ждать 5 секунд, не останавливая электродвигатель
  prizm.setMotorPower(1,0);       // остановить электродвигатель (остановить выбегом)
  delay(2000);                    // после остановки ждать здесь 2 секунды
  prizm.setMotorPower(1,-25);     // вращать электродвигатель № 1 против ЧС с мощностью 25 %
  delay(5000);                    // ждать 5 секунд, не останавливая электродвигатель
  prizm.setMotorPower(1,0);       // остановить электродвигатель (остановить выбегом)
  delay(2000);                    // после остановки ждать здесь 2 секунды, после чего повторить
}
```

Пример 3: GettingStarted_Act3_Move_Servo

```
/*  пример программы для контроллера PRIZM
 *   Программа задаёт сервоприводу № 1 скорость 25 %.
 *   Сервопривод № 1 после этого вращается вперёд-назад в пределах от 0 до 180 градусов.
 *   автор PWU от 08/05/2016
*/

#include <PRIZM.h>                  // добавить в скетч библиотеку контроллера PRIZM
PRIZM prizm;                        // создать экземпляр объекта PRIZM "prizm",
                                    // чтобы можно было использовать его функции

void setup() {

  prizm.PrizmBegin();               // начать работу контроллера PRIZM
  prizm.setServoSpeed(1,25);        // задать сервоприводу № 1 скорость 25 %

}

void loop() {                       // повторять циклически

  prizm.setServoPosition(1 180);    // повернуть сервопривод № 1 в положение 180 градусов
  delay(3000);                      // ждать 3 секунды, чтобы дать сервоприводу 1 попытку
                                    // встать в положение 180
  prizm.setServoPosition(1,0);      // повернуть сервопривод № 1 в положение 0 градусов
  delay(3000);                      // ждать 3 секунды, чтобы дать сервоприводу 1 попытку
                                    // встать в положение 0
}
```

Пример 4: GettingStarted_Act4_Intro_LineFinder

```
/*  пример программы для контроллера PRIZM
 *  Программа считает цифровой сигнал датчика линии, подсоединённого к цифровому порту D3.
 *  Если датчик обращён чувствительным элементом к светоотражающей поверхности и принимает
отражённый инфракрасный луч, тогда красный светодиод на контроллере PRIZM
 *  должен гореть. Если датчик обращён чувствительным элементом к тёмной поверхности или находится
слишком далеко от светоотражающей поверхности,
 * тогда красный светодиод не должен гореть.
*/

  #include <PRIZM.h>                // добавить библиотеку контроллера PRIZM
  PRIZM prizm;                      // создать экземпляр объекта PRIZM "prizm",
                                    // чтобы можно было использовать его функции

void setup() {                      // этот код используется один раз

  prizm.PrizmBegin();               // начать работу контроллера PRIZM

}

void loop() {                       // этот код повторяется циклически
  if(prizm.readLineSensor(3) == HIGH) {prizm.setRedLED(LOW);}       // светодиод не горит
  if(prizm.readLineSensor(3) == LOW) {prizm.setRedLED(HIGH);}       // светодиод горит
delay(50);                          // замедлить цикл
}
```

Пример 5: GettingStarted_Act5_Intro_UltraSonic

```
/*   пример программы для контроллера PRIZM
 *   Этот пример программы будет считывать цифровой сигнал
 *   ультразвукового датчика, подсоединённого к порту D3 для цифровых датчиков.
 *   Расстояние в сантиметрах до объекта, установленного перед
 *   датчиком, будет отправлено на экран прибора контроля последовательной передачи данных.
 */

  #include <PRIZM.h>                // добавить библиотеку контроллера PRIZM
  PRIZM prizm;                      // создать экземпляр объекта PRIZM "prizm",
                                    // чтобы можно было использовать его функции

void setup() {                      // этот код используется один раз

  prizm.PrizmBegin();               // начать работу контроллера PRIZM

  Serial.begin(9600);               // настроить прибор контроля последовательной передачи данных
                                    // на скорость передачи 9600
}

void loop() {                       // этот код повторяется циклически

  Serial.print(prizm.readSonicSensorCM(3)); // напечатать расстояние в см в окне прибора контроля
                                    //  последовательной передачи данных

  Serial.println(" Centimeters");   // напечатать " Сантиметры"

  delay(200);                       // замедлить цикл. подождать 200 мс, чтобы не печатать в окне прибора
                                    //контроля последовательной передачи данных слишком быстро
}
```

Пример 6: TaskBot_Act7_Drive_Forward

```
/* пример программы для контроллера PRIZM
 *  Программа заставит робота-исполнителя PRIZM двигаться передним ходом, через 3 секунды остановиться,
* после чего завершит свою работу.
 *  автор PWU 08/05/2016
 */

  #include <PRIZM.h>                // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                      // создать экземпляр объекта PRIZM "prizm", чтобы можно было использовать
его функции

void setup() {

  prizm.PrizmBegin();               // начать работу контроллера PRIZM

  prizm.setMotorInvert(1,1);        // заменить вращение электродвигателя постоянного тока № 1 на обратное
                                    // чтобы согласовать направление вращения противолежащих приводных
электродвигателей
}

void loop() {

  prizm.setMotorPowers(50,50);      // вращать электродвигатели №№ 1 и 2 с мощностью 50 %
  delay(3000);                      // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.PrizmEnd();                 // завершить программу и вернуть контроллер PRIZM в исходное состояние
}
```

Пример 7: TaskBot_Act8_Drive_Circle

```
/*  пример программы для контроллера PRIZM
 *  Программа заставит робота-исполнителя PRIZM непрерывно двигаться по кругу.
 *  Чтобы остановить программу, нажмите красную кнопку сброса параметров.
 *  автор PWU 08/05/2016
 */

  #include <PRIZM.h>              // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                    // создать экземпляр объекта PRIZM "prizm", чтобы можно
                                  // было использовать его функции

void setup() {

  prizm.PrizmBegin();             // начать работу контроллера PRIZM
  prizm.setMotorInvert(1,1);      // изменить вращение электродвигателя постоянного тока № 1 на обратное,
                                  // чтобы согласовать направление
}

void loop() {

  prizm.setMotorPowers(50,25);    // вращать электродвигатель № 1 с мощностью 50 %,
                                  // а электродвигатель № 2 с мощностью 25 %,
                                  // в результате чего робот будет двигаться по кругу по часовой стрелке
}
```

Пример 8: TaskBot_Act9_Drive_Square_1

```
/*  пример программы для контроллера PRIZM
 *  Эта программа заставит робота-исполнителя PRIZM двигаться по траектории в форме квадрата.
 *  автор PWU 08/05/2016
*/
  #include <PRIZM.h>                   // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                         // создать экземпляр объекта PRIZM "prizm", чтобы можно было использовать
его функции

void setup() {
  prizm.PrizmBegin();                  // начать работу контроллера PRIZM
  prizm.setMotorInvert(1,1);           // включить обратное вращение электродвигателя постоянного тока № 1,
                                       // чтобы согласовать направление
}
void loop() {
  prizm.setMotorPowers(50,50);        // двигаться передним ходом с мощностью 50 %
  delay(3000);                         // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,-50);        // повернуть вправо
  delay(600);                          // ждать здесь 0,6 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,50);         // двигаться передним ходом с мощностью 50 %
  delay(3000);                         // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,-50);        // повернуть вправо
  delay(600);                          // ждать здесь 0,6 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,50);         // двигаться передним ходом с мощностью 50 %
  delay(3000);                         // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,-50);        // повернуть вправо
  delay(600);                          // ждать здесь 0,6 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.setMotorPowers(50,50);         // двигаться передним ходом с мощностью 50 %
  delay(3000);                         // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);       // остановить оба электродвигателя в режиме торможения
  delay(1000);                         // ждать здесь 1 секунду
  prizm.PrizmEnd();                    // завершить программу и вернуть контроллер PRIZM в исходное состояние
}
```

Пример 9: TaskBot_Act10_Drive_Square_2

```
/*  пример программы для контроллера PRIZM
 *   Эта программа с помощью функций переднего хода и поворота вправо заставит робота-исполнителя
* двигаться по траектории в форме квадрата.
 *  автор PWU 08/05/2016
 */
  #include <PRIZM.h>                  // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                        // создать экземпляр объекта PRIZM "prizm", чтобы можно
                                      // было использовать его функции

void setup() {

  prizm.PrizmBegin();                 // начать работу контроллера PRIZM
  prizm.setMotorInvert(1,1);          // изменить вращение электродвигателя постоянного тока № 1
                                      // на обратное, чтобы согласовать направление
}

void loop() {

for(int x=0; x<=3; x++) {             // сделать это четыре раза, с приращением x на +1
  forward();
  rightTurn();
 }
  prizm.PrizmEnd();
}

void forward() {                      // функция для движения передним ходом
  prizm.setMotorPowers(50,50);        // двигаться передним ходом с мощностью 50 %
  delay(3000);                        // ждать здесь 3 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);      // остановить оба электродвигателя в режиме торможения
  delay(1000);                        // ждать здесь 1 секунду

  }

void rightTurn() {                    // функция для поворота вправо
  prizm.setMotorPowers(50,-50);       // повернуть вправо
  delay(600);                         // ждать здесь 0,6 секунды, оставив электродвигатели вращаться
  prizm.setMotorPowers(125,125);      // остановить оба электродвигателя в режиме торможения
  delay(1000);                        // ждать здесь 1 секунду

  }
```

Пример 10: TaskBot_Act11_Drive_To_Line

```
/*  пример программы для контроллера PRIZM
 *  Эта программа заставит робота-исполнителя PRIZM двигаться передним ходом по поверхности
*   белого цвета, пока не будет обнаружена чёрная
 *  линия. Обнаружив линию, робот остановится.
 *  Подсоединить датчик линии к цифровому порту D3.
 *  автор PWU 08/05/2016
*/

  #include <PRIZM.h>                    // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                          // создать экземпляр объекта PRIZM "prizm", чтобы можно
                                        // было использовать его функции

void setup() {

  prizm.PrizmBegin();                   // начать работу контроллера PRIZM

  prizm.setMotorInvert(1,1);            // заменить вращение электродвигателя постоянного тока № 1 на
                                        // обратное для согласования направления вращения приводных
                                        // электродвигателей, расположенных друг напротив друга
}

void loop() {

  if(prizm.readLineSensor(3) == 0) {    // когда датчик уловит отражённый луч света
    prizm.setMotorPowers(35,35);        // вращать электродвигатели №№ 1 и 2 с мощностью 35 %,
                                        // если обнаружится световой луч

  }

  if(prizm.readLineSensor(3) == 1) {    // когда датчик обнаружит чёрную полосу
    prizm.setMotorPowers(125,125);      // остановить электродвигатели в режиме торможения,
                                        // если обнаружится чёрная линия

    while(1) {                          // бесконечный цикл – остаётся запертым в этом цикле,
                                        // пока не будет нажата кнопка сброса параметров
    prizm.setRedLED(HIGH);              // заставить красный светодиод контроллера PRIZM включаться
                                        // и выключаться, пока не будут сброшены параметры
    delay(500);
    prizm.setRedLED(LOW);
    delay(500);
    }
  }
}
```

Пример 11: TaskBot_Act12_Follow_A_Line

```
/* пример программы для контроллера PRIZM
 *  Эта программа заставляет робота-исполнителя PRIZM двигаться по линии. Используя датчик  *  линии, подсоединённый к порту D3 для датчиков, робототехническая модель будет двигаться по краю чёрной полосы на белой
*  поверхности.
*  Программа будет по очереди включать каждый электродвигатель, из-за чего робототехническая модель станет рыскать, двигаясь
 *  передним ходом, постоянно пересекая край линии.
 *   автор PWU 08/05/2016
 */

  #include <PRIZM.h>                    // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                          // создать экземпляр объекта PRIZM "prizm", чтобы можно
                                        // было использовать его функции

void setup() {

  prizm.PrizmBegin();                   // начать работу контроллера PRIZM

  prizm.setMotorInvert(1,1);            // заменить вращение электродвигателя постоянного тока № 1 на
                                        // обратное для согласования направления вращения приводных
                                        // электродвигателей, расположенных друг напротив друга
}

void loop() {

  // луч отразился, не обнаружено никакой линии
  if(prizm.readLineSensor(3) == 0) {prizm.setMotorPowers(125,30); prizm.setRedLED(LOW);}

  // луч не отразился, обнаружена линия
  if(prizm.readLineSensor(3) == 1) {prizm.setMotorPowers(30,125); prizm.setRedLED(HIGH);}

}
```

Пример 12: TaskBot_Act13_Drive_To_Wall

```
/*   пример программы для контроллера PRIZM
 *   Программа с помощью ультразвукового датчика, подсоединённого к
 *   порту D4 для датчиков, обнаруживает препятствие на пути следования робота
 *   на расстоянии 25 см. Обнаружив препятствие, робот должен остановиться и ждать,
 *   пока предмет, загораживающий ему путь, не будет убран.
 *   автор PWU 08/05/2016
 */

  #include <PRIZM.h>                    // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                          // создать экземпляр объекта PRIZM "prizm",
                                        // чтобы можно было использовать его функции

void setup() {

  prizm.PrizmBegin();                   // начать работу контроллера PRIZM

  prizm.setMotorInvert(1,1);            // заменить вращение электродвигателя постоянного тока № 1 на
                                        // обратное для согласования направления вращения приводных
                                        // электродвигателей, расположенных друг напротив друга
}

 void loop() {

  if(prizm.readSonicSensorCM(4) > 25)
  {
    prizm.setMotorPowers(50,50);        // если расстояние больше 25 см, сделать следующее
  }
 else
  {
    prizm.setMotorPowers(125 125);      // если расстояние меньше 25 см, сделать следующее
  }
}
```

Пример 13: TaskBot_Act14_Avoid_Obstacle

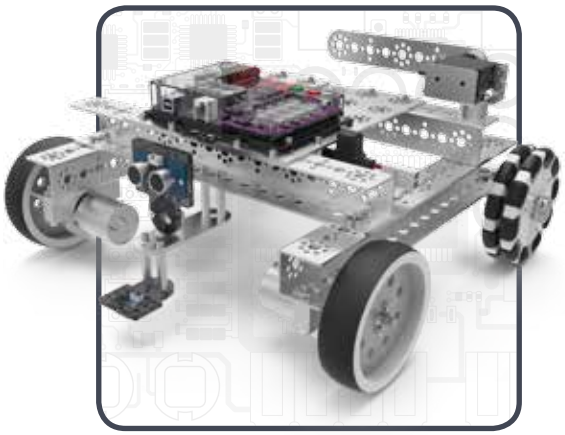
```
/*   пример программы для контроллера PRIZM
 *   Программа с помощью ультразвукового датчика, подсоединённого к порту D4 для датчиков,
*   обнаруживает объекты на пути следования робота.
 *   При обнаружении препятствия робот должен остановиться, отъехать назад, повернуть вправо и
*   продолжить движение.
 *   автор PWU 08/05/2016
 */

  #include <PRIZM.h>                    // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                          // создать экземпляр объекта PRIZM "prizm",
                                        // чтобы можно было использовать его функции

void setup() {

  prizm.PrizmBegin();                   // начать работу контроллера PRIZM

  prizm.setMotorInvert(1,1);            // заменить вращение электродвигателя постоянного тока № 1 на
                                        // обратное для согласования направления вращения приводных
                                        // электродвигателей, расположенных друг напротив друга
}

 void loop() {

  if(prizm.readSonicSensorCM(4) > 25)   // дальность обнаружения препятствий установлена
                                        // на 25 сантиметров
  {
    prizm.setMotorPowers(35,35);        // двигаться передним ходом, пока не обнаружится препятствие
    prizm.setGreenLED(LOW);             // выключить красный светодиод
    prizm.setGreenLED(HIGH);            // включить зелёный светодиод
  }
  else
  {
   prizm.setGreenLED(LOW);              // выключить зелёный светодиод
   prizm.setRedLED(HIGH);               // обнаружено препятствие, включить красный светодиод
   prizm.setMotorPowers(125 125);       // остановиться, обнаружено препятствие
   delay(500);
   prizm.setMotorPowers(-35,-35);       // двигаться задним ходом
   delay(1000);
   prizm.setMotorPowers(125,125);       // остановиться
   delay(500);
   prizm.setMotorPowers(35,-35);        // повернуть вправо
   delay(500);

  }
}
```

Пример 14: TaskBot_Act15_Combining_Sensors

```
/*   пример программы для контроллера PRIZM
 *   Программа одновременно использует датчик линии и ультразвуковой датчик.
 *   Робот должен двигаться по краю чёрной линии (полосы) на белой поверхности и искать какой-нибудь
 *   объект на своём пути. При обнаружении препятствия он должен дождаться, чтобы препятствие убрали,
*   после чего продолжить движение и поиск.
 *  Искатель линии подсоединён к цифровому порту D4. Ультразвуковой датчик подсоединён к
 *  цифровому порту D4. Красный светодиод показывает состояние искателя линии. Зелёный светодиод
*    показывает  состояние звукового датчика.
 *    автор PWU 08/05/2016
 */

  #include <PRIZM.h>                          // добавить библиотеку для контроллера PRIZM
  PRIZM prizm;                                // создать экземпляр объекта PRIZM "prizm",
                                              // чтобы можно было использовать его функции

void setup() {

  prizm.PrizmBegin();                         // начать работу контроллера PRIZM
  prizm.setMotorInvert(1,1);                  // заменить вращение электродвигателя постоянного тока № 1
                                              // на обратное для согласования направления вращения
                                              // приводных электродвигателей,
                                              // расположенных друг напротив друга
  prizm.setServoSpeed(1,50);                  // задать сервоприводу № 1 скорость 50
}

void loop() {
    if(prizm.readLineSensor(3) == 1) {
       prizm.setMotorPowers(30,125);          // обнаружена линия
       prizm.setRedLED(HIGH);
     }
     else
     {
       prizm.setMotorPowers(125,30);          // линии не обнаружены
       prizm.setRedLED(LOW);
     }

    while(prizm.readSonicSensorCM(4) < 25) {  // на пути объект, продолжать цикл здесь, пока путь не расчистят
         prizm.setGreenLED(HIGH);             // включить зелёный светодиод
         prizm.setMotorPowers(125 125);       // остановиться, обнаружено препятствие
         prizm.setServoPosition(1,0);         // поднять метку обнаружения!
     }

  prizm.setGreenLED(LOW);                     // выключить зелёный светодиод
  prizm.setServoPosition(1,90);               // понизить положение метки
}
```

PITSCO TETRIX MAX

Контроллер к робототехническим моделям серии TETRIX® PRIZM™ Руководство по программированию



Бесплатный звонок
800•835•0686

Загляните на наш сайт
TETRIXrobotics.com

PITSCO EDUCATION