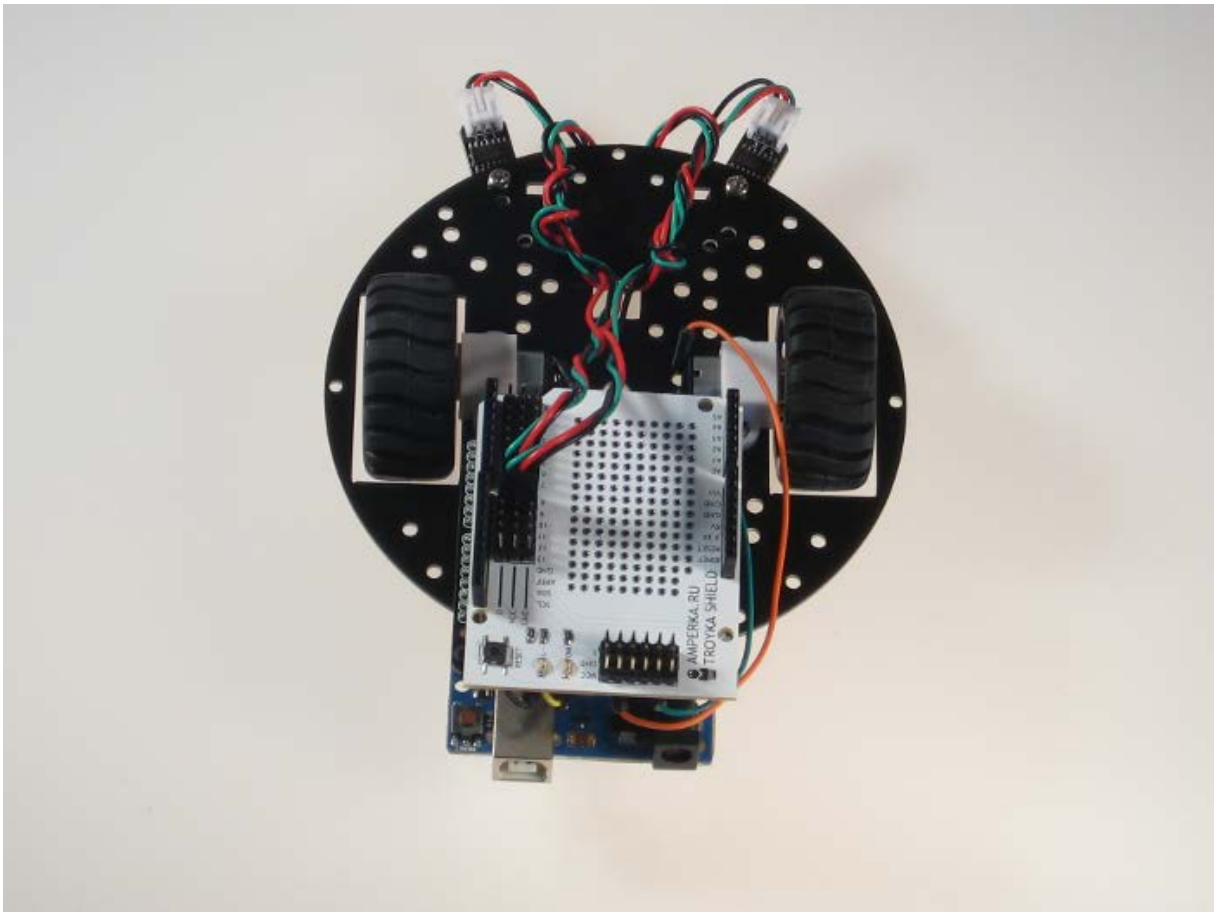
Робот, ездящий по линии под управлением Arduino

В данной статье будет описан процесс создания робота, ездящего по линии. Эта задача является классической, идейно простая, она может решаться много раз, и каждый раз вы будете открывать для себя что-то новое. Решение этой задачи и реализация полученного решения позволяют приобрести необходимые начальные навыки для дальнейшего совершенствования в робототехнике.

Существует множество подходов для решения задачи следования по линии. Выбор одного из них зависит от конкретной конструкции робота, от количества сенсоров, их расположения относительно колёс и друг друга.

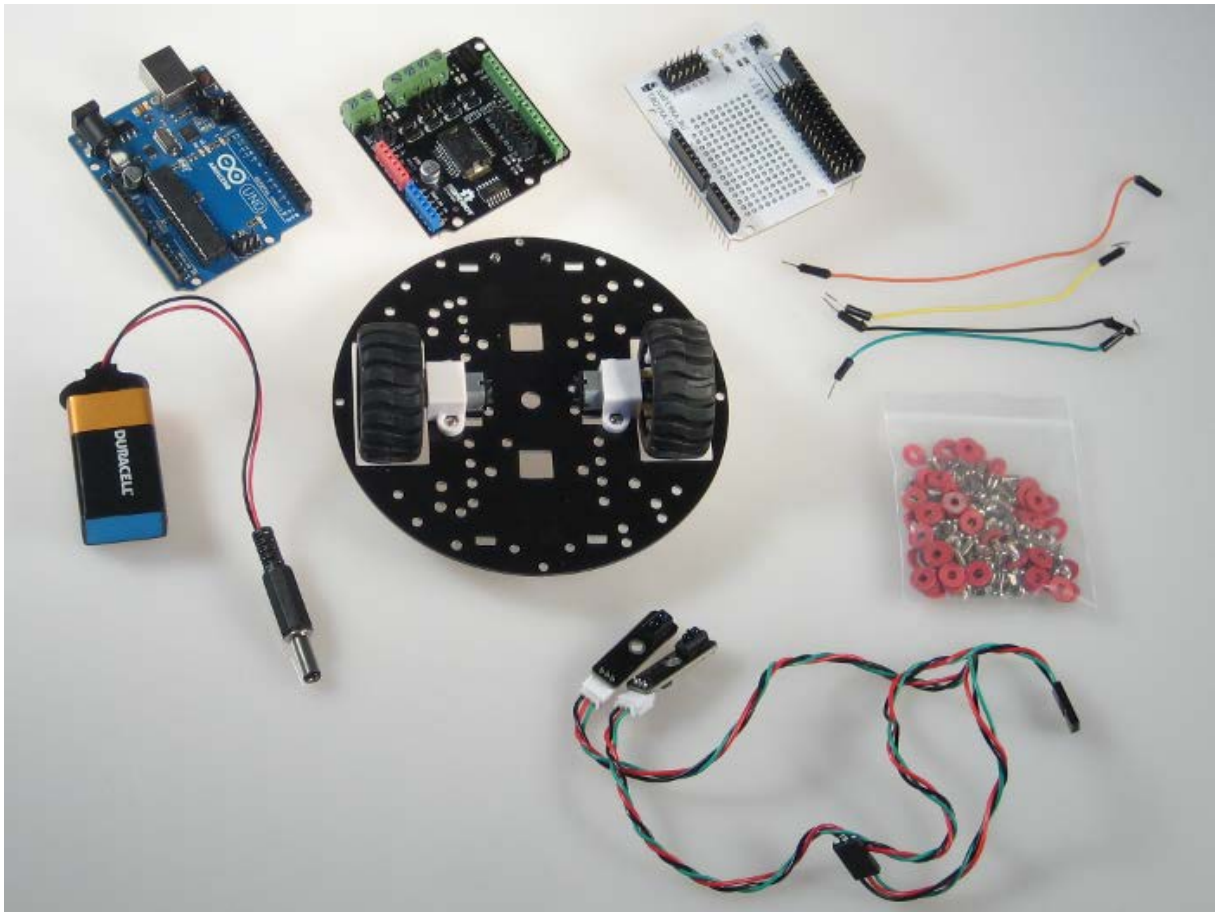
В нашем примере будет собран робот на лёгкой платформе с двумя колёсами и двумя датчиками линии, расположенными на днище робота перед колёсами.

В результате выглядеть он будет так:



Что понадобится

Для нашего примера понадобятся следующие детали:



- Arduino Uno
- Двухколёсная платформа miniQ
- Motor Shield
- Troyka Shield
- Пара датчиков линий
- Несколько соединительных проводов и болтов и гаек для крепления датчиков и Arduino Uno
- Кабель питания от батарейки Крона и сама батарейка

  Вообще говоря, лучше было бы использовать NiMH-аккумуляторы: они лучше отдают ток и значительно дольше держат напряжение, но для целей этого проекта одной батарейки на 9 В вполне хватило.

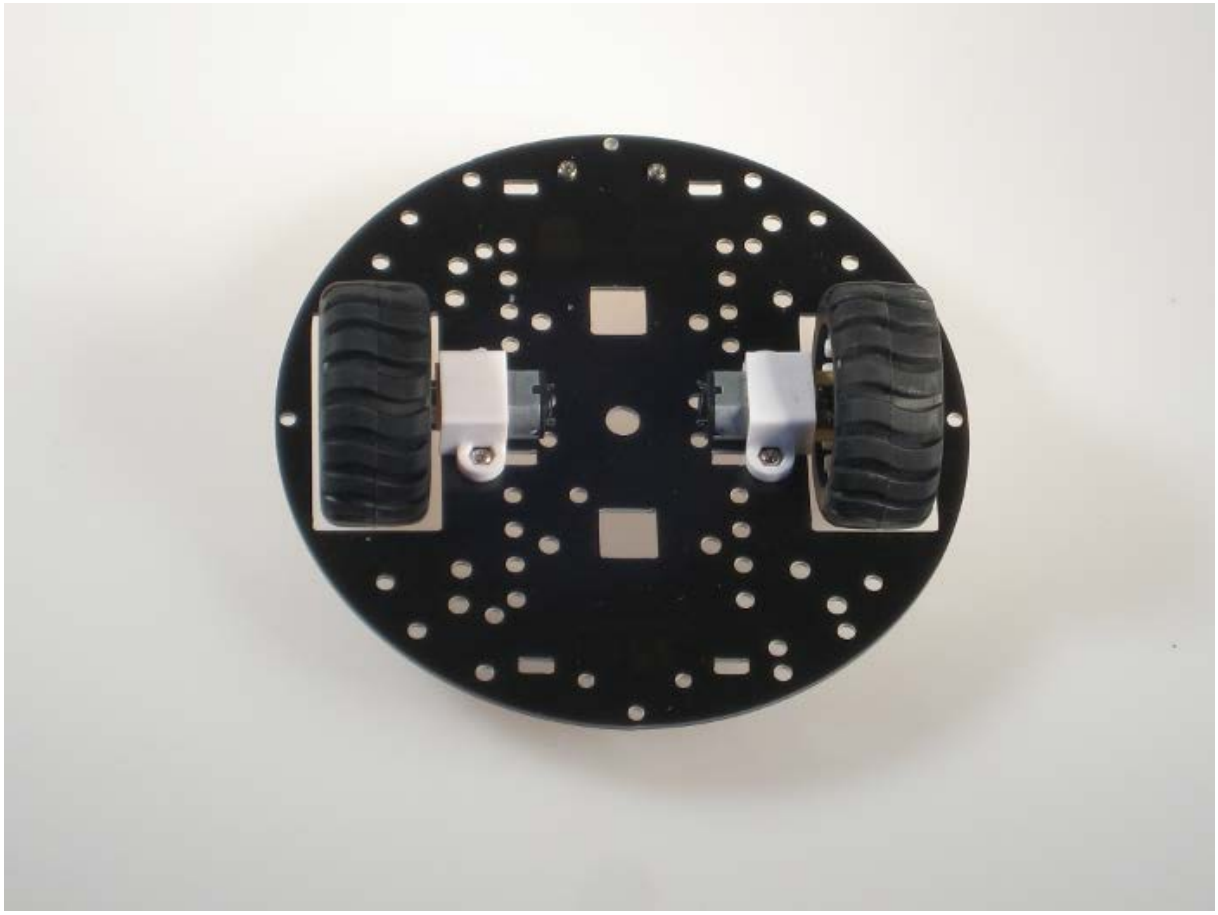
Собираем робота

Сначала соберём робота, установим всю механику и электронику.

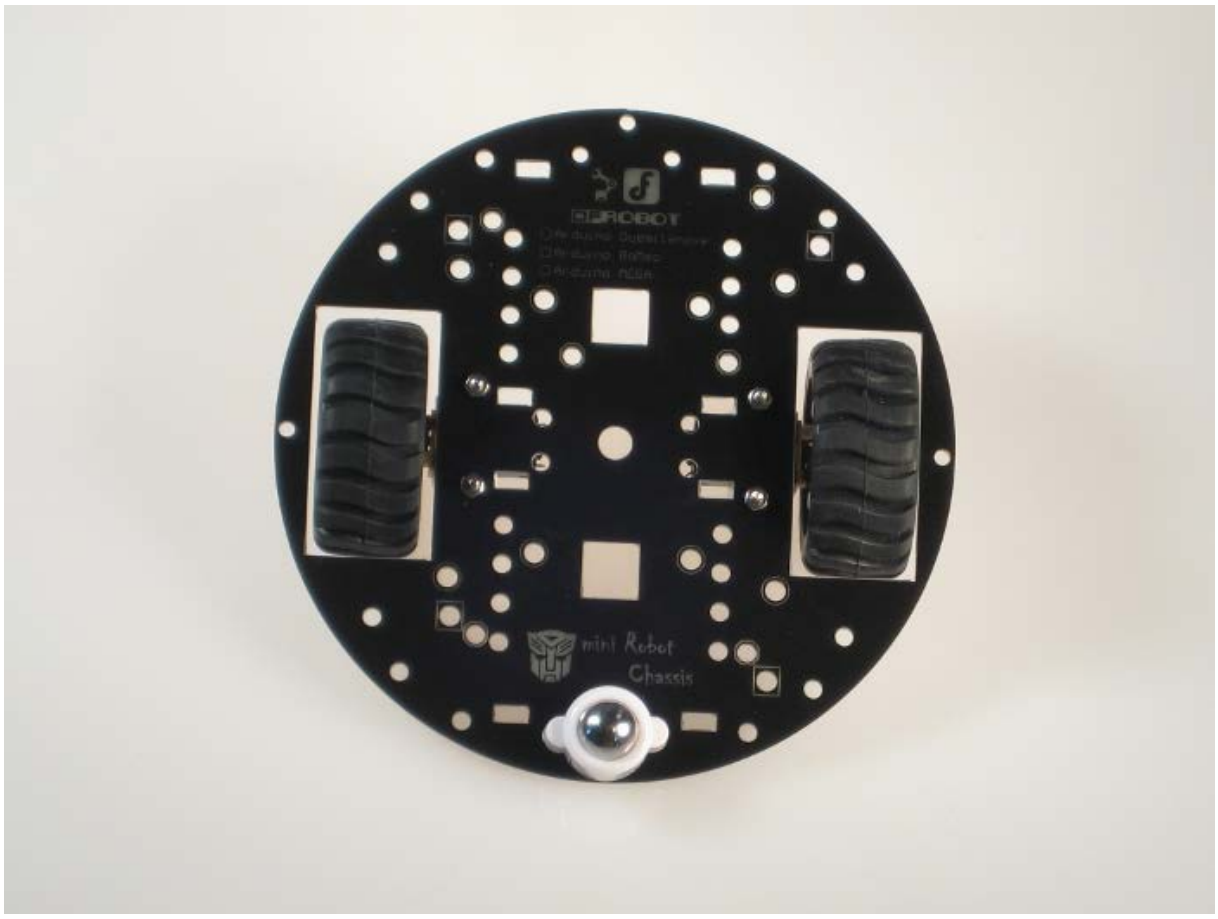
Собираем платформу

Для начала прикрепим колёса к моторам.

Затем с помощью пластиковых П-образных креплений прикручиваем моторчики к платформе. Обратите внимание на взаимное расположение крепления и моторчики: в креплении есть небольшие углубления, так что если всё соединить правильно, то моторчики будут крепко держаться и никуда не выскочат.



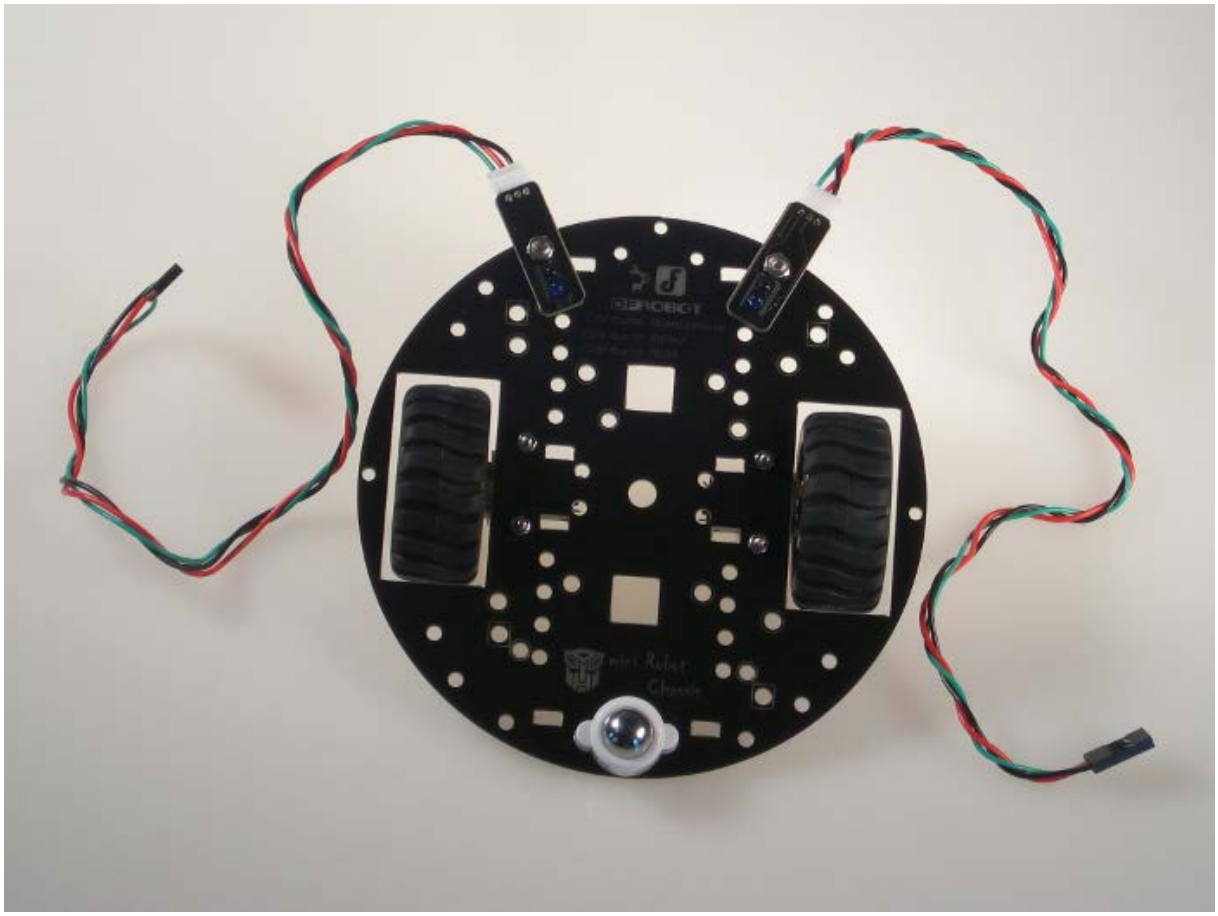
Теперь крепим балансировочный шар.



Отлично! Платформа собрана. Если вам кажется, что колёсам отведено слишком мало места и они трутся о платформу, то скорее всего вам нужно посильнее надавить на колёса, чтобы они плотнее сели на вал мотора.

Крепим сенсоры

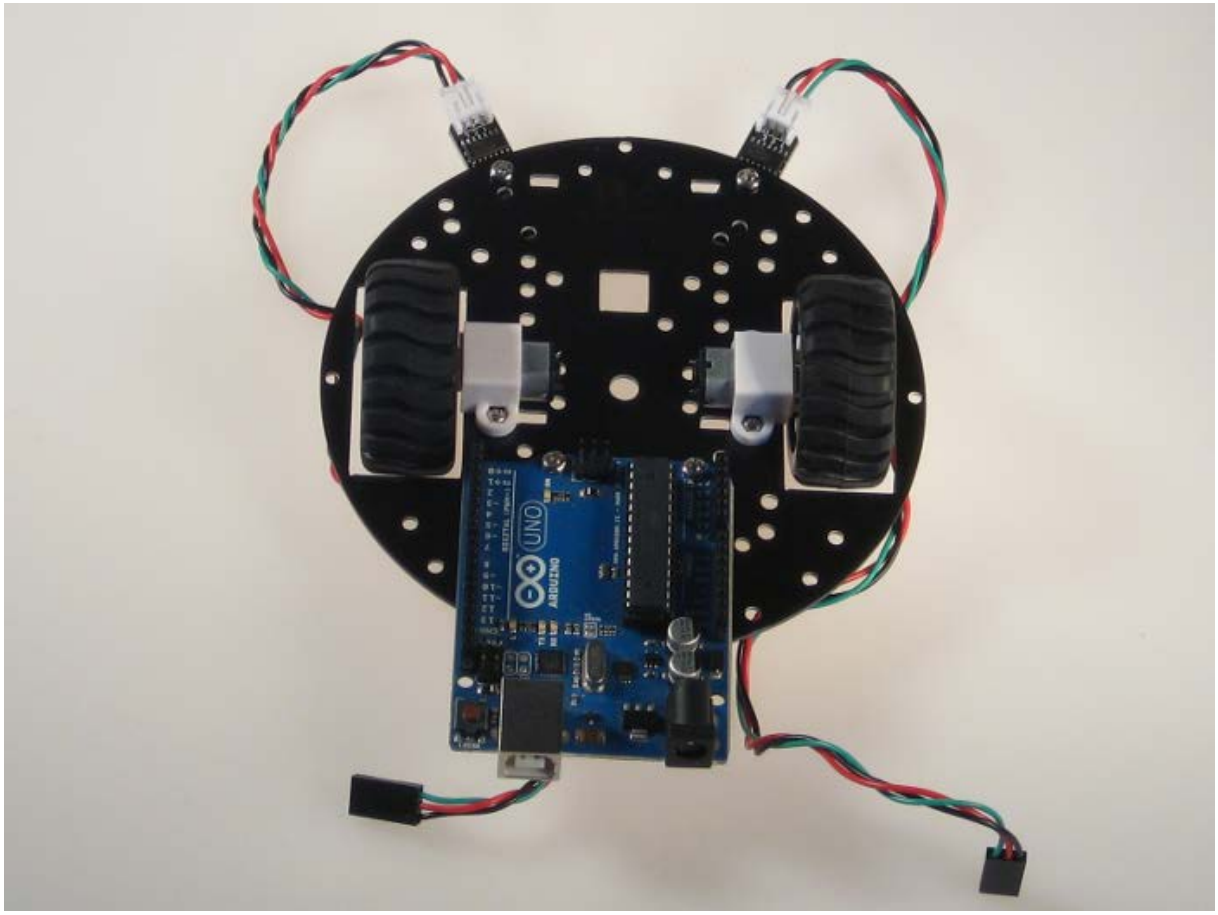
Закрепим их, как показано на фото:



Можно было бы выбрать и другое место. Это могло бы сделать контроль проще или сложнее, а самого робота более или менее эффективным. Оптимальное расположение — вопрос серии экспериментов. Для этого проекта просто был выбран такой способ крепления.

Крепим Arduino

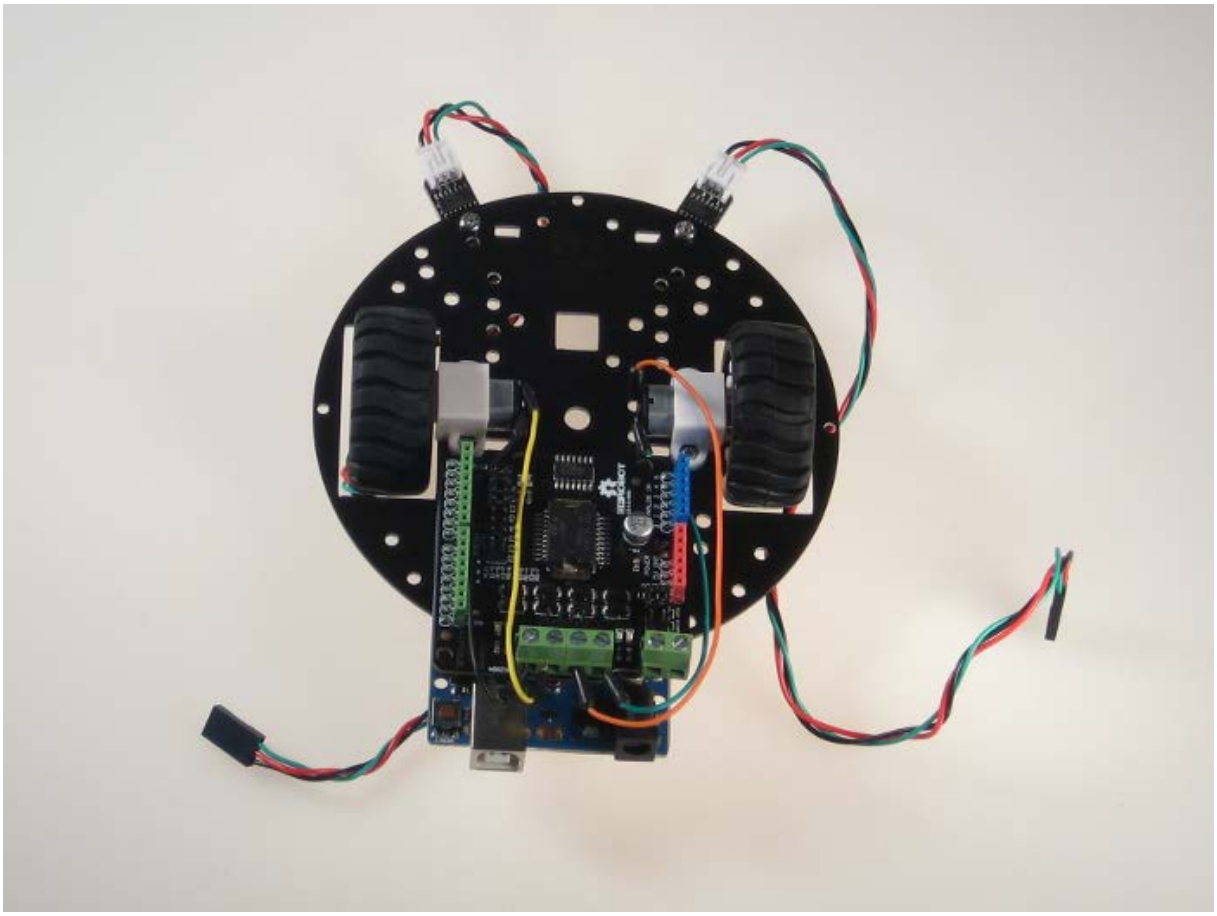
Arduino закрепим с противоположной стороны двумя винтиками и гайками.



Опять же, можно выбрать и другое место. Например над колёсами, если приподнять Arduino на латунных стойках. Это изменило бы положение центра масс и повлияло бы на эффективность робота в лучшую или худшую сторону.

Крепим Motor Shield и соединительные провода

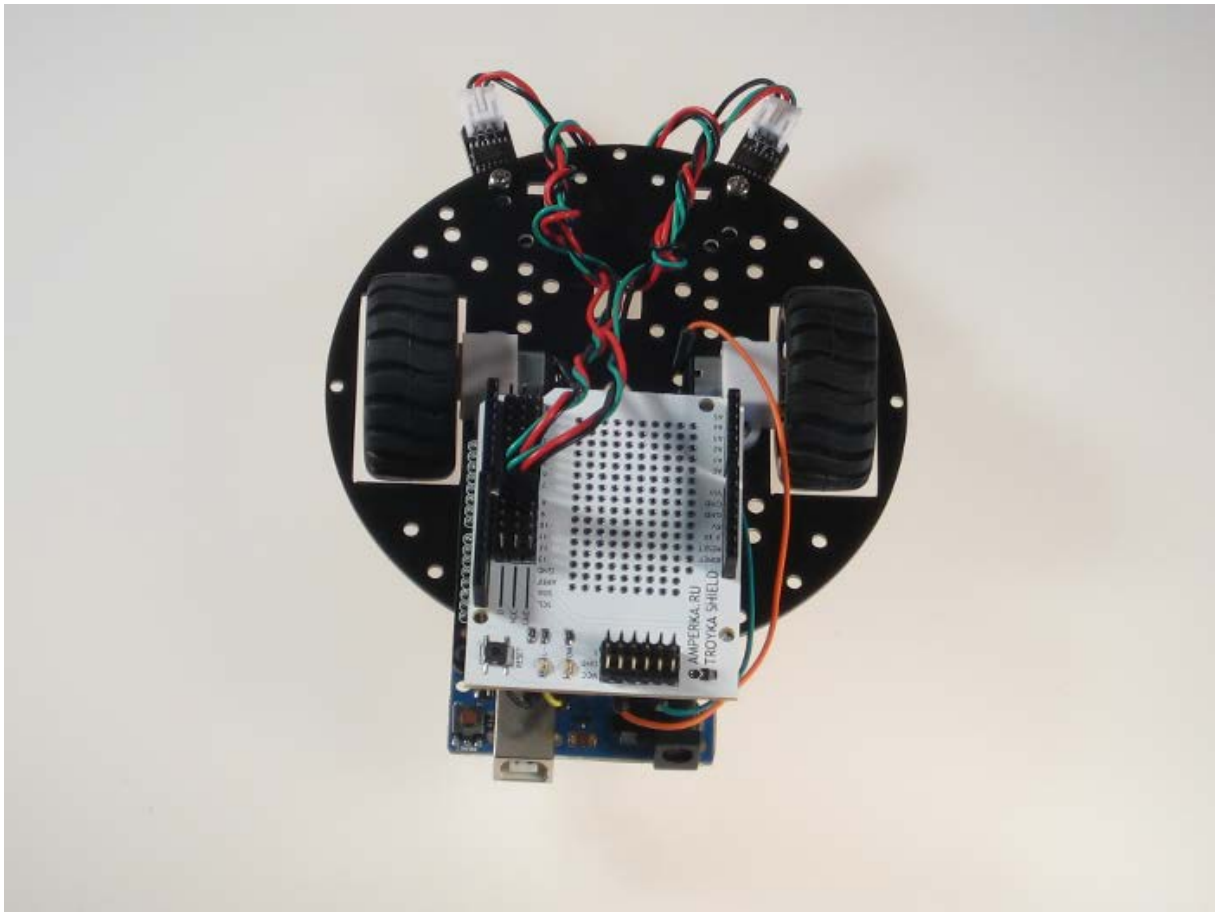
Установим Motor Shield на Arduino и подсоединим соединительные провода. Обратите внимание, чтобы соотвествовать программному коду из примера ниже, моторчики соединены с Motor Shield так: правый — к клеммам M1 с прямой полярностью (плюс к плюсу), а левый — к M2 с обратной (плюс к минусу).



В этом проекте, для экономии времени концы соединительных проводов просто скручены с контактами моторов. При работе «начисто» стоит жёстко припаять провода к моторам.

Крепим Troyka Shield

Присоединяем сверху Troyka Shield и подключаем датчики к 8 и 9 цифровым контактам. В итоге получаем следующую конструкцию:



Программирование

Теперь напишем программу, которая заставит собранную конструкцию двигаться по нарисованной линии. В проекте мы будем использовать чёрную линию, напечатанную на белых листах бумаги.

Основная идея алгоритма

Пусть у нас усть белое поле, и на нём чёрным нарисован трек для нашего робота. Используемые датчики линии выдают логический ноль, когда «видят» чёрное и единицу, когда «видят» белое.

На прямой робот должен пропускать трек между сенсоров, то есть оба сенсора должны показывать единички.

При повороте траектории направо, правый сенсор наезжает на трек и начинает показывать логический ноль. При повороте налево, ноль показывает левый сенсор.

Таким образом получаем простую систему с тремя состояниями:

- `STATE_FORWARD` — нужно ехать вперёд
- `STATE_RIGHT` — нужно поворачиваться направо
- `STATE_LEFT` — нужно поворачиваться налево

На вход системы поступает информация с сенсоров. Получаем следующую логику переходов:

Левый	Правый	Целевое состояние
0	0	`STATE_FORWARD`
0	1	`STATE_RIGHT`
1	0	`STATE_LEFT`
1	1	`STATE_FORWARD`

Реализация на Arduino

LineRobot_v1.ino

```
// Моторы подключаются к клеммам M1+,M1-,M2+,M2-
// Motor shield использует четыре контакта 6,5,7,4 для управления моторами
#define SPEED_LEFT       6
#define SPEED_RIGHT      5
#define DIR_LEFT         7
#define DIR_RIGHT        4
#define LEFT_SENSOR_PIN  8
#define RIGHT_SENSOR_PIN 9

// Скорость, с которой мы движемся вперёд (0-255)
#define SPEED            35

// Коэффициент, задающий во сколько раз нужно затормозить
// одно из колёс для поворота
#define BRAKE_K          4

#define STATE_FORWARD    0
#define STATE_RIGHT      1
#define STATE_LEFT       2

int state = STATE_FORWARD;

void runForward()
{
    state = STATE_FORWARD;

    // Для регулировки скорости `SPEED` может принимать значения от 0 до 255,
    // чем болше, тем быстрее.
    analogWrite(SPEED_LEFT, SPEED);
    analogWrite(SPEED_RIGHT, SPEED);

    // Если в DIR_LEFT или DIR_RIGHT пишем HIGH, мотор будет двигать
соответствующее колесо
    // вперёд, если LOW - назад.
    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
```

```
}

void steerRight()
{
    state = STATE_RIGHT;

    // Замедляем правое колесо относительно левого,
    // чтобы начать поворот
    analogWrite(SPEED_RIGHT, SPEED / BRAKE_K);
    analogWrite(SPEED_LEFT, SPEED);

    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
}

void steerLeft()
{
    state = STATE_LEFT;

    analogWrite(SPEED_LEFT, SPEED / BRAKE_K);
    analogWrite(SPEED_RIGHT, SPEED);

    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
}


void setup()
{
    // Настраивает выводы платы 4,5,6,7 на вывод сигналов
    for(int i = 4; i <= 7; i++)
        pinMode(i, OUTPUT);

    // Сразу едем вперёд
    runForward();
}

void loop()
{
    // Наш робот ездит по белому полю с чёрным треком. В обратном случае не
нужно
    // инвертировать значения с датчиков
    boolean left = !digitalRead(LEFT_SENSOR_PIN);
    boolean right = !digitalRead(RIGHT_SENSOR_PIN);

    // В какое состояние нужно перейти?
    int targetState;

    if (left == right) {
        // под сенсорами всё белое или всё чёрное
```

```
        // едем вперёд
        targetState = STATE_FORWARD;
    } else if (left) {
        // левый сенсор упёрся в трек
        // поворачиваем налево
        targetState = STATE_LEFT;
    } else {
        targetState = STATE_RIGHT;
    }

    if (state == targetState) {
        // мы уже делаём всё что нужно,
        // делаем измерения заново
        return;
    }

    switch (targetState) {
        case STATE_FORWARD:
            runForward();
            break;

        case STATE_RIGHT:
            steerRight();
            break;

        case STATE_LEFT:
            steerLeft();
            break;
    }

    // не позволяем сильно вилять на прямой
    delay(50);
}
```

Проблема инертности и её решение

Однако если выставить скорость моторов побольше, мы столкнёмся со следующей проблемой: наш робот будет вылетать с трека, не успевая отреагировать на поворот. Это связано с тем, что наши моторчики не умеют тормозить мгновенно.

В этом легко убедиться поставив следующий эксперимент: с заданной скоростью робот будет двигаться по поверхности, и в некоторый момент будет установлена нулевая скорость и измерен тормозной путь робота. Пусть робот разгоняется по монотонной поверхности и тормозится при фиксировании импровизированной стоп-линии.

Эксперимент проведём для разных скоростей. Код программы для эксперимента таков:

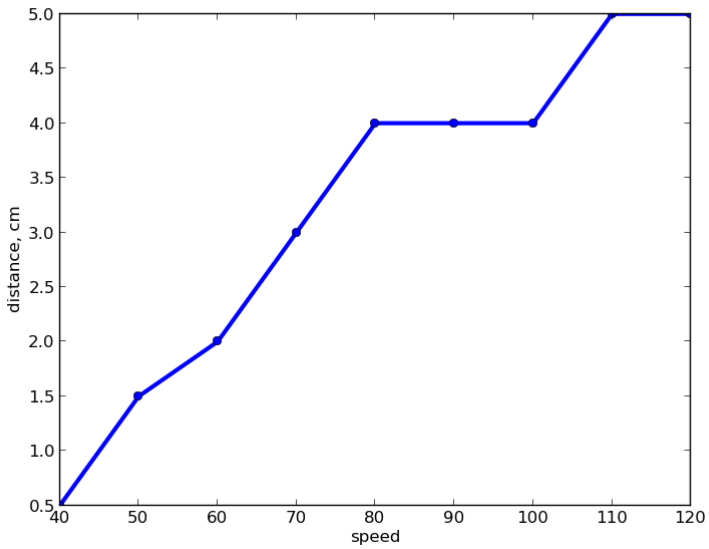
stopping_distance_experiment.ino

```
#define LEFT_SENSOR_PIN   8
#define RIGHT_SENSOR_PIN  9
#define SPEED_LEFT        6
#define SPEED_RIGHT       5
```

```
#define DIR_LEFT          7
#define DIR_RIGHT         4

// Для того чтобы убедиться, что именно тормозной путь долог, а не команда
остановиться
// приходит слишком поздно, будем включать светодиод, когда отдаётся команда.
#define LED_PIN           13

int currSpeed = 40;
void setup()
{
    for(int i = 4; i <= 7; ++i)
        pinMode(i, OUTPUT);

    analogWrite(SPEED_RIGHT, currSpeed);
    digitalWrite(DIR_RIGHT, HIGH);

    analogWrite(SPEED_LEFT, currSpeed);
    digitalWrite(DIR_LEFT, HIGH);

    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    if (currSpeed > 120)
        return;

    boolean white[] = {
        !digitalRead(LEFT_SENSOR_PIN),
        !digitalRead(RIGHT_SENSOR_PIN)
    };

    if (white[0] && white[1]) {
        // едем пока не упрёмся
        return;
    }

    // зажигаем светодиод, останавливаем моторы
    // и наблюдаем
    digitalWrite(LED_PIN, HIGH);
    analogWrite(SPEED_RIGHT, 0);
    analogWrite(SPEED_LEFT, 0);
    delay(5000);

    // повторяем эксперимент, увеличивая скорость
    // на 10 пунктов
    currSpeed += 10;
    if (currSpeed > 120)
        return;
```

```
    digitalWrite(LED_PIN, LOW);
    analogWrite(SPEED_RIGHT, currSpeed);
    analogWrite(SPEED_LEFT, currSpeed);
}
```

На той поверхности, на которой проводился эксперимент, были получены следующие результаты:



Таким образом, начиная с некоторого момента у нашего робота нет никакой возможности успеть среагировать и остаться на треке.

Что можно сделать?! После того, как сенсоры улавливают поворот, можно остановиться и вернуться назад на некоторое расстояние, зависящее от скорости перед остановкой. Однако мы можем отдать команду роботу ехать с какой-то скоростью, но не можем приказать ему проехать какое-то расстояние.

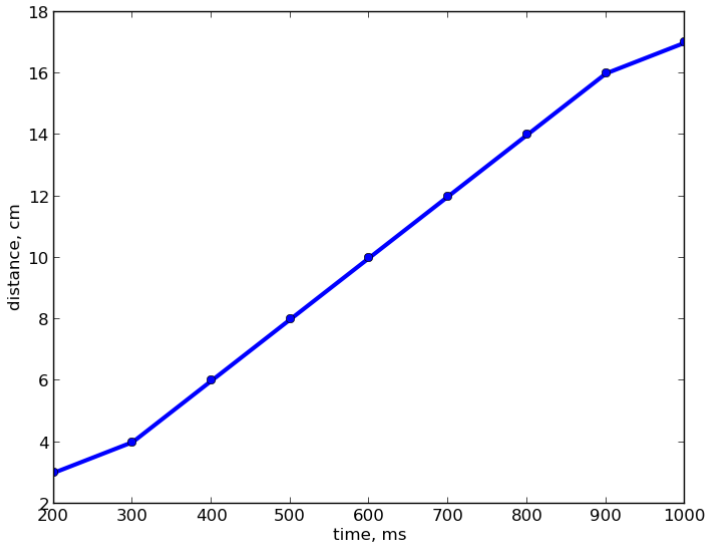
Для того, чтобы понять зависимость расстояния при заднем ходе от времени, был проведён ещё один замер:

time_distance_rate.ino

```
#define SPEED_LEFT      6
#define SPEED_RIGHT     5
#define DIR_LEFT        7
#define DIR_RIGHT       4

void go(int speed, bool reverseLeft, bool reverseRight, int duration)
{
    analogWrite(SPEED_LEFT, speed);
    analogWrite(SPEED_RIGHT, speed);
    digitalWrite(DIR_LEFT, reverseLeft ? LOW : HIGH);
    digitalWrite(DIR_RIGHT, reverseRight ? LOW : HIGH);
    delay(duration);
}

void setup()
```

```
{
    for(int i = 4; i <= 7; ++i)
        pinMode(i, OUTPUT);
}

void loop()
{
    // Задержка 5 секунд после включения питания
    delay(5000);

    for (int i = 200; i <= 1000; i += 100) {
        // Несколько сотен мс вперёд
        go(50, false, false, 200);
        go(0, false, false, 0);

        // Задержка 5 секунд
        delay(5000);
    }

    // Остановка до ресета или выключения питания
    go(0, false, false, 0);

    // Приехали
    while (true)
        ;
}
```

На скорости 50, например, робот проделывал путь, зависящий от времени следующим образом:



Полученные две зависимости были линейно аппроксимированы, затем была выведена формула зависимости времени, которое надо двигаться назад, от скорости перед остановкой.

Обратим внимание на то, что у вас значения могут оказаться другими: из-за особенностей сборки либо из-за поверхности, поэтому в общем случае лучше провести все измерения самостоятельно.

Адаптивное поведение

Перед финальным экспериментом произведём ещё несколько поправок.

Во-первых, нам необязательно давать команду ехать назад перед каждым поворотом, как мы помним, на маленькой скорости робот прекрасно справляется и без этого. К тому же лучше ему двигаться не прямо назад, а немного поворачивая, всё-таки робот находится перед поворотом.

Во-вторых, нам стоит различать состояния робота: когда он движется по прямой, и ничто ему не мешает ускоряться; и когда робот входит в поворот. В первом случае действительно будем увеличивать скорость робота для более динамичного прохождения трека, во втором случае будем сбрасывать скорость до значения, достаточного для успешного прохождения поворота, и будем держать эту скорость ещё какое-то время.

В итоге наш код будет выглядит следующим образом:

Robot_v02.ino

```
// Моторы подключаются к клеммам M1+,M1-,M2+,M2-
// Motor shield использует четыре контакта 6,5,7,4 для управления моторами
#define SPEED_LEFT       6
#define SPEED_RIGHT      5
#define DIR_LEFT         7
#define DIR_RIGHT        4
#define LEFT_SENSOR_PIN  8
#define RIGHT_SENSOR_PIN 9

// Скорость, с которой мы движемся вперёд (0-255)
#define SPEED            100

// Скорость прохождения сложных участков
#define SLOW_SPEED       35

#define BACK_SLOW_SPEED  30
#define BACK_FAST_SPEED  50

// Коэффициент, задающий во сколько раз нужно затормозить
// одно из колёс для поворота
#define BRAKE_K          4

#define STATE_FORWARD    0
#define STATE_RIGHT      1
#define STATE_LEFT       2

#define SPEED_STEP       2

#define FAST_TIME_THRESHOLD     500
```

```
int state = STATE_FORWARD;
int currentSpeed = SPEED;
int fastTime = 0;

void runForward()
{
    state = STATE_FORWARD;

    fastTime += 1;
    if (fastTime < FAST_TIME_THRESHOLD) {
        currentSpeed = SLOW_SPEED;
    } else {
        currentSpeed = min(currentSpeed + SPEED_STEP, SPEED);
    }

    analogWrite(SPEED_LEFT, currentSpeed);
    analogWrite(SPEED_RIGHT, currentSpeed);

    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
}

void steerRight()
{
    state = STATE_RIGHT;
    fastTime = 0;

    // Замедляем правое колесо относительно левого,
    // чтобы начать поворот
    analogWrite(SPEED_RIGHT, 0);
    analogWrite(SPEED_LEFT, SPEED);

    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
}

void steerLeft()
{
    state = STATE_LEFT;
    fastTime = 0;

    analogWrite(SPEED_LEFT, 0);
    analogWrite(SPEED_RIGHT, SPEED);

    digitalWrite(DIR_LEFT, HIGH);
    digitalWrite(DIR_RIGHT, HIGH);
}


void stepBack(int duration, int state) {
```

```
    if (!duration)
        return;

    // В зависимости от направления поворота при движении назад будем
    // делать небольшой разворот
    int leftSpeed = (state == STATE_RIGHT) ? BACK_SLOW_SPEED : BACK_FAST_SPEED;
    int rightSpeed = (state == STATE_LEFT) ? BACK_SLOW_SPEED : BACK_FAST_SPEED;

    analogWrite(SPEED_LEFT, leftSpeed);
    analogWrite(SPEED_RIGHT, rightSpeed);

    // реверс колёс
    digitalWrite(DIR_RIGHT, LOW);
    digitalWrite(DIR_LEFT, LOW);

    delay(duration);
}


void setup()
{
    // Настраивает выводы платы 4,5,6,7 на вывод сигналов
    for(int i = 4; i <= 7; i++)
        pinMode(i, OUTPUT);

    // Сразу едем вперёд
    runForward();
}

void loop()
{
    // Наш робот ездит по белому полю с чёрным треком. В обратном случае не
нужно
    // инвертировать значения с датчиков
    boolean left = !digitalRead(LEFT_SENSOR_PIN);
    boolean right = !digitalRead(RIGHT_SENSOR_PIN);

    // В какое состояние нужно перейти?
    int targetState;

    if (left == right) {
        // под сенсорами всё белое или всё чёрное
        // едем вперёд
        targetState = STATE_FORWARD;
    } else if (left) {
        // левый сенсор упёрся в трек
        // поворачиваем налево
        targetState = STATE_LEFT;
    } else {
        targetState = STATE_RIGHT;
```

```
    }

    if (state == STATE_FORWARD && targetState != STATE_FORWARD) {
        int brakeTime = (currentSpeed > SLOW_SPEED) ?
            currentSpeed : 0;
        stepBack(brakeTime, targetState);
    }

    switch (targetState) {
        case STATE_FORWARD:
            runForward();
            break;

        case STATE_RIGHT:
            steerRight();
            break;

        case STATE_LEFT:
            steerLeft();
            break;
    }

}
```