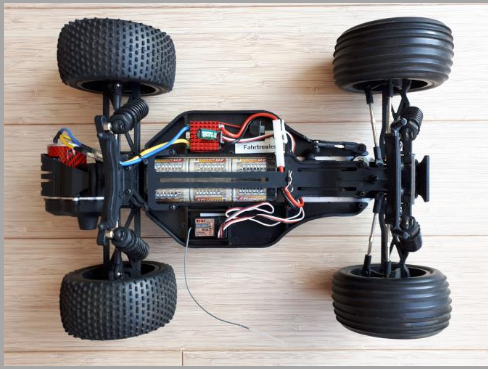
Introduction to Arduino

Controlling RC model cars



HAW HAMBURG

Prof. Dr. Marc Hensel

http://www.haw-hamburg.de/marc-hensel

Contents

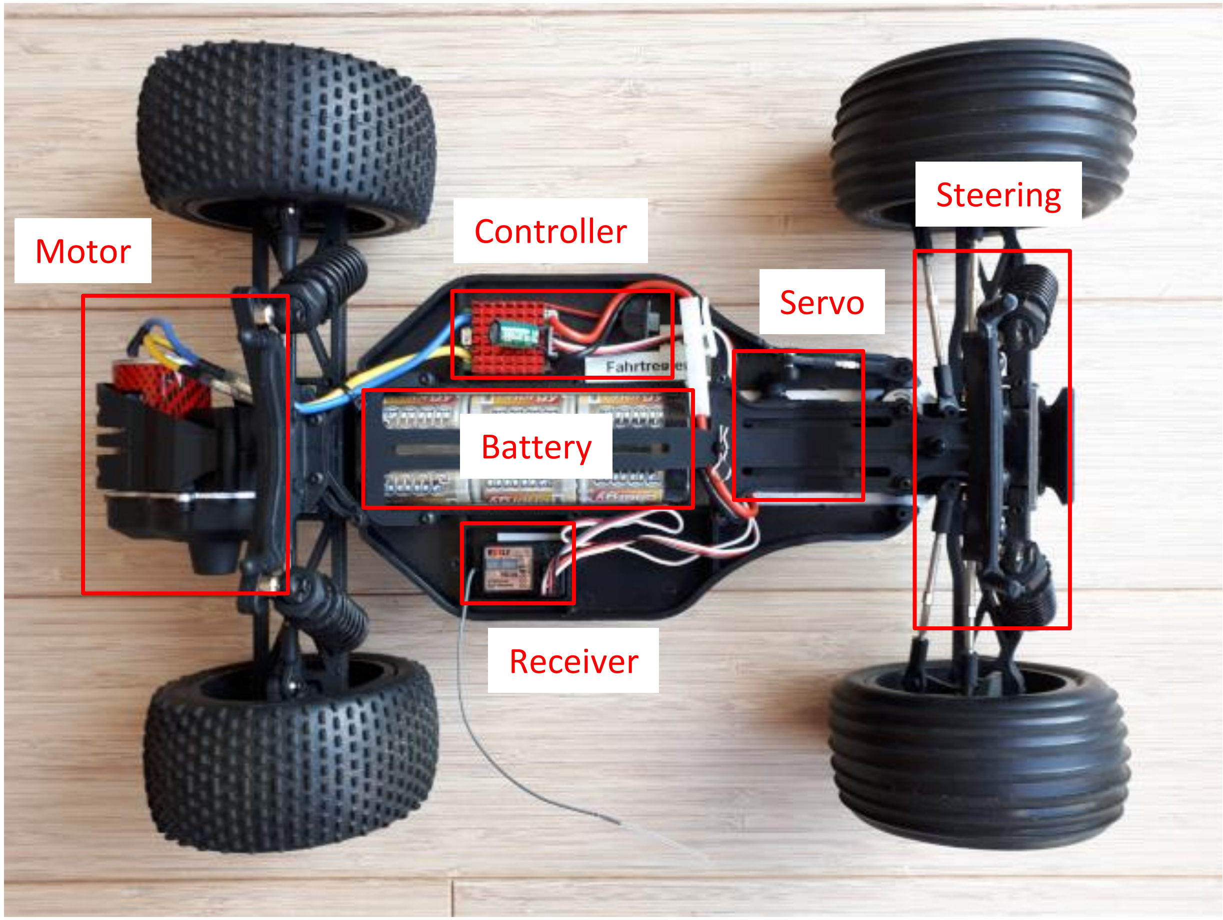
Be aware, this is a *Getting Started*, not an extensive course. Enjoy! ☺

Receiver channels & signals

- You know the main components of a RC car.
- You understand how the receiver interacts with steering and throttle.
- You know the principal signals generated by the receiver.

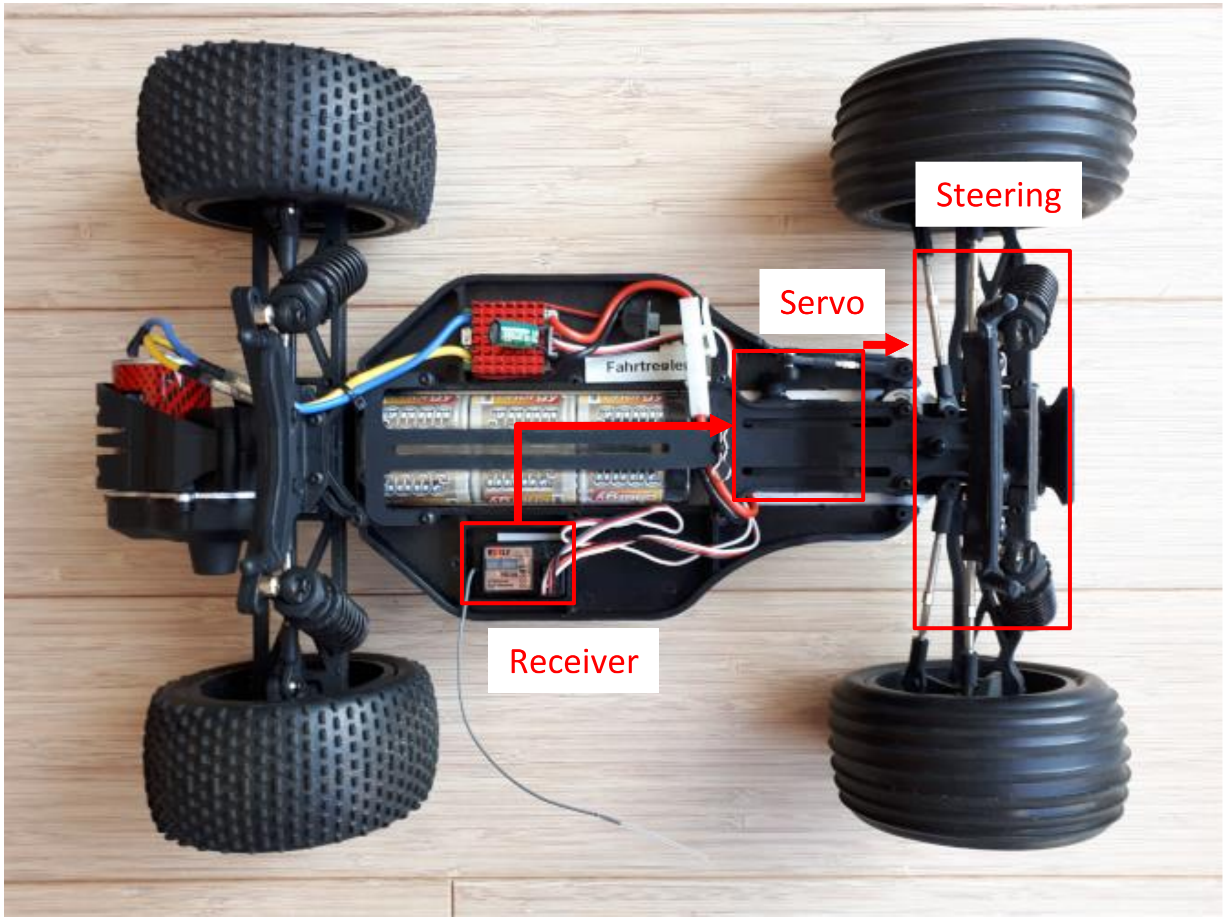
Principle components of the RC car

Example of a car with receiver, servo, and controller easily accessible:



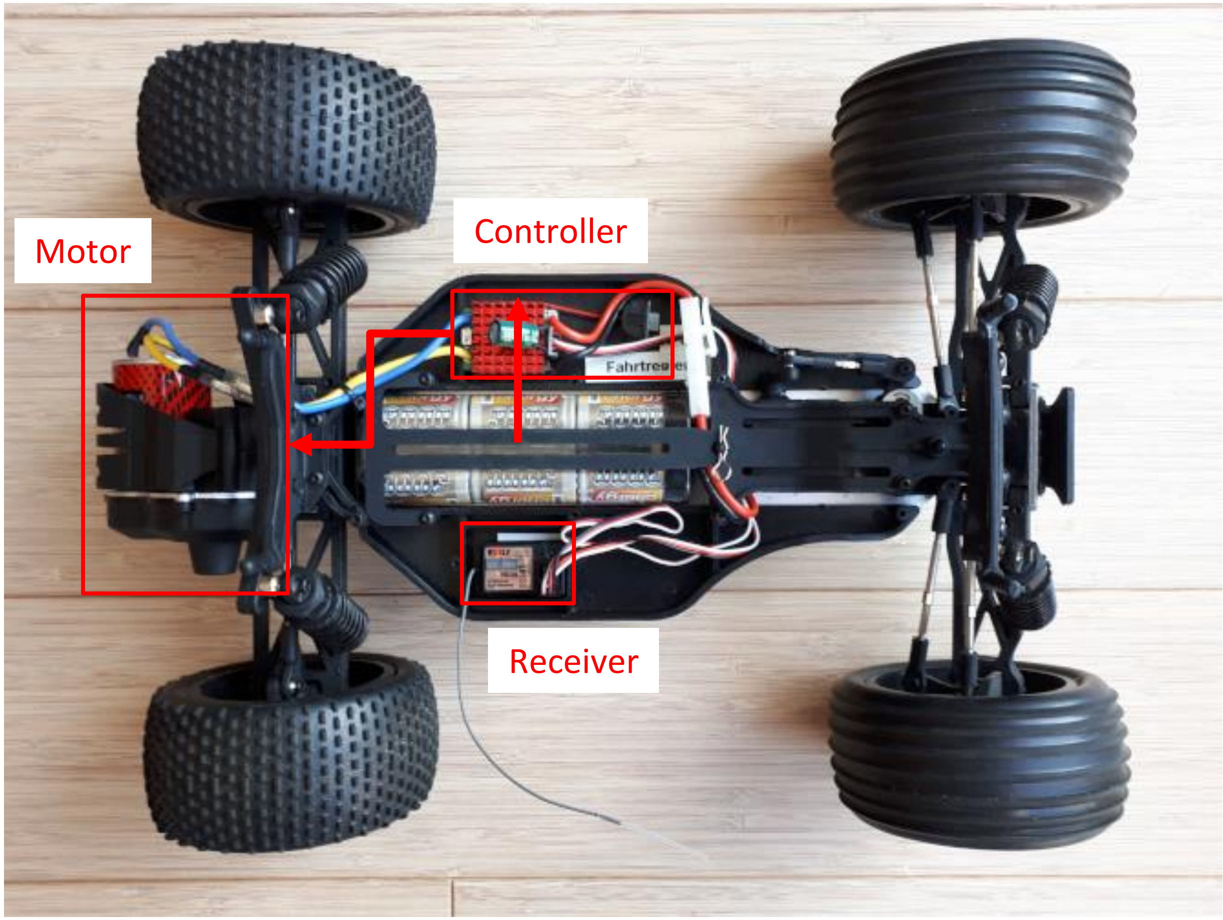
Receiver channel 1

Channel 1 controls the servo position (steering):



Receiver channel 2

Channel 2 sets the electronic speed controller ESC (throttle):



Receiver signals

For the direct approach:

- The receiver generates signals to control the steering servo and throttle.
- We want to generate these signals with an Arduino, instead.
- Hence, we need to know how these signals look like.

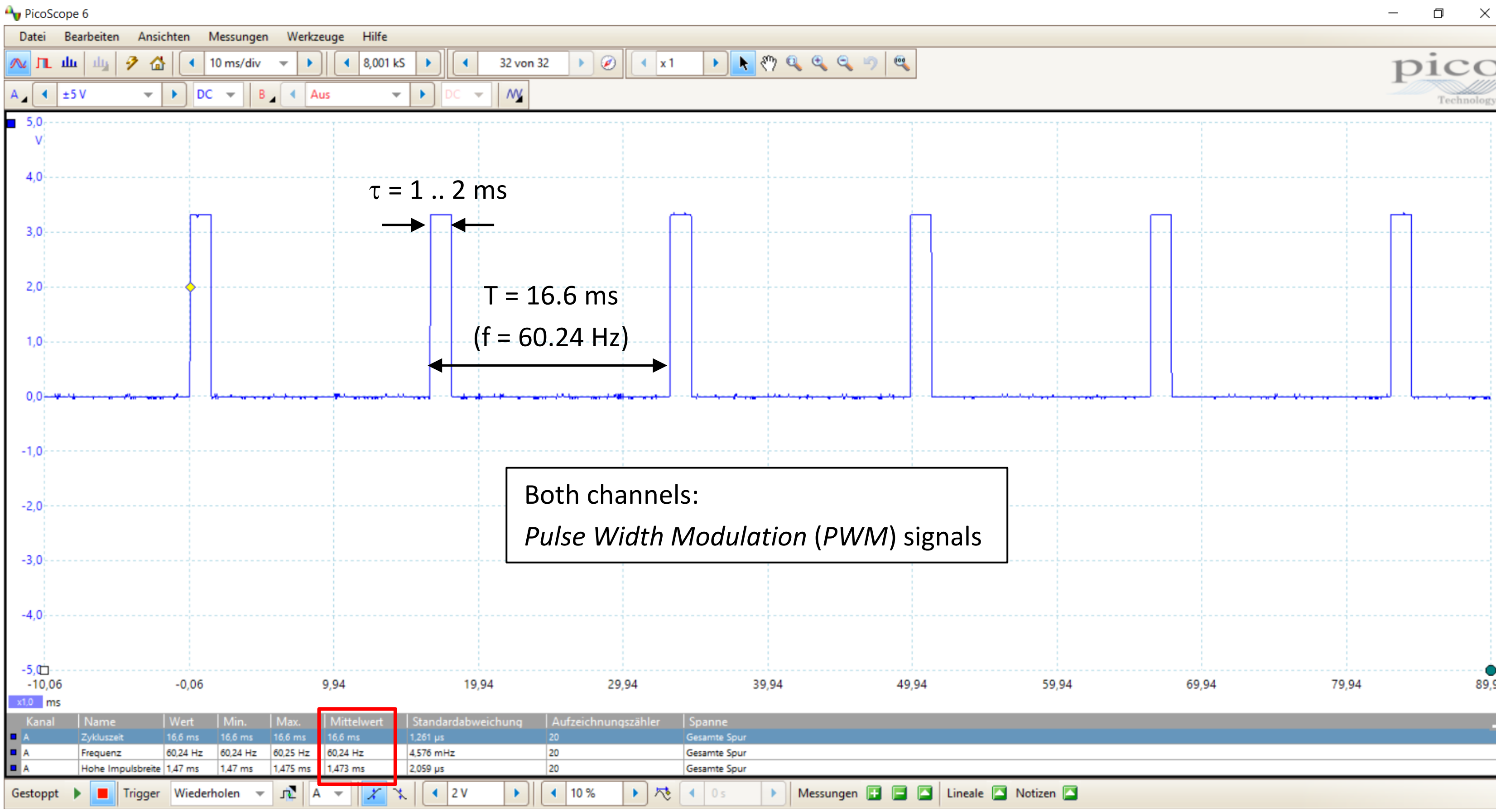
Let’s find out:

- It is a good exercise to measure the receiver signals with an oscilloscope.
- Move the steering to full left and full right and observe the signal.
- Use the full range of the throttle and observe the signal.

If you do not have access to an oscilloscope, don’t worry. Just look at the next slide.

Receiver signals

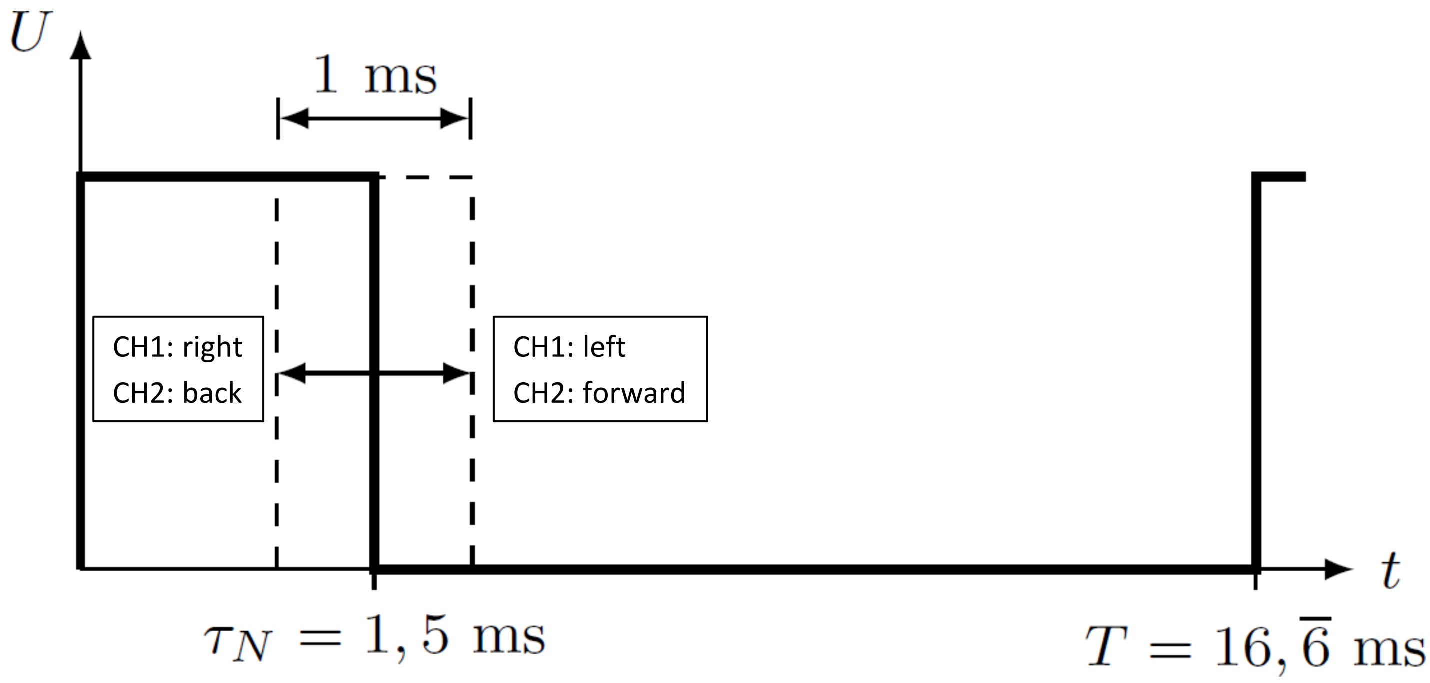
Here is what my oscilloscope came up with:



Receiver signals

Both signals (servo channel 1 and controller channel 2) look the same:

- Pulse width modulation (PWM) signals (i. e., periodic on/off pulses)
- Pulses of 1 to 2 ms duration control the steering angle and speed
- Pulses are repeated with about 60 Hz



$\Rightarrow$ Good! That's easy to generate with an Arduino.

Direct control

- You generate PWM signals using the Arduino.
- You control steering and throttle by the generated signals.

Note:

We won’t use Arduino’s PWM pins and the servo library on purpose, so that you get a better feeling for the signals involved.

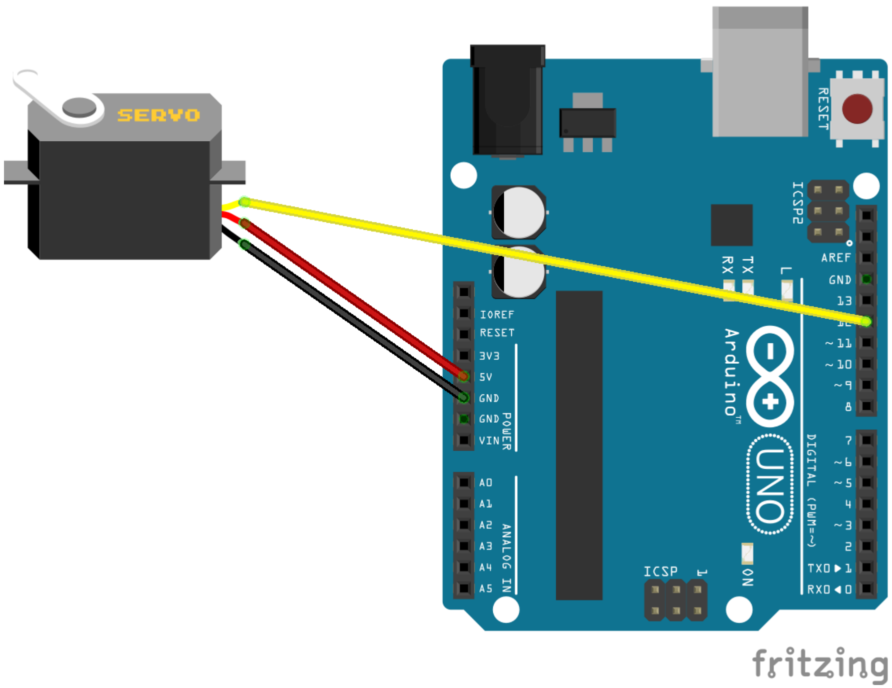
Direct control

Connect the servo to the Arduino:

- Voltage (5V) and ground (GND)
- PWM signal (digital pin 12 – could be another one)

Required functionality:

- Set PWM pin as output pin
- Set pin to HIGH and LOW
- Wait a specific time



Recall the corresponding methods:

- *pinMode(pin, OUTPUT)*
- *digitalWrite(pin, HIGH)*
- *digitalWrite(pin, LOW)*
- *delay(ms)*
- *delayMicroseconds(µs)*

Exercise: Control steering

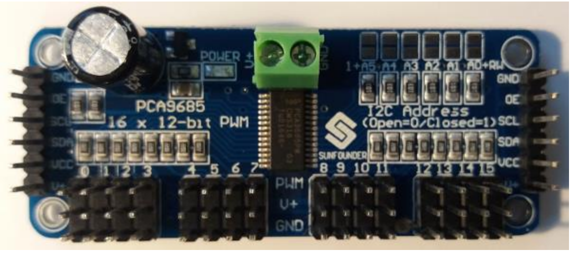
Write a program to repeatedly make the car steer left and right.

Sample solution:

```
#define PIN_PWM_SERVO 12     // PWM output pin
#define PWM_PERIOD 16666     // PWM period in microsecs (60 Hz)
#define PWM_MIN 1100         // Min. pwm pulse in microsecs
#define PWM_MAX 1900         // Max. pwm pulse in microsecs

int pwmServo = 1500;         // Neutral servo position: 1.5 ms
int pwmServoDelta = 10;      // Increment/decrement of pwm pulse in µs

void setup() {
  pinMode(PIN_PWM_SERVO, OUTPUT);
}

void loop() {
  // Turn steering left or right
  pwmServo += pwmServoDelta;
  if ((pwmServo >= PWM_MAX) || (pwmServo <= PWM_MIN))
    pwmServoDelta = -pwmServoDelta;

  // Set PWM pulse to output pin
  digitalWrite(PIN_PWM_SERVO, HIGH);
  delayMicroseconds(pwmServo);
  digitalWrite(PIN_PWM_SERVO, LOW);
  delayMicroseconds(PWM_PERIOD - pwmServo);
}
```

Control using a servo driver board

- You generate PWM signals using a servo driver board.
- You calibrate the required signals for the RC car.
- You control steering and throttle by the servo driver board.



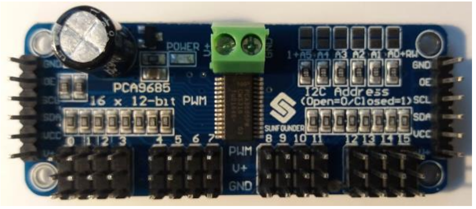
PWM servo driver board

Recall that generating PWM signals directly has its limitations:

- We could use the Arduino to generate both PWM signals in a loop.
- But how to use the Arduino for other (time critical) tasks?

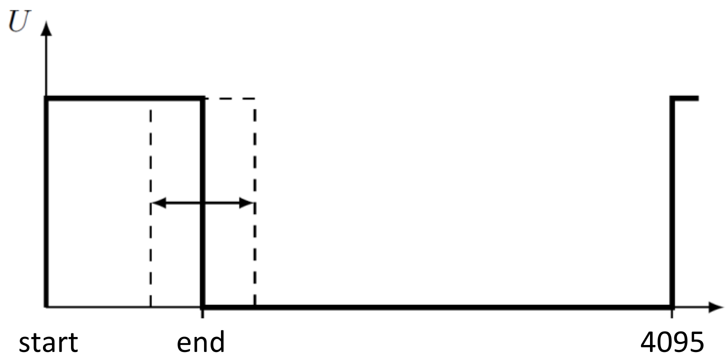
Solution:

- PWM servo driver board PCA9685
- Generates signals for up to 16 servos
- Arduino needs to set frequency and PWM values



Note pulse lengths are not described in ms:

- Set frequency in Hz
- Set pulses' start and end in 0 .. 4095 (*ticks*)

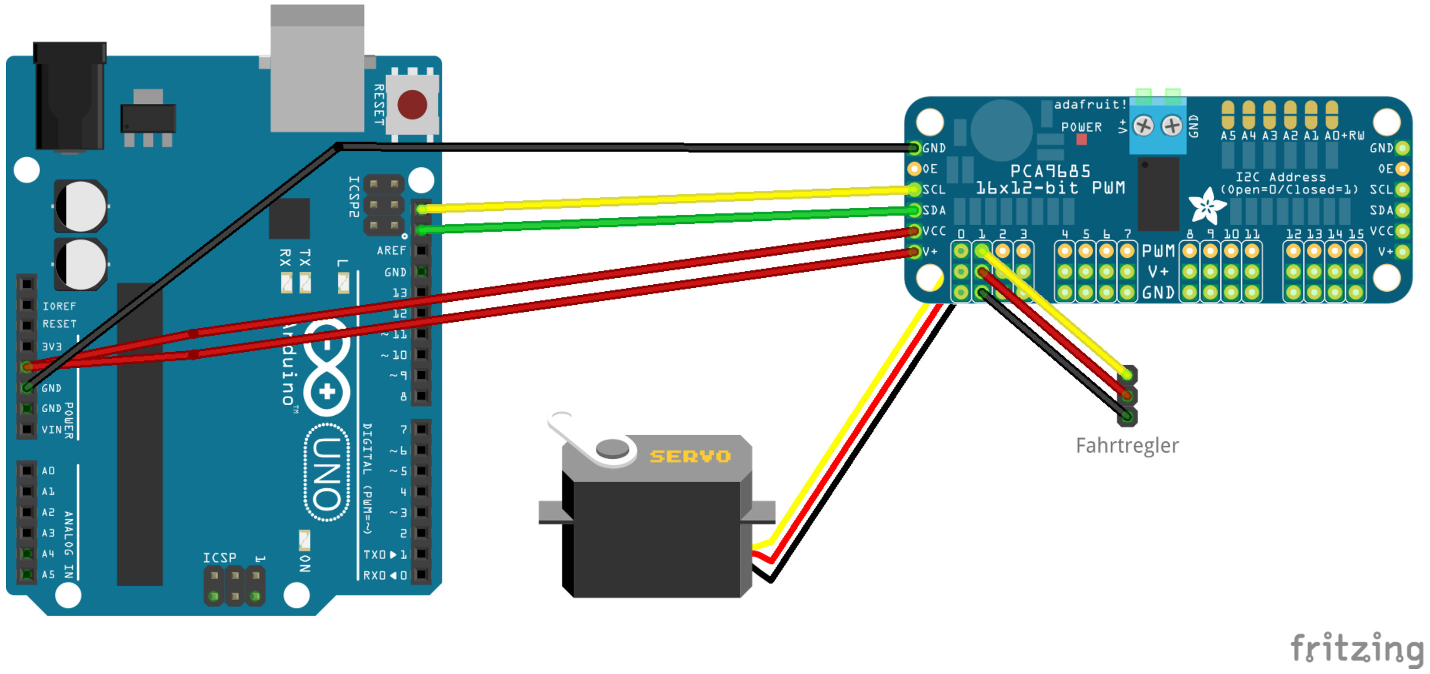


Connecting the PWM servo driver board

- Servo board connects by I^{2}C (*Inter Integrated Circuit*)
- Common alias for I^{2}C is TWI (*Two Wire Interface*)

Voltage supply:

- Pin VCC provides voltage for module, only
- Pin V+ provides voltage for servos (recommended to use connector at top, instead)



Exercise: Determine servo parameters

Write a program to determine the pulse lengths in ticks to control the servo:

- Neutral position
- Full right
- Full left

Data type and functions:

- *Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();*
- *pwm.begin ( ) ;*
- *pwm.setPWMFreq(frequencyHz) ;*
- *pwm.setPWM(servoChannel, startTick , endTick) ;*

Required include files:

- *Wire.h*
- *Adafruit_PWMServoDriver.h*

Exercise: Determine servo parameters

Sample solution (with method *loop()* being empty):

```
#define PWM_FREQUENCY_HZ 60     // Frequency of PWM signals
#define PCA_CHANNEL_SERVO 0     // Board PCA9685: servo channel
#define PWM_SERVO_CENTER 340    // Neutral position in 0..4095
#define PWM_SERVO_MAX_DELTA 120 // Max. +/- deviation from neutral position

void setup() {
  Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
  int minPulse = PWM_SERVO_CENTER - PWM_SERVO_MAX_DELTA;
  int maxPulse = PWM_SERVO_CENTER + PWM_SERVO_MAX_DELTA;

  pwm.begin();
  pwm.setPWMFreq(PWM_FREQUENCY_HZ);

  // Turn right (starting at neutral position)
  for (int pwmPulse = PWM_SERVO_CENTER; pwmPulse >= minPulse; pwmPulse--) {
    pwm.setPWM(PCA_CHANNEL_SERVO, 0, pwmPulse);
    delay(10);
  }

  // Turn left (starting at neutral position)
  for (int pwmPulse = PWM_SERVO_CENTER; pwmPulse <= maxPulse; pwmPulse++) {
    pwm.setPWM(PCA_CHANNEL_SERVO, 0, pwmPulse);
    delay(10);
  }

  // Stay in neutral position
  pwm.setPWM(PCA_CHANNEL_SERVO, 0, PWM_SERVO_CENTER);
}
```

Adapt parameters until observation matches expected behavior

Exercise: Break & reverse

Write a program with following functionality:

- Accelerate for 2.5 seconds from neutral to moderate forward drive
- Switch directly back to neutral
- Repeat, but switch directly to moderate backward speed, instead

Hints:

- Recall that the speed controller receives the same signal as a servo.
- Tick range: 220 (max. backward) to 460 (max. forward), neutral position at 340
- Function *millis()* returns milliseconds since start of program (as *unsigned long*)

Insights:

- Switch from forward to neutral $\Rightarrow$ Roll out
- Switch from forward to backward $\Rightarrow$ Break
- Needs to be in neutral shortly before driving backwards (for motor protection)

Exercise: Break & reverse

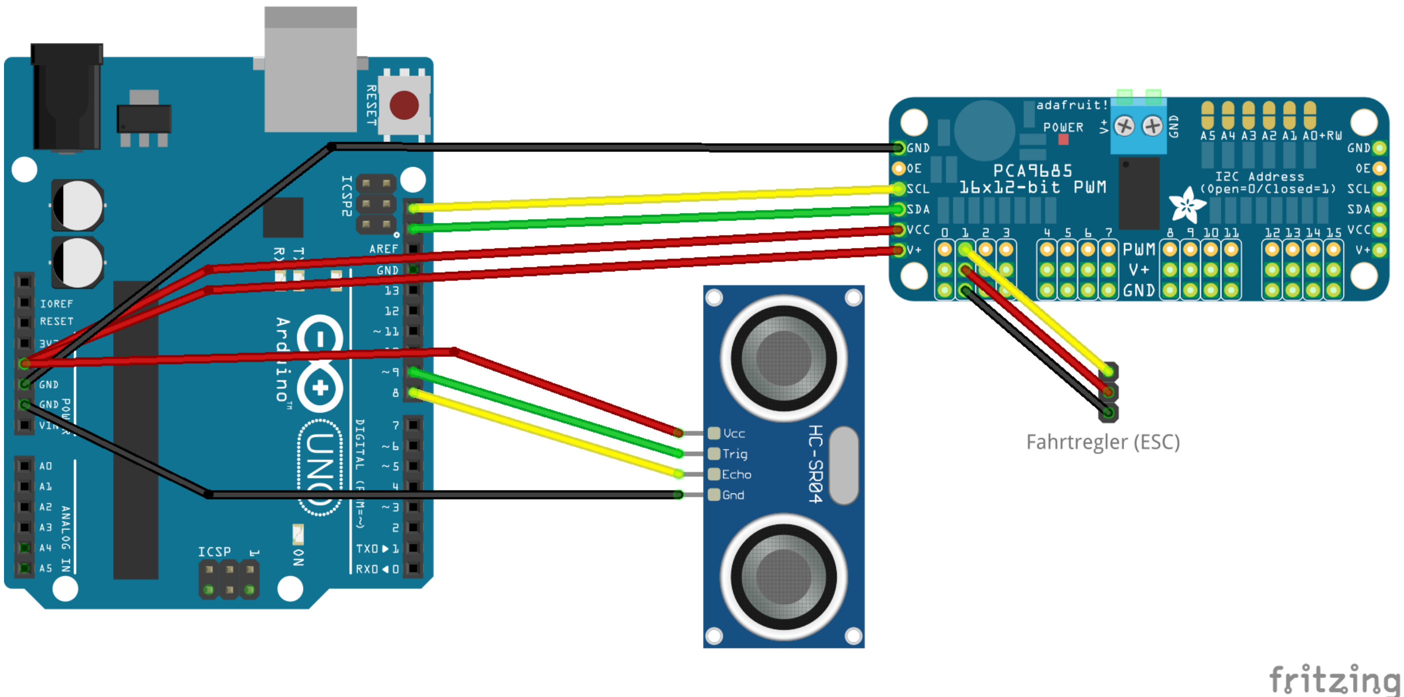
Sample solution:

```
#define PWM_CHANNEL_ESC 1
#define PWM_NEUTRAL 340
#define PWM_MAX 460
#define PWM_BREAK 240
#define ACCELERATION_TIME_MS 2500

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

void setup() {
  // Init servo driver
  int pwmLength = PWM_NEUTRAL;
  pwm.begin();
  pwm.setPWMFreq(60);
  pwm.setPWM(PWM_CHANNEL_ESC, 0, pwmLength);

  // Accelerate
  unsigned long startMillis = millis();
  while (millis() - startMillis < ACCELERATION_TIME_MS)
    if (pwmLength < PWM_MAX)
      pwm.setPWM(PWM_CHANNEL_ESC, 0, ++pwmLength);
    delay(100);
  }

  // Break
  pwmLength = PWM_BREAK;
  pwm.setPWM(PWM_CHANNEL_ESC, 0, pwmLength);
  delay(1000);

  // Neutral
  pwmLength = PWM_NEUTRAL;
  pwm.setPWM(PWM_CHANNEL_ESC, 0, pwmLength);
}
```

Exercise: Automatic stop

Write a program with following functionality:

- Set slow forward speed
- Repeatedly measure distance using a HC-SR04 module (see slide set “Fundamentals”)
- Stop when an object less than 1 m away from the car has been detected



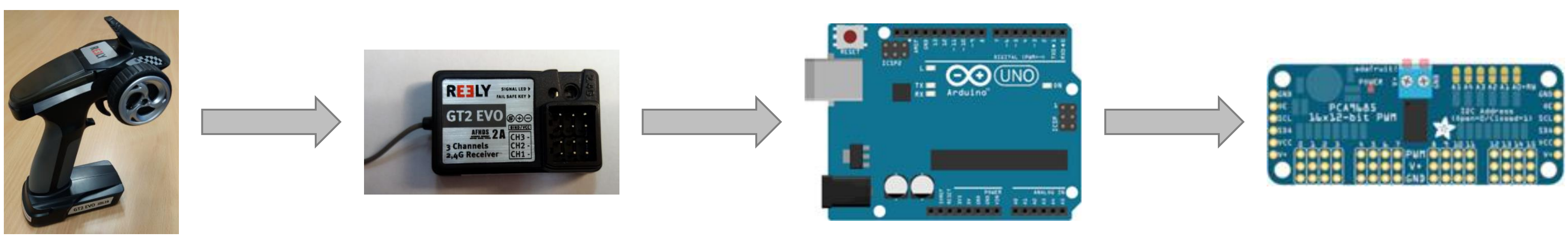
Combining manual & automatic control

- You read the receiver signals using the Arduino.
- You simultaneously use manual control and control by the Arduino.

Combining manual & automatic control

General idea:

1. Remote control sends human user input to receiver
2. Receiver generates PWM signals for servo and motor
3. Arduino measures pulse lengths
4. Arduino generates pulses for servo and motor:
   - Manual control: replicate receiver pulses
   - Automatic control: compute pulse length (e. g., limit maximum speed)



Exercise: Measure receiver pulses

Now it is your turn:

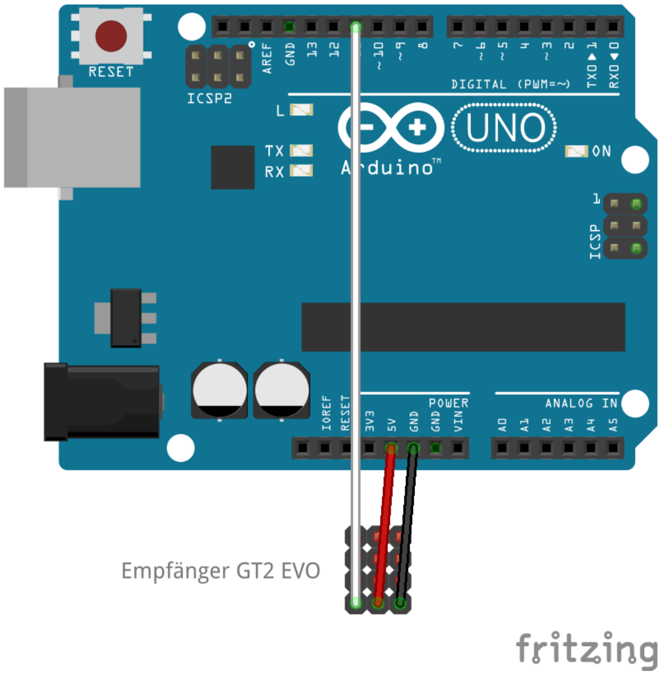
- Measure the pulse lengths and display them on the serial monitor.

Circuit:

- Connect receiver to the Arduino’s power pins
- Connect PWM line to a digital input pin (here: pin 11)

Reading the pulse length:

- *pinMode(pin, INPUT)* sets pin as input pin
- *puseIn(pin, HIGH)* returns length in ms



Exercise: Measure receiver pulses

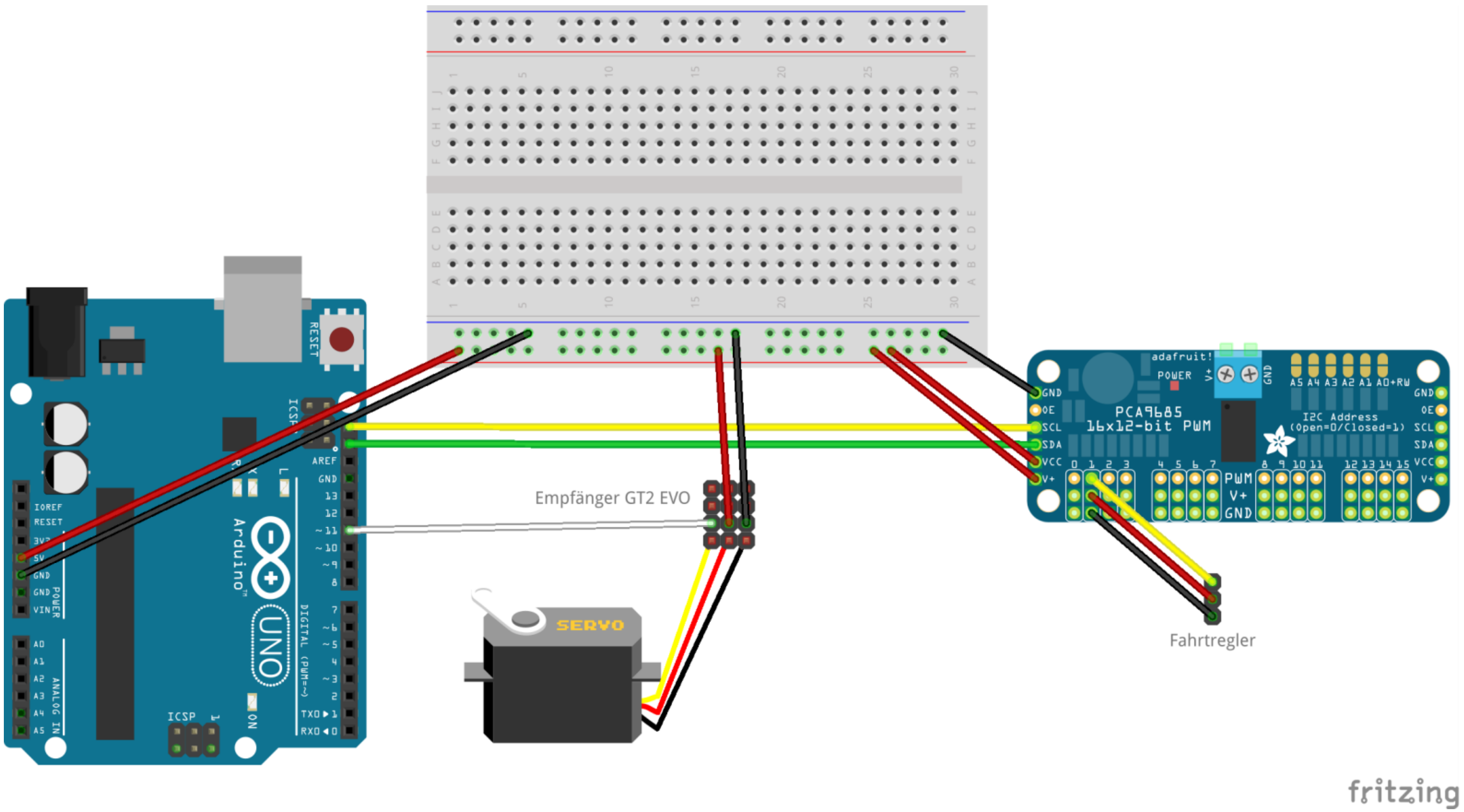
Sample solution:

```
#define PIN_RECEIVER_SERVO 11    // Receiver pwm pin
#define PWM_PERIOD 16666.7       // PWM period in microsecs (60 Hz)

unsigned long pulseSumMs = 0;
int pulseCount = 0;
double pulseMeanMs;

void setup() {
  Serial.begin(9600);
  pinMode(PIN_RECEIVER_SERVO, INPUT);
}

void loop() {
  // Measure pulse length of receiver PWM
  pulseSumMs += pulseIn(PIN_RECEIVER_SERVO, HIGH);

  // Display mean length (in ms and in 0 .. 4095 for servo board PCA9685)
  pulseMeanMs = pulseSumMs / ++pulseCount;
  Serial.print((int) pulseMeanMs);
  Serial.print(" ms (");
  Serial.print((int)((pulseMeanMs / PWM_PERIOD) * 4095));
  Serial.println(" in 4095)");
}
```

Exercise: Limit maximum speed

We are ready for the first real application:

- Let users control the car manually.
- But limit the maximum forward speed to a fixed value (e.g., 390 ticks).



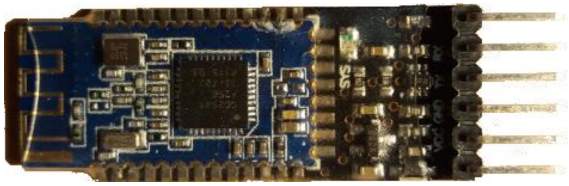
Exercise: Limit maximum speed

Sample solution:

```
#define PIN_RECEIVER_ESC 11    // Receiver pwm pin
#define PWM_PERIOD 16666.7     // PWM period in microsecs (60 Hz)
#define PCA_CHANNEL_ESC 1      // Board PCA9685: ESC channel
#define PWM_MIN_LIMIT 340      // Min. pwm pulse backward drive (0..4095)
#define PWM_MAX_LIMIT 380      // Max. pwm pulse forward drive (0..4095)

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

void setup() {
  pinMode(PIN_RECEIVER_ESC, INPUT);
  pwm.begin();
  pwm.setPWMFreq(1e6 / PWM_PERIOD);    // Frequency in Hz
}

void loop() {
  // Read pwm pulse from receiver
  int pwmPulseMicrosecs = pulseIn(PIN_RECEIVER_ESC, HIGH);
  int pwmPulseTicks = (int)((pwmPulseMicrosecs / PWM_PERIOD) * 4095);

  // Limit forward and backward speed
  if (pwmPulseTicks > PWM_MAX_LIMIT)
    pwmPulseTicks = PWM_MAX_LIMIT;
  else if (pwmPulseTicks < PWM_MIN_LIMIT)
    pwmPulseTicks = PWM_MIN_LIMIT;

  // Set pwm pulse for motor
  pwm.setPWM(PCA_CHANNEL_ESC, 0, pwmPulseTicks);
}
```

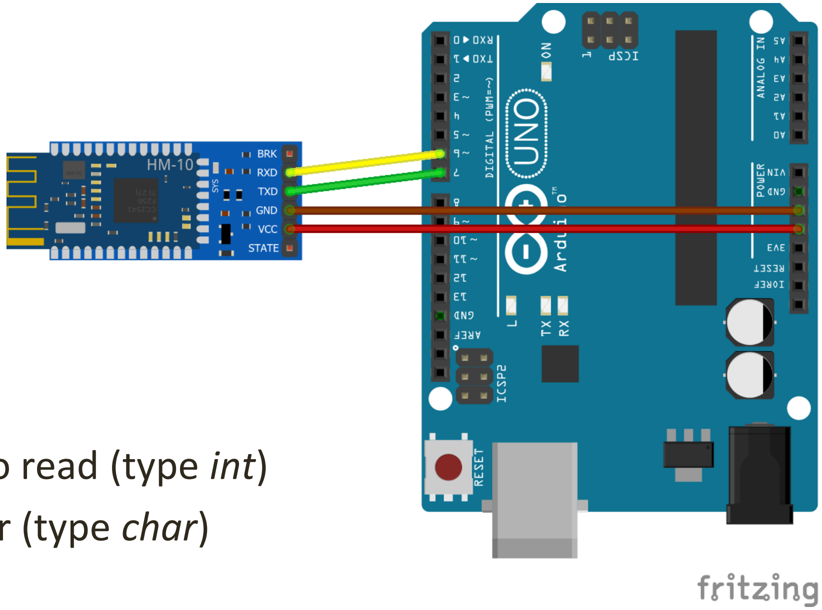
Control by mobile applications

- You send and receive data using a Bluetooth LE module.
- You modify the car's properties using an Android app.



Read & write via Bluetooth LE

- Bluetooth module connects over a 2-wire serial interface
- Like the serial monitor, e.g., using *SoftwareSerial.h*



Data types and initialization:

- *SoftwareSerial ble(RX pin, TX pin)*
- *ble.begin(9600)*
- You are free to choose the identifier *ble*

Read bytes (receive):

- *ble.available()* $\Rightarrow$ Number of bytes ready to read (type *int*)
- *ble.read()* $\Rightarrow$ Get next byte from buffer (type *char*)

Write data (send):

- *ble.println(…)*

Exercise: Read & write via Bluetooth LE

At first, we should get this running in principle:

- Install a Bluetooth terminal app on your smartphone.
- Send data from the app to the Arduino and display it in the serial monitor.
- Send data from the Arduino to the app.

Android app:

- Recommended (feel free to use another one): *Serial Bluetooth Terminal*
- https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal

App usage:

1. Open *Devices* in the menu
2. Select tab *Bluetooth LE* and scan for the HM-10 module
3. Open *Terminal* in the menu
4. Select the *Connect / Disconnect* icon on the top

Exercise: Read & write via Bluetooth LE

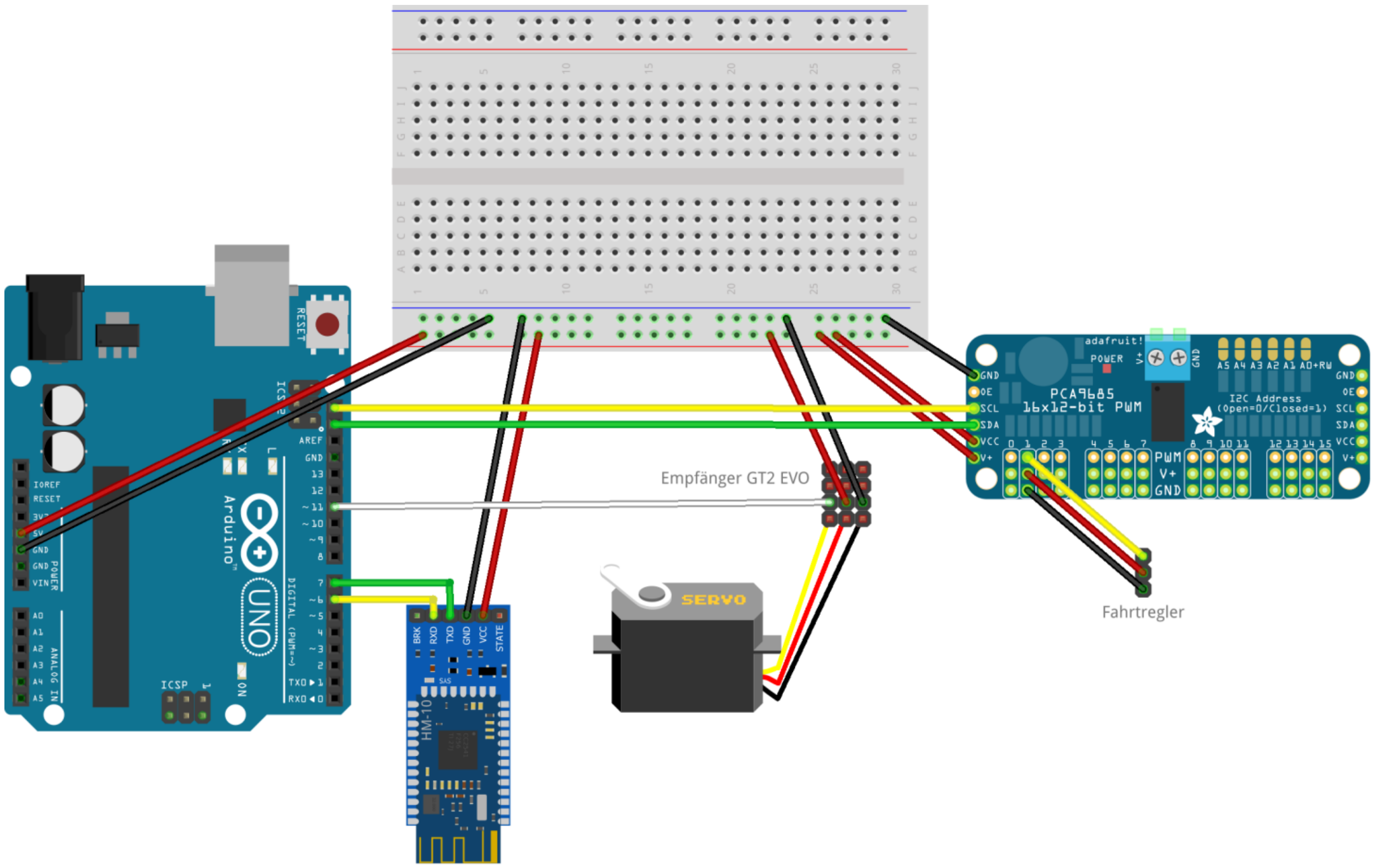
Sample solution:

```
#include <SoftwareSerial.h>

#define PIN_SOFT_TX 6        // Connect to HM-10's RX pin
#define PIN_SOFT_RX 7        // Connect to HM-10's TX pin

SoftwareSerial bleSerial(PIN_SOFT_RX, PIN_SOFT_TX);
String message = "";

void setup() {
  // Set up serial communication
  Serial.begin(9600);        // Serial monitor within Arduino IDE
  bleSerial.begin(9600);     // Serial connection to HM-10
  Serial.println("Connected to HM-10");
}

void loop() {
  // Check, if buffer overflow occurred
  if (bleSerial.overflow())
    Serial.println("HM-10: overflow");

  // Read and print data available over Bluetooth LE
  int bytesAvailable = bleSerial.available();

  while (bytesAvailable-- > 0) {
    char received = bleSerial.read();
    if (isDigit(received))              // Append digits to number string
      message += received;
    else if (message.length() > 0) {    // Non-digit => Number complete
      Serial.println(message.toInt());
      bleSerial.println("Value received: " + message);
      message = "";
    }
  }
}
```

Exercise: Limit maximum speed via Bluetooth LE

Let's improve our prior sample application:

- Set the maximum forward speed dynamically using a smartphone.



Exercise: Limit maximum speed via Bluetooth LE

Sample solution:

- Well, it is a lot of code … would cover many, many slides.
- How about looking at the source file provided, instead?