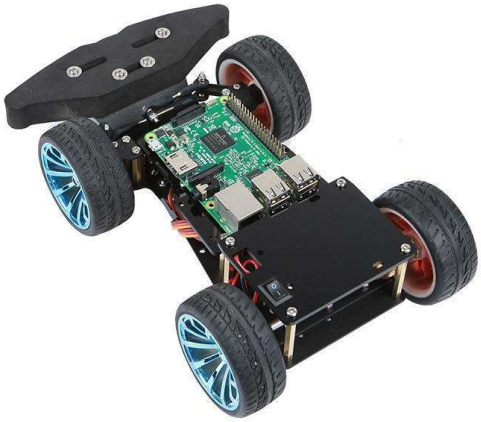
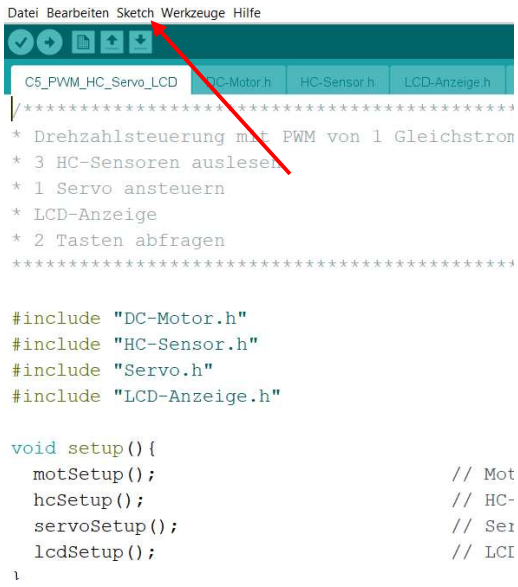
Kapitel C1 – RoboCar



Robot-Chassis nutzen
Kabelfarben und Pin-Belegungen
Software-Objekte kapseln
I2C – LCD Display verwenden
Ein Menü-System mit 2 Tasten generieren

Aufgabe C1.1:

- Erstelle eine Tabelle mit 3 Spalten, wie jedes Kabel zwischen Arduino und den Komponenten angeschlossen ist! Nutze die linke Spalte für die Komponente mit Anschlussbezeichnung, die mittlere Spalte für die Kabelfarbe und die rechte Spalte für den betreffenden Arduino-Pin.
- Die Belegungstabelle dient auch der GitHub-Dokumentation (Hardware).



Aufgabe C1.2:

- Wir verändern den HC-Sensor-Sketch aus Kapitel B8 um eine neue Datei, in die die Routine „gekapselt“ wird. Alle notwendigen Einstellungen werden neben dem Code hinüber kopiert. Erzeuge über *Sketch/Sketchordner anzeigen* eine neue Datei „HC-Sensor.h“ (Header). Der Sketch muss danach neu gestartet werden. Diese Datei wird mit `#include „HC-Sensor“` in das Hauptprogramm eingebunden. Im `void loop()` verbleibt lediglich `messHC()`. Die Variable `readHC` führt ähnlich `millis()` ständig den aktuellen Abstand zum Sensor.
- Bringe mit `Serial.print(readHC);` innerhalb einer kleinen `void` mit „Wecker“ alle 500ms den aktuellen Abstand zur Anzeige.
- Speichere den Skript unter C1_Robo_HC ab!
- Für unseren Robo benötigen wir diese Routine 3 mal. Aus messHC() wird messHcL(), messHcM(), messHcR() und analog readHcL, readHcM und readHcR.

Aufgabe C1.3:

- Für den Robo nutzen wir eine spezielle Bibliothek *LiquidCrystal_I2C*, die in der Datei „LCD-Anzeige.h“ zum Einsatz kommt. Die Verwendung entspricht Kapitel B6, nur dass nur 4 Leitungen (Serielle Datenverbindung) anzuschließen sind. Da diese Bibliothek intern delays verwendet, können wir die Anzeige während der Robo-Fahrt nicht nutzen. Sie beeinflusst den DC-Motor, den Servo und vor allem die HC-Sensoren.
  Die Anzeige von Messwerten oder dem Menü erfolgt wie bekannt über eine kleine Wecker-Routine.
- Zwei Tasten dienen der Robo-Steuerung: die linke wählt verschiedene Menüpunkte und die rechte Taste startet bzw. stoppt den Programmablauf. In der LCD-Datei initialisieren wir die Ports A2, A3 als D15, D16 als `INPUT_PULLUP`. Um die Tasten beim Betätigen zu entprellen, schaltet in einer kleinen Wecker-Routine nur die HL-Flanke!

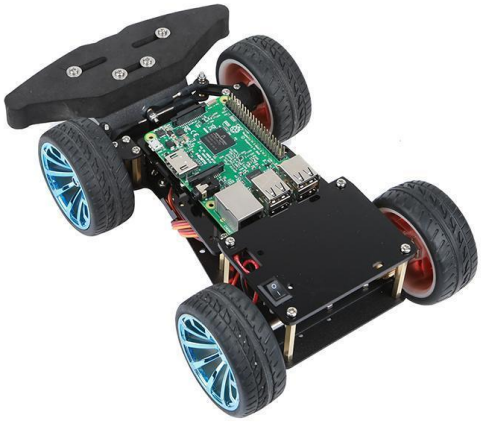
Aufgabe C1.4:

- Analog binden wir den DC-Motor aus Kapitel B7 Teil 2 und den Servo-Motor aus Kapitel B4 in unser Projekt ein. Alle sollten ein eigenständiges Objekt bilden.

Aufgabe C1.5:

- Erstelle eine Übersicht über die nutzbaren Funktionen mit ihren Attributen, um die Objekte DC-Motor und Servo-Motor zu steuern. Ergänze alle nutzbaren Variable. Beschreibe die Benutzung für den Robo, auch für die Dokumentation bei GitHub (Software).

Kapitel C2 – RoboCar Fahrregler



Robot-Chassis nutzen
Fahreigenschaften programmieren
Menü-System nutzen
PID-Regler

Info PID-Regler: Um dem RoboCar autonomes Fahren zu lernen, benötigt er Antrieb, Lenkung und Sensoren.
Antrieb und Lenkung nennt man in der Automatisierungstechnik „Stellglieder“. Aus Sensorinformationen wird fortlaufend eine Abweichung von einem Zielwert (Fehler) ermittelt, die vom Regler durch Änderungen an den Stellgliedern ausgeglichen werden soll. Es entsteht ein „Regelkreis“. Den Regler nennt man auch „Regelstrecke“.
PID-Regler bestehen aus maximal 3 Komponenten, die zusammenwirken: P-Proportionalregler, I-Integralregler und D-Differentialregler. Wie stark jeder Regler einwirkt wird über 3 Faktoren eingestellt: Kp, Ki, Kd und damit an jedes beliebige zu steuernde System angepasst.
P-Regler verstärken (>1) oder dämpfen (<1) über Kp den Fehler: P-Regler = Fehler * Kp
I-Regler addieren gedämpft (<1) Fehler der Vergangenheit auf: I-Regler = I-Regler + Fehler * Ki, sinngemäß tastet sich der Regler langsam an den Zielwert heran.
D-Regler reagieren auf plötzliche Änderungen des Fehlers im Vergleich zum letzten Wert. Er erzeugt eine sofortige Reaktion: D-Regler = Fehler * Kd.
Alle Regler gemeinsam wirken auf das Stellglied: Stellglied = P-Regler + I-Regler + D-Regler

Aufgabe C2.1:

- Das entscheidende Stellglied am RoboCar ist die Lenkung. Ermittle mit der vorgegebenen Testroutine (Listing 1) unter Menüpunkt 0 die einstellbaren Lenkwinkel! Wie lässt sich das Lenkverhalten links / rechts ausgleichen?
- Welchen Fehler können die HC-Sensoren ermitteln? Was ist der Zielwert?
  Simuliere in einer Teststrecke die Bewegung des RoboCars, checke alle möglichen auftretbaren Fälle und ermittle die Sensorwerte in einer Tabelle für Sensor Links, Mitte und Rechts! Skizziere die Lage des RoboCars mit Dreiecken!
- Berechne in einer 4. Spalte für jeden Fall den gewünschten Zielwert den Fehler! Was muss der Regler in einer 5. Spalte tun, um den Fehler auszugleichen? Auf welches Stellglied muss die Korrektur wirken?

Aufgabe C2.2:

- Innerhalb unseres Menü-Systems können wir PID-Regler mit verschiedenen K-Parametern vergleichsweise einbauen und testen. Suche eine optimale Konfiguration für die Rundfahrt des RoboCars, indem Kp, Ki, Kd optimiert wird!

```
static boolean lr=0;
  …
  switch(robotMenue){
    case 0: {
      if(robotGo){                    // Test der möglichen Lenkwinkel
        if(lr==0) lenkWinkel--; else lenkWinkel++;
        if(lenkWinkel==40) lr=1;
        if(lenkWinkel==120) lr=0;
      }
      break;
    }
```

```
  …
  case 1: {
    if(robotGo){
      fahrRegler();
    }
    break;
  }
  …

void fahrRegler(){
  int fehler = (readHcL+readHcR)/2-readHcL;      // Fehler = Soll - Ist
  int PRegler;                                   // Proportionalregler
  static int IRegler                             // Integralregler
  int DRegler;                                   // Differentialregler
  static int hcLOld, hcMOld, hcROld;  // Vergleichswerte vom letzten Lauf
  int deltaL=readHcL-hcLOld;
  int deltaM=readHcM-hcMOld;
  int deltaR=readHcR-hcROld;
      // PID Regler berechnen
  PRegler = fehler * 1.2;  // Kp
  IRegler = IRegler + ((readHcL+readHcR)/2-readHcL) * 0.2; // Ki
  DRegler = 0;
       // Linkskorrektur zur Mitte
  if(readHcL>readHcR) lenkWinkel=90 + PRegler + IRegler + DRegler;
  if (lenkWinkel<40) lenkWinkel=40;
       // Rechtskorrektur zur Mitte
  if(readHcL<readHcR) lenkWinkel=90 + 0.83*(PRegler + IRegler + DRegler);
  if (lenkWinkel>120) lenkWinkel=120;
       // Motor Halt
  if(readHcM<30) motorSoll=0; else motorSoll=100;
  hcLOld = readHcL;
  hcMOld = readHcM;
  hcROld = readHcR;        // Sensorwerte für nächsten Lauf merken
}
```