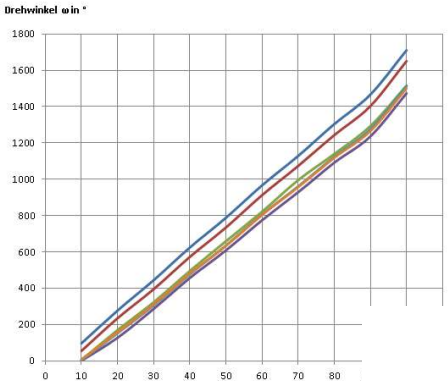
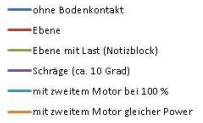
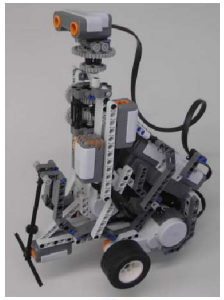
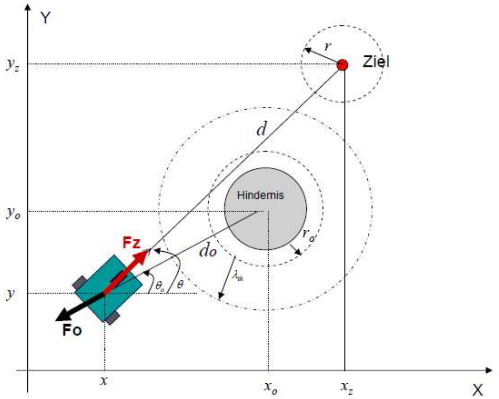
Einführung in die Robotik

Prof. Dr. Alexander Metzner

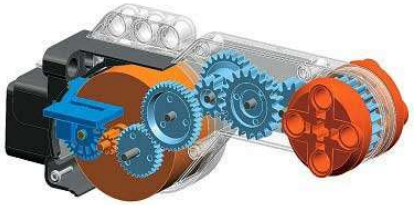


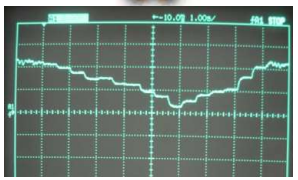


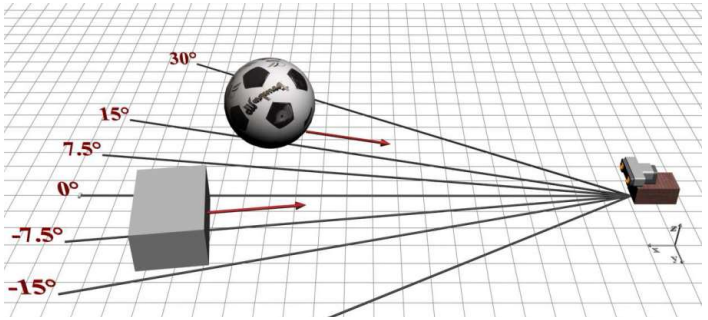





```
task main()
{

 int anzahl=4;
 int i=1;

 while (i<= anzahl)
 {
   OnFwdSync(OUT_AC, 50,0);
   Wait(2500);

   OnFwdSync(OUT_AC, 50,100);
   Wait(650);
   i = i + 1;
 }

 Off(OUT_AC);
}
```







Lernziel der Veranstaltung

- Siehe Modulhandbuch:

Angestrebte Lernergebnisse	*Die Studierenden kennen die Herausforderungen beim Entwurf von Systemen, die mit physikalischen Umgebungen interagieren.* *Die Studierenden kennen die Entwurfsprinzipien für diese Systeme und können diese realisieren.* *Die Studierenden kennen die Prinzipien HW-naher Systemprogrammierung und können diese realisieren.*
Inhalt	- *Sensorik/Aktorik in der Praxis* - *Wegeplanung und Umgebungsmodellbildung* - *Steuerung und Regelung* - *Abstraktionsschichten in HW-nahen Programmen*

- Weitere Herausforderung:
  - Systeme mit sehr eingeschränkten Ressourcen (Speicher, Rechenleistung, kaum Debugging-Möglichkeiten, …)

ORGANISATORISCHES

Wie erreichen wir das Ziel?

- Viele praktische Versuche (deswegen 1VL+3Ü!)
  - Vorlesung stellt die grundlegenden Konzepte vor
  - Im Praktikum setzen Sie diese um, **dokumentieren** und **demonstrieren** Ihre Lösungen
  - Programmiersprache: **C**
- Vorbereitung ist wichtig:

Arbeitsaufwand in Zeitstunden	*120h, davon 60h Präsenz und 60h Eigenstudium*

- Versuche am Demonstrator kosten **viel Zeit** und können **nicht** außerhalb des Labors durchgeführt werden. Nutzen Sie deshalb die Präsenzzeit ausschließlich dafür und bereiten Sie vor und nach!

Was sind die zu erbringenden Leistungen?

Phase 1 (2er-Gruppen)

- Aufgaben finden Sie auf den Übungszetteln
  - Lesen Sie die Aufgaben sorgfältig durch
  - Sprechen Sie mit mir, falls etwas unklar ist; viele Aufgaben akzeptiere ich nur in einer bestimmten Form der Implementierung
  - Lösen Sie die Aufgaben zeitnah!
- Zu (fast) jeder Aufgabe gibt es Demonstrationspflicht!

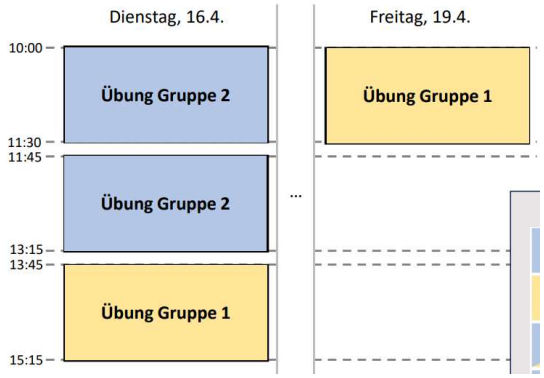
Phase 2 (1er-Gruppen)

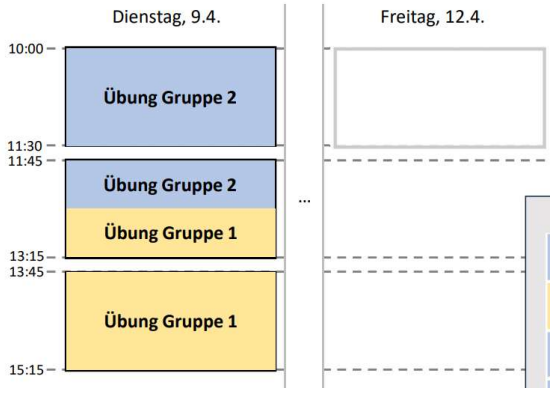
- Klausur

Termine

- Vorlesung nur asynchron als Video, bitte als Vorbereitung zu den Übungen zu Hause schauen
  - Vorteil: Sie bekommen neuen Vorlesungsstoff, sobald Sie den brauchen
- Übungszeiten werden wöchentlich gewechselt,

  abwechselnd 2 Übungen und 1,5 Übungen

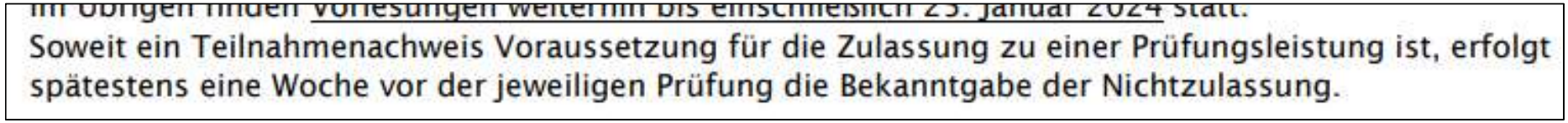




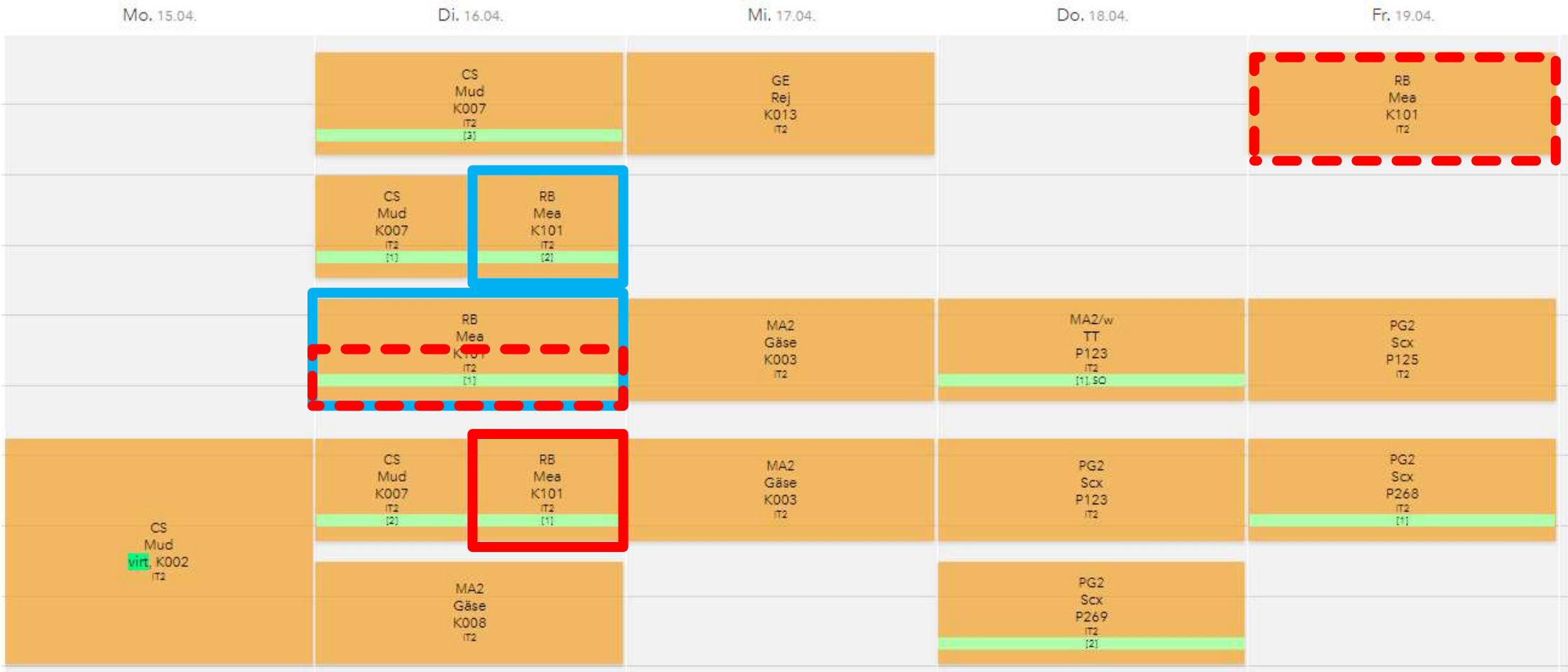
- **Übungsgruppen maximal 24 Personen**

Zulassungsvoraussetzungen

- Zulassungsvoraussetzung zur Klausur ist die Abgabe korrekter Lösungen für alle Aufgaben, die auf dem Übungszettel gekennzeichnet sind (100%-Anorderung!)
- Abgaben erfolgen im 4-6 Augengespräch
  - Ich akzeptiere **keine Abgaben per Email**!
  - Abgaben müssen korrekt sein, andernfalls muss eine Nachbearbeitung und erneute Vorlage erfolgen
- Geben Sie bitte zeitnah ab; das erleichtert uns allen die Organisation der Übungen und vermeidet lange Wartezeiten am Ende des Semesters
- Beachten Sie den Terminkalender der OTH insb. für die Meldung der Nichtzulassungen:


Im übrigen finden Vorlesungen weiterhin bis einschließlich 23. Januar 2024 statt.
Soweit ein Teilnahmenachweis Voraussetzung für die Zulassung zu einer Prüfungsleistung ist, erfolgt spätestens eine Woche vor der jeweiligen Prüfung die Bekanntgabe der Nichtzulassung.

Übungsgruppen und -Zeiten



Organisatorisches – Übung

- Übungen finden im Labor **K101** statt
  - Praktische Übungen mit realer Hardware
  - Für Vorbereitung zu Hause stehen verschiedene Simulatoren zur Verfügung, die Sie auf Privatrechner (Windows!) installieren können

- ACHTUNG: K101 ist kein „normales“ Labor:
  - Sie müssen eine Sicherheitsunterweisung durchführen
  - Die Sicherheitsunterweisung wird durch einen einfachen online-Test überprüft
  - Erst bei Bestehen des Tests dürfen Sie das Labor betreten
  - Erst bei Bestehen des Tests werden die Inhalte der VL im ELO-Kurs freigeschaltet
  - Bitte Zertifikat ausdrucken, unterschreiben und zur ersten Übung mitbringen

EINFÜHRUNG

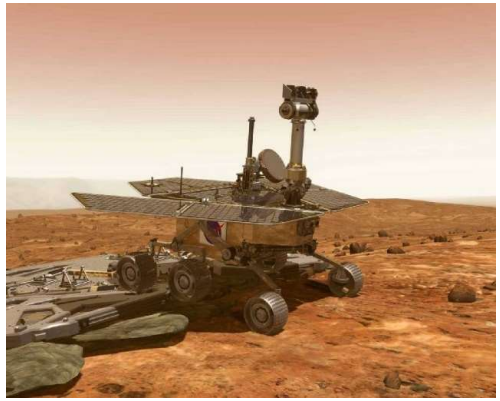
Robotik – Was ist das?

- Roboter [Hoppen]:

  „Maschine, die sich in einer natürlichen Umgebung aus eigener Kraft und ohne Hilfestellung von außen bewegen und dabei ein ihr gestelltes Ziel erreichen kann. […] Dabei erkennt sie die Umwelt, sofern dies notwendig ist, über eigene Sensoren.“

- Man unterscheidet:
  - Stationäre Roboter (z.B. Industrieroboter)
  - Mobile teilautonome Roboter (z.B. Drohnen)
  - Mobile autonome Roboter (z.B. Putzroboter)

- Beispiele:



Robotik – Was ist die Herausforderung?

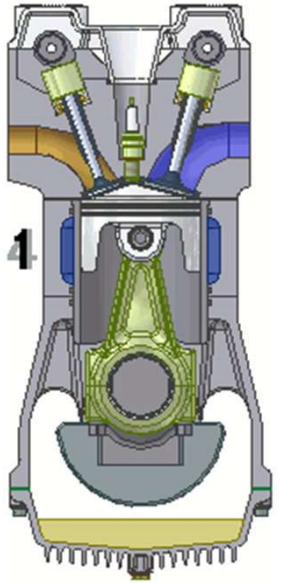
- Roboter [Hoppen]:

  „Maschine, die sich in einer natürlichen Umgebung aus eigener Kraft und ohne Hilfestellung von außen bewegen und dabei ein ihr gestelltes Ziel erreichen kann. […] Dabei erkennt sie die Umwelt, sofern dies notwendig ist, über eigene Sensoren.“

- D.h. Roboter interagieren mit der physikalischen Umgebung:
  - Nehmen den **Zustand der Umwelt** auf (**digitalisiert**!)
  - Verändern den Zustand der Welt, die auf **physikalischen Gesetzen** beruht
  - Autonom heißt: Der Mensch greift **nicht** steuernd in den Prozess der Informationsverarbeitung ein, sondern **definiert lediglich das Ziel**
- Ist grundsätzlich schwieriger, als Business-Anwendungen: Benutzern können Sie sagen „*falsche Eingabe, nochmal!*“. Sagen Sie dieses einmal dem Marsrover, nachdem er in einem Krater zerschellt ist!

Robotik – Warum Teil der TI?

- Technische Informatik beschäftigt sich zu einem großen Teil mit der Entwicklung von SW für technische Anlagen, sogenannte Steuergeräte:
  - Technische Anlagen sind physikalische Systeme
  - SW muss den Zustand der Anlage kennen
  - SW muss den Zustand der Anlage ändern
  - Fehlverhalten bedeutet oft Zerstörung der Anlage



➔Dies ist sehr ähnlich zu Robotern, vgl. oben

Roboter wecken aber unseren Spieltrieb und machen (meistens) viel Spaß ☺

Roboter – Der Demonstrator

- In der VL setzen wir die Lego Mindstorms NNXT ein
  - Einfach im Auf- und Umbau
    - Bekanntes Lego-Technik-Prinzip
  - Einfach in der Ansteuerung
    - Wir müssen uns an den meisten Stellen nicht darum kümmern, wie genau man die Werte aus der Umwelt bekommt (Inhalt der VL EM im 4. Semester)

➔Schnelle Lernerfolge

  - Trotz einfacher Aufbauten sind nette Versuche machbar
  - Man kann System anfassen, sie sind robust, bieten aber gleichzeitig all das, was ein Steuergerät auszeichnet

NNXT – Systemeigenschaften

Typisch für große Steuergeräte!

ARM-Cortex-M4 32-Bit Prozessor @168MHz
1 MB Flash-Speicher
192 KB RAM
USB-Anschluss

160x128 Pixel Farbdisplay
4 farbige LEDs
Piezo-Summer
SDCard-Slot
Bluetooth-Modul

Das ist Debugging-Luxus pur!

Meist nur wenige LEDs!!

Programmentwicklung I

- Programmentwicklung sind Herausforderung!
  - Wenig Flash (hier 1MB): Betriebssystem und Ihre Applikation muss hier hineinpassen
    - Schreiben sie kompakte Programme, d.h. entwickeln Sie möglichst Code-sparende Algorithmen
    - Vermeiden Sie implizite Dinge, wie Umwandlungen von Datentypen, Sie verlieren sonst schnell das Gefühl für die Größe des Codes
    - Vermeiden Sie ausladende Standard-Bibliotheken, sondern nehmen Sie lieber für den Prozessor spezifische Bibliotheken
    - Vermeiden Sie Code-Wiederholungen und verwenden stattdessen besser Funktionen (aber Vorsicht bei der Verschachtelungstiefe, siehe nächste Folie)

Programmentwicklung II

- Programmentwicklung sind Herausforderung!
  - Wenig RAM-Speicher (hier 192K): Betriebssystemdaten, Applikationsdaten und Laufzeitstack muss hier reinpassen:
    - Verwenden Sie nur die kleinste mögliche Dateneinheit für Variablen (Bits bei wenigen Bit, kleinstmögliche Integer). Wenn Sie nicht wissen, was der größte Wert einer Variablen ist, dann ist Ihr Programm schlecht geschrieben oder Sie haben das Problem nicht verstanden! (Achtung: Überläufe)
    - Verwenden Sie möglichst keine Gleitkommazahlen (müssen über SW emuliert werden)
    - Verwenden Sie niemals Rekursion (Laufzeitstack wird zu groß)
    - Übergeben Sie niemals Datenstrukturen, sondern immer nur Zeiger (Laufzeitstack wird zu groß)
    - Überprüfen Sie die Aufruftiefe von Funktionen/benutzen Sie ggf. lieber Makros per #define (Laufzeitstack wird zu groß)

Programmentwicklung III

- Programmentwicklung sind Herausforderung!
  - Kaum Debugging-Möglichkeiten
    - Nur in Ausnahmefällen lassen sich wenige Bits des Speichers anschauen (z.B. LEDs, manchmal spartanisches Display)
    - Fehlverhalten führt entweder zu kompletten Systemabsturz des ganzen Rechners, oder aber die SW tut einfach nicht das, was sie soll
    - Physikalische Prozesse spielen sich in der Regel im Bereich von Millisekunden ab, d.h. Fehlerdiagnose verlangt oft Oszilloskop oder Digital Analyzer (verlangt aufwändige Einrichtung, Messpunkte, etc.)
    - Programmteile laufen i.a. Nebenläufig, was Fehlersuche nochmals deutlich erschwert, weil nicht klar, was wann lief und ob es nicht Probleme bei der Nebenläufigkeit selbst gibt
    - **Testen ist extrem zeitaufwändig!**
  - **Es ist daher unbedingt notwendig, die SW vor der Implementierung genau zu planen und die Abläufe zu dokumentieren**
    - Nutzen Sie den Simulator → Tests außerhalb Laborzeiten möglich

Programmentwicklung IV

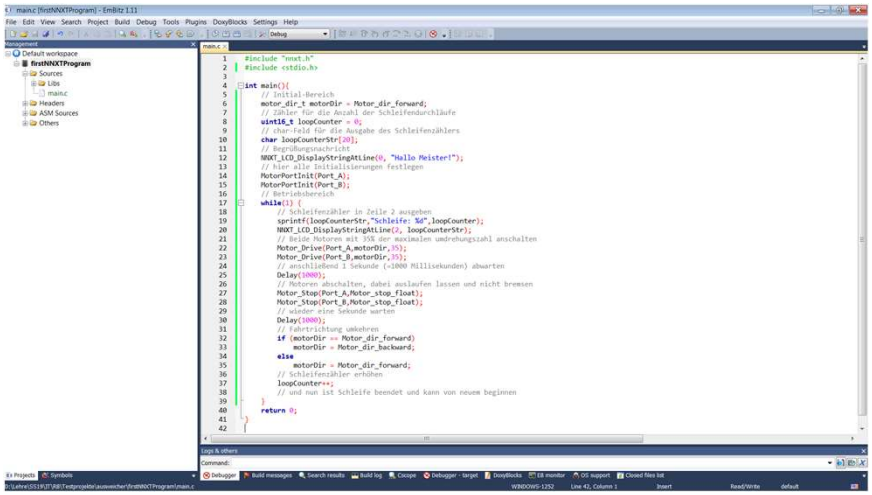
- Programmentwicklung sind Herausforderung!
  - Programme steuern physikalische Maschinen:
    - Machen Sie sich vorher ein Bild von der Physik des Systems: Was sind die Gesetze, welche Zustände kann das System einnehmen, was sind die Auswirkungen des Eingreifens
    - Abstrahieren Sie die Physik soweit wie nötig (Beispiel: Bouncing ball, Temperatur), aber nur so weit wie möglich
    - Machen Sie sich ein Bild von Ungenauigkeiten von Messungen (zeitliche Auflösung, Wandlungsfehler, Ungenauigkeiten der Sensoren)

  ➔Erst wenn das alles verstanden ist, lässt sich mit vernünftigem Aufwand ein funktionierender Algorithmus für ein Problem erstellen
    - Versuchen Sie nicht, kopflos funktionale Korrektheit in die SW zu testen; das kostet Sie meist viel zu viel Zeit (siehe vorige Folie)

Entwicklungsumgebung & -Sprache

- Programmierung des NNXT erfolgt in C
- NNXT-Library mit vielfältigen Funktionen
  - Erweiterte Möglichkeiten durch integriertes Betriebssystem
  - Einfache Ansteuerung von Peripherie (Sensoren, Motoren)
- Compiler ist ein gcc-Compiler für ARM-Prozessoren

- Entwicklungsumgebung EmBitz:
  - Überträgt übersetzte Programme zum NNXT
  - Ermöglicht Debugging auf NNXT
  - IDE mit viel Funktionalität (Autocompletion, …)



Zur Vorbereitung: der Simulator

- Zur Vorbereitung existiert ein einfacher Simulator
  - Code kann identisch für den Roboter genutzt werden
  - Implementierung jedoch in CodeBlocks, nicht Embiz
  - Programmierung in C++, aber bitte nutzen Sie nur C-Anteile von C++ für Ihr Programm
  - Installation und Bedienung siehe GRIPS-Kurs

Motoren des NNXT

- Maximal lassen sich 3 Motoren anschließen
- C-Funktionen zur Ansteuerung sind gegeben
  - Motoren an (Richtung, Antriebsstärke in %)
  - Motoren aus (gebremst [Achtung! Verbraucht Strom] oder „floating“)

```
Motor_Drive(Port_A, Motor_dir_forward, 50);  // Motor an Port A mit 50% vorwärts

Motor_Drive(Port_B, Motor_dir_backward, 75); // Motor an Port B mit 75% rückwärts

Motor_Stop(Port_A, Motor_stop_float);        // Motor an Port A stoppen, auslaufend

Motor_Stop(Port_B, Motor_stop_break);        // Motor an Port A stoppen, bremsend
```

Sensoren des NNXT

- 4 Sensoren können angeschlossen werden
  - Ultraschall-Sensor
  - Farbsensor
  - Touchsensor
  - Weitere kann man kaufen (Licht, Temperatur, Gyro, …)
- Pro Motor noch zusätzlich 1 Umdrehungssensor
- Weitere Sensoren/Aktoren kann man sich selbst basteln und über I^2C anschließen (Elektronik-Bastelerfahrung notwendig). Möglichkeiten:
  - IR-Entfernungssensoren
  - Lichtsensoren
  - Taster
  - LEDs als zusätzliche Anzeigen

Programmierung der Sensoren I

- Für Standard-Sensoren gibt es Funktionen
  - Setzen des Typs
  - Sensor-spezifische Funktionen zum Holen der Werte
- Andere Sensoren oder selbstgebaute Sensoren
  - Entweder analoge Werte lesen
  - Oder über „Netzwerk“ angeschlossene Geräte verwenden

- Siehe Dokumentation und Tutorial

Programmierung der Sensoren – Beispiel

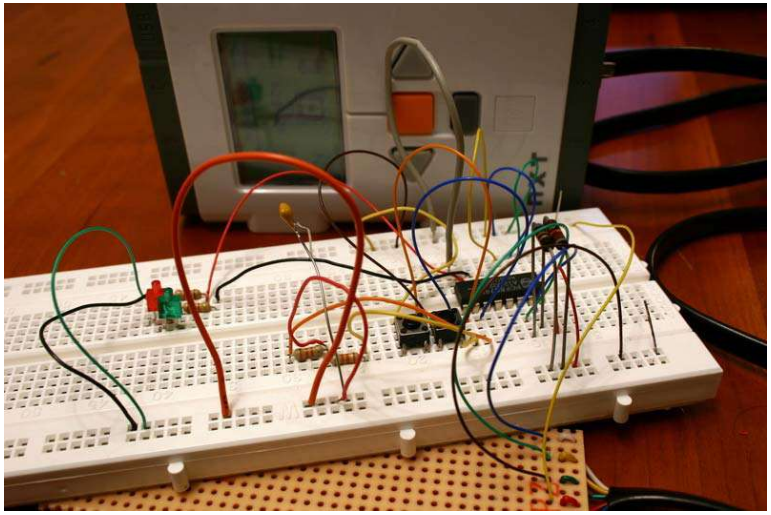
- Aufgabe: Wann immer ein Taster (angeschlossen an Eingang 1) gedrückt ist, soll „an“ im Display erscheinen, sonst nichts.
- Beobachtung: Programme auf dem Roboter sind i.a. nicht terminierend! → Endlosschleife

```
int main() {
    sensor_touch_clicked_t val;
    SensorConfig(Port_0,SensorTouch);
    while(1) {
        TouchClicked(Port_0, &val);
        if (val==SensorTouch_clicked)
            NNXT_LCD_DisplayStringAtLine(0,"an");
        else
            NNXT_LCD_DisplayStringAtLine(0," ");
    }
}
```

Frage: Wie muss der Code aussehen, wenn nur beim Runterdrücken etwas passieren soll?
Antwort: Übung!

Programmierung der Sensoren II

- Selbstgebastelte Sensoren werden nicht direkt unterstützt, sondern müssen über die I^2C-Schnittstelle bedient werden
- Für I^2C gibt es Bibliotheksfunktionen



Alles weitere …

… zum Thema der Programmierung der NNXT-Bausteine bzgl.

- Sensoren
- Motoren
- etc.

können Sie am besten in der Praxis erlernen!