Java 6, ftMscLib.dll, umFish50.dll & javaFish50.dll

ftComputing für Java

Java für Vergnügungssüchtige, VISTA sei Dank
C/C++ zu spießig, C#/VB.NET zu monopolistisch
Teil 2 : Ausgabe für den ROBO TX Controller
Alternativ mit Eclipse 3.4 und BlueJ 2.5

Ulrich Müller

Inhaltsverzeichnis

Das System

Allgemeines

Mit dem Package ftcomputing.roboTX.jar steht eine Möglichkeit zur Verfügung, den aktuellen fischertechnik **ROBO TX Controller** recht komfortabel zu programmieren. Ausgeliefert wird ein Original Workspace - Workspace34 - von Eclipse 3.4 (ohne .metadata), der neben dem oben genannten Package noch die erforderliche umFish50.DLL und die Wrapper.DLL javaFish50.DLL sowie diese Dokumentation enthält. Hinzu kommen noch eine Reihe von Beispielprojekten.

ftcomputing.roboTX.jar

Enthält als zentrale Klassen die Basis-Klasse JavaFish und die darauf aufsetzende Klasse FishFaceTX. Hinzu kommen enums für die verwendeten Parameter.

Die Klasse **JavaFish** kapselt zusammen mit der JNI-Wrapper.DLL javaFish50.DLL die Funktionen der zentralen umFish50.DLL und reicht sie weitgehend 1:1 an die Anwendung weiter. JavaFish ist primär Basis für FishFaceTX oder eigene Kreationen.

Achtung : JNI erwartet hier zwingend den Klassennamen JavaFish. Bei eigenen Erweiterungen also lieber eine neue Klasse ableiten anstatt schnell mal ein bischen zu ändern.

Die Klasse **FishFaceTX** enthält neben den verbesserten Methoden von JavaFish (Unterbrechbarkeit, Abbrechbarkeit) noch eine Reihe weiterer Methoden (meist Wait…), die im Programm ein stilvolles Warten auf den Abschluß der durch z.B. setMotor angestoßenen Operationen ermöglichen.

Die Beispiele sind sehr einfach gehalten, sie setzen nur ein Interface mit ein paar angebauten Teilen (Lampen, Motoren, Taster, Sensoren) voraus, aber kein bestimmtes Modell.

Install

JDK 1.6 : Install mit Setup-Programm, soweit nicht bereits geschehen. Ein älteres JDK wird's sicher auch tun.

Eclipse 3.4 : Entpacken ZIP, Desktopsymbol einrichten. Hier wird auch eine Versionsnummer kleiner gehen.

Von fischertechnik.de Computing Downloads : PC Programming ROBO-TXC 1.1 downloaden. Die **ftMscLib.DLL** aus dem Paket nach Windows\System32 kopieren (auch wenn schon ROBO Pro installiert ist, die dort benutzte liegt im Verzeichnis \Programme. RoboTXTest bereitstellen für den ersten Modelltest.

umFish50.DLL und **javaFish50.DLL** nach Windows\System32 kopieren

Eclipse : Workspace anlegen

Im Workspace Verzeichnis **ftComputing** anlegen, dort ftcomputing.robo.jar (mit JavaFish) ablegen. Beispielprogramme kopieren.

Eclipse : Projekt anlegen

Eclipse in Java Perspektive mit Standard Einstellungen.
Arbeiten im Package Explorer : Kontext-Menu über RechtsClick

1. **New** | JavaProject
   Project Name : NeuesProjekt | Finish
2. **NeuesProjekt** | New | Package | Name : neuesProjekt
3. **Package** neuesProjekt aufklappen | New | Class
   Name : MainFish, x Checkbox : public static void main …
4. **NeuesProjekt** | Build Path | Add External Archives
   Verzeichnis ftComputing im Workspace suchen und ftcomputing.roboTX.jar wählen
5. In class MainFish (im Editor) hinter package neuesProjekt
   import ftcomputing.roboTX.*; einfügen

Weiter im Editor mit Programmdetails für reine Console- bzw. einfache Swing-Programme

Eclipse : jar erstellen und nutzen

Eclipse in Java Perspektive mit Standard Einstellungen.
Arbeiten im Package Explorer : Kontext-Menu über RechtsClick

Erstellen einer Bibliothek, die in einem Projekt genutzt werden kann :

1. **Projekt** im Package Explorer markieren und aufklappen
2. RechtsClick auf Projekt | Export | **JAR** File
   Next : Gewünschtes Package (oder ganzes Projekt) markieren
   Bei einem Package sind die .files nicht erforderlich
   Zielpfad eingeben | Finish

Erstellen einer selbständig ablaufenden Anwendung (geht so erst mit Eclipse 3.4) :

1. RechtsClick auf Projekt | Export | Runnable JAR File
2. Next | Launch Configuration | Projekt wählen
3. Export destination (Name und Pfad des JAR Files) festlegen | Finish

Achtung : System.out.println Ausgabe werden nicht angezeigt, also eher Swing-Projekte einsetzen.

JavaFish

JavaFish Methoden

Die Klasse JavaFish ist Bestandteil des Package ftcomputing.roboTX.jar. Für die Parameter werden num Werte verwendet, die in der Zählung mit 0 beginnen : z.B. I1 wird dann zu Parameterwert 0…) Drehrichtung : 0 Off, 1 Links, 2 Rechts. Speed / Power im Bereich 0 - 512. Das hat sich gegenüber ftcomputing.robo ein wenig geändert.

Die Funktionen ist public native, eine Instanzierung ist erforderlich.

Die Rückgabewerte der Funktionen enthalten neben einem "Nutzwert" immer auch einen möglichen jxError-Code über den der Verlauf der Operation abgefragt werden kann. mit iState sind Wahrheitswerte gemeint : 0 = false, jxError = Fehler, alles andere gleich true.

Verbindungsaufbau TX Controller

jxOpenController : Herstellen einer Verbindung zum TX Controller

jxCloseController : Schließen einer bestehenden Verbindung zum TX Controller

Universaleingänge

Auslesen des aktuellen Eingangswertes, beim ersten Zugriff wird der Anschluß auf die dem Zugriffsbefehl entsprechende Eingangsart fliegend konfiguriert. Ab da gilt dann diese Konfigurierung und die weiteren Zugriffe laufen dann deutlich schneller

jxGetAnalog : A5K-Eingang (NTC, Photowiderstand, Potentiometer)

jxGetDistance : Dist-Eingang (UltraschallSensor)

jxGetInput : D5K-Eingang (Taster, Reedkontakt, PhotoTransistor)

jxGetTrack : D10V-Eingang (SpurSensor)

jxGetVoltage : A10V-Eingang (FarbSensor, Spannung allgem.)

Countereingänge

jxGetCounter : Auslesen des aktuellen Counterstandes

jxClearCounter : Zurücksetzen des Counters

M / O - Ausgänge

jxSetLamp : Schalten eines O-Ausganges

jxSetMotor : Schalten eines M-Ausganges

jxStartMotor : Starten eines "Encoder"-Motors für die angegebene Impulszahl, der Motor wird bei Erreichen des Ziels asynchron abgeschaltet.

jxIsMotorReady : Abfrage, ob Motorziel erreicht ist (läuft noch : -1, 0 und größer abgeschaltet mit Angabe der zusätzlich gefahrenen Impulse)

Systemfunktionen

sleep :	Warten für MilliSekunden
escape :	Abfrage, ob ESC-Taste gedrückt wurde
getTickCount :	Verstrichene Zeit in Ticks seit Mitternacht.

Console Programme mit JavaFish

Generell gilt hier, daß die Klasse JavaFish eher Basis für eigene, darauf aufsetzende Klassen ist, als eine, die im normalen Betrieb eingesetzt werden soll. Dafür ist die Klasse FishFaceTX vorgesehen.

Hier ein einfacher **HelloFish** Blinker :

```
import ftcomputing.roboTX.*;
public class MainProbe {
  public static void main(String[] args) {
    MainProbe mp = new MainProbe();
    System.out.println("Hallo Meister");
    mp.Action();
  }

  private void Action(){
    JavaFish ft = new JavaFish();
    ft.jxOpenController(4);
    ft.jxSetMotor(0, 0, 1, 444);
    JavaFish.sleep(1234);
    ft.jxSetMotor(0, 0, 0, 0);
    do{
      ft.jxSetLamp(0, 4, 512);
      ft.jxSetLamp(0, 5, 0);
      JavaFish.sleep(444);
      ft.jxSetLamp(0, 4, 0);
      ft.jxSetLamp(0, 5, 512);
      JavaFish.sleep(333);
    } while((ft.jxGetInput(0,2) == 0) && (JavaFish.escape() == 0));
    ft.jxSetLamp(0, 4, 0);
    ft.jxSetLamp(0, 5, 0);
    ft.jxCloseController();
    System.out.println("Ja, Meister, woll!");
  }
}
```

`import ftcomputing.roboTX.*;`
Verweis auf die Klasse JavaFish mit den Zugriffsfunktionen zum TX Controller. `JavaFish ft = new JavaFish();`
Instanzieren.

`ft.jxOpenController(4);`
Herstellen einer Verbindung zum ROBO TX Controller an COM4 (USB-Anschluß). Ggf. anpassen. Auf eine Fehlerabfragen wurde verzichtet. Gleichzeitig kann immer nur ein TX Controller (Einstellung Main) an USB bzw. Bluetooth betrieben werden

Das eigentliche (Nutz)Programm :
- Den Motor an Main, M1 mit Power 444 nach links drehen, 1234 mSek Pause, aus
- Wechselblinken mit Lampen an O5 und O6
- Endlosschleife bis Taster an I3 true oder ESC-Taste
- Lampen aus.

`ft.jxCloseController();`
Schließen der Verbindung zum TX Controller.

FishFaceTX

Allgemeines

Parameter für die Methoden sind häufig enums :
Dir (Off, Left, Right) Allgemeines Schalten der Eingänge, dabei gilt Left auch für On von O-Ausgängen
Ctr (Main, Ext1… Ext8) Main steht für den direkt über USB angeschlossenen Controller, die Ext. Controller sind an Main mit Flachband-Kabeln angeschlossen. Die Controller müssen über ihr lokales Display entsprechend konfiguriert werden.
Unv (I1…I8, C1 .. C4) Universaleingänge, C1 - C4 nur bei der Methode getInput, wenn sie als D5K-Eingang genutzt werden.
Cnt (C1 … C4) Counter-Eingänge vom EncoderMotor und Impulstaster
Mot (M1 … M4) Motor-Ausgänge zweipolig geschaltet
Out (O1 … O8) Einpolige Ausgänge (plus Masse).

Umwandlung eines enums in ein int : Methode ordinal() (Mot.M1.ordinal() ergibt 0)
Umwandlung eines int in einen enum-Namen : Methode values()
z.B. Mot.M1 ergibt sich aus Mot.values()[0]

Mit Ausnahme von Version, open/closeController verlangen alle Methoden eine intakte Verbindung zum Controller. Andernfalls wird die allgemeine **Exception** : FishFaceException ausgelöst.

Einige (langlaufende) Methoden sind **unterbrechbar** (Thread.yield) um ein Update der Oberfläche zu ermöglichen und **abbrechbar** ESC-Taste, NotHalt um in Crash-Situationen das Programm schnell beenden zu können.

Methoden

Zusätzlich zu dem angegebenen Parameter für Ein- oder Ausgang kann bei Bedarf noch die Controllerbezeichnung (enum Ctr) angegeben werden, ohne Angabe wird Ctr.Main angenommen.

static String **Version**()
Auslesen der FishFace Version

boolean **getNotHalt**() / **setNotHalt**(boolean OnOff)
Auslesen / Setzen NotHaltwunsch

void **openController**(String ComName)
Herstellen einer Verbindung zum TX Controller (USB / Bluetooth)
Der ComName ist rechnerspez. und kann mit dem Tool Robo_TX_Test ermittelt werden.

void **closeController**()
Beenden einer Verbindung zum TX Controller

boolean **finish**([Unv inpNr])
Feststellen eines Abbruchwunsches (ESC-Taste, NotHalt, Digitaleingang)

void **pause**(int mSek)
Anhalten des Programms (Threads) um mSek

int **getAnalog**(Unv inpNr)
Auslesen eines Analogwertes (A5K-Eingang : NTC, Photowiderstand, Potentiometer)

int **getDistance**(Unv inpNr)
Auslesen des aktuellen Wertes in cm des Ultraschallsensors

boolean **getInput**(Unv inpNr)
Auslesen des aktuellen Wertes eines digitalen Einganges (D5K-Eingang : Taster, Reedkontakt, PhotoTransistor)

boolean **getTrack**(Unv inpNr)
D10V-Eingang : Auslesen des aktuellen Wertes eines Sensors des SpurSensors (on Track (schwarz) = true)

int **getVoltage**(Unv inpNr)
Auslesen des aktuellen Spannungswertes (A10V-Eingang : Farbsensor, Spannung allgem.)

int **getCounter**(Cnt cntNr)
Auslesen des Zählers für den angegebenen Zähler-Eingang C1…

clearCounter(Cnt cntNr)
Zurücksetzen des Zählers für den angegebenen Zähler-Eingang

setLamp(Out outNr, int OnOff)

Setzen eines O-Ausganges (Out.O1… , OnOff 0 - 512)
OnOff = 0 aus, 1 - 512 ein mit angegebener Power

void **setMotor**(Mot motNr, Dir direction)
void **speedMotor**(Mot motNr, Dir direction, int speed)
void **startMotor**(Mot motNr, Dir direction, int speed, iCount)
void **startRobMotor**(Mot motNr, Dir direction int speed iCount)

Setzen eines M-Ausganges (Mot.M1…, Dir.Left.., speed 0 - 512). Bei angeschlossenem ImpulsCounter (an C-Eingang mit gleicher Nr, EncoderMotor oder "normaler" + Impulstaster) auch Setzen der zu drehenden Impulse. Die Methoden beenden sich dann selbst (asynchron), das Ende kann aber auch mit WaitForMotor bzw. WaitForMotors abgewartet werden. Bei startRobMotor wird zusätzlich ein zugeordneter Endtaster (I-Eingang mit gleicher Nr) ausgewertet.

int **waitForMotor**(Mot motNr)
Warten auf das Ende des mit startMotor / startRobMotor gestartetem Motors. Der Rückgabewert enthält eine evtl. Differenz zur Impulsvorgabe.

void **waitForMotors**(Mot… motorNrs)
Warten auf das "Fertigwerden" mehrerer mit startMotor / startRobMotor gestarteter Motoren.

void **waitForInput**(Unv inpNr, boolean OnOff)
void **waitForHigh**(Unv inpNr)
void **waitForLow**(Unv inpNr)
Warten auf einen digitalen Wert am I-Eingang (Taster, Photowiderstand ..)
Bei Input auf den vorgegebenen Wahrheitswert, bei High auf einen false / true Durchgang und bei Low auf einen true / false Durchgang

FishFaceTX Beispiel TXaFace (Console Programm)

```
import ftcomputing.roboTX.*;
public class TXmain {
   FishFaceTX tx = new FishFaceTX();
   final int FullPower = 512;
   final int Off= 0;

   public static void main(String[] args) {
      TXmain mf = new TXmain();
      System.out.println("TXmain gestartet");
      System.out.println("FishFace-Version : " +
                         FishFaceTX.Version());
      try { mf.Action(333); }
      catch(FishFaceException eft) { System.out.println(eft);}
      finally { System.out.println("Finito");      }
   }
   private void Action(int Dauer) {
       tx.openController("COM4");
       int Runde = 0;
       tx.startMotor(Mot.M1, Dir.Left, 444, 12);
       tx.waitForMotor(Mot.M1);
       do {
           System.out.println("Runde    : " + ++Runde);
           tx.setLamp(Out.O5, FullPower);
           tx.pause(Dauer);
           tx.setLamp(Out.O6, FullPower);
           tx.pause(Dauer);
           tx.setLamp(Out.O7, FullPower);
           tx.pause(Dauer);
           tx.setLamp(Out.O5, Off);
           tx.setLamp(Out.O6, Off);
           tx.setLamp(Out.O7, Off);
           tx.pause(Dauer * 2);
       } while(!tx.finish());
       tx.closeController();
       System.out.println("--- FINITO ---");
   }
}
```

Das eigentliche Nutzprogramm Action wird durch try / catch gekapselt, eventuelle Fehler werden auf der Console angezeigt.

In Action wird mit openController zunächst eine Verbindung zu einem ROBO TX hergestellt.

Dann der Motor an M1 ein wenig und asynchron für 12 Impulse mit halber Kraft gescheucht.

Ist das überstanden, wird in Action heftig geblinkt : Lampen an O5 - O7. Zusätzlich wird mit einer Photozelle noch die Raumhelligkeit und mit einem FarbSensor Farbe gezeigt.Das geschieht in einer Endlosschleife, die nur durch die ESC-Taste wieder abgebrochen werden kann.

FishFaceTX Beispiel GFaceTX (Swing Programm)

```
import java.awt.*;
import javax.swing.*;
import ftcomputing.roboTX.*;
public class gFaceTX {

  JFrame frmMain    = new JFrame("Hello FishFace");
  JLabel lblStatus  = new JLabel("Kontrolle Interface",
                          JLabel.CENTER);
  JLabel lblUltra   = new JLabel(" ", JLabel.CENTER);
  JLabel lblWider   = new JLabel(" ", JLabel.CENTER);
  JLabel lblHinweis = new JLabel(" ", JLabel.CENTER);
  FishFaceTX tx = new FishFaceTX();

public gFaceTX() {
   frmMain.setLayout(new GridLayout(0, 1, 12, 24));
   frmMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frmMain.setSize(300, 200);
   lblStatus.setForeground(Color.BLUE);
   frmMain.add(lblStatus);
   frmMain.add(lblUltra);
   frmMain.add(lblWider);
   frmMain.add(lblHinweis);
   frmMain.setVisible(true);
   }

public static void main(String[] args) {
   gFaceTX gf = new gFaceTX();
   gf.Action();
}

private void Action() {
  try {
    tx.openController("COM4");
    lblHinweis.setText("Ende : ESC-Taste oder Taster I3");
    int Runde = 0;
    do {
      lblStatus.setText("Runde : " + (++Runde));
       lblWider.setText("Photowiderstand I6 : " +
                           tx.getAnalog(Unv.I6));
       lblUltra.setText("FarbSensor I4      : " +
                         tx.getVoltage(Unv.I4));
       Blinken(333);
    } while (!tx.finish(Unv.I3));
  }
  catch (FishFaceException eft) {
          lblStatus.setText(eft.toString());}
  finally {
    tx.closeController();
    lblHinweis.setText("Finito : X-Drücken");
  }
}
```

```
private void Blinken(int Dauer) {
   tx.setLamp(Out.O5, 512);
   tx.pause(Dauer);
   tx.setLamp(Out.O6, 512);
   tx.pause(Dauer);
   tx.setLamp(Out.O7, 512);
   tx.pause(Dauer);
   tx.setLamp(Out.O5, 0);
   tx.setLamp(Out.O6, 0);
   tx.setLamp(Out.O7, 0);
   tx.pause(Dauer);
}
```

Hinzugekommen ist einiges Swing, darauf wird hier nicht eingegangen. Die eigentliche Funktion läuft wieder in Action in einer do-Schleife. Das Blinken wurde in ein Unterprogramm Blinken ausgelagert. Dafür werden direkt in der Schleife einige Analogwerte angezeigt.

FishFaceTX : Elemente - Erproben der Methoden

Zusammengefaßt in der Klasse Elemente. Die Klasse ist Teil eines **BlueJ** - Projektes. In BlueJ-Umgebung können deswegen die Methoden einzeln erprobt werden.

Es wird von keinem besonderen Modell ausgegangen sondern einem einheitlichen Versuchsaufbau :

```
* Motor M1 mit Endtaster I1 und Impulstaster C1Z (setMotor,   Mot)
* Motor M2 mit Endtaster I2 und Impulstaster C2Z (getCounter, Cnt)
* Taster an I3                                   (getInput,   Unv)
* Farbsensor an I4                               (getVoltage, Unv)
* Phototransistor an I5                          (getInput,   Unv)
* Photowiderstand an I6                          (getAnalog,  Unv)
* Spursensor an I7 links und I8 rechts           (getTrack,   Unv)
* Lampen an O5, O6, O7                           (setLamp,    Out)
```

Die Klasse hat einen kleinen Konstruktor

```
public class Elemente {
    private FishFaceTX tx;
    public Elemente(){
        tx = new FishFaceTX();
        tx.openController("COM4");
    }
```

zur Erzeugung des FishFaceTX Objekts und zum Herstellen einer Verbindung mit dem TX-Contoller. Hier an COM4. Der auf dem eigenen Rechner zutreffende Port wird am besten mit RoboTX-Test ermittelt.

Test der Ausgänge

```
    public void outTest() {
        final int fullPower = 512;
        final int noPower = 0;
        System.out.println("outTest gestartet");
        for(int i = 0; i < 7; i++) {
            Out o = Out.values()[i];
            System.out.println("Out." + o);
            tx.setLamp(o, fullPower);
            tx.waitForHigh(Unv.I3);
            tx.setLamp(o, noPower);
        }
        System.out.println("--- Finito ---");
    }
```

Die Ausgänge O1 - O8 werden nacheinander eingeschalten und nach Betätigen des Tasters an I3 wieder ausgeschaltet. Wenn Motoren zweipolig (z.B. an M1) angeschlossen sind, drehen sie nacheinander in wechselnden Richtungen.

Test Farbsensor

```
    public void farbSensor(Unv f) {
        System.out.println("FarbSensor an : " + f);
        do {
            System.out.println("Farbwert : " + tx.getVoltage(f));
            tx.waitForHigh(Unv.I3);
        } while(!tx.finish());
        System.out.println("--- Finito ---");
    }
```

In einer Endlosschleife (Beenden ESC-Taste) wird der aktuelle Farbwert angezeigt, nach Betätigen des I3-Tasters der nächste.

Anzeige von wählbaren Sensorwerten

```
public void sensoren(int i) {
    Unv s = Unv.values()[i-1];
    do {
       switch(i) {
            case 4:
                System.out.println("FarbSensor : " +
                                   tx.getVoltage(s));
                break;
            case 5:
                System.out.println("Phototransistor : " +
                                    tx.getInput(s));
                break;
            case 6:
                System.out.println("Photowiderstand : " +
                                    tx.getAnalog(s));
                break;
            case 7:
                System.out.println("SpurSensor : " +
                                    tx.getTrack(s));
                break;
        }
        tx.waitForHigh(Unv.I3);
    } while(!tx.finish());
    System.out.println("--- Finito ---");
}
```

Bei Aufruf der Methode kann der interessierende Sensor ausgewählt werden : FarbSensor, Phototransistor, Photowiderstand und Spursensor (hier nur eine Spur). Nach betätigen des I3-Taster wird der dann aktuelle Wert des gleichen Sensors angezeigt. Mit Unv.values()[i-1] wird die übergebene Sensornummer in den entsprechenden enum-Wert konvertiert.

Motor links/rechts zwischen zwei Endtastern

```
public void motPendel() {
    System.out.println("Motor M1 links bis I1 und rechts bis C1");
    do{
        tx.setMotor(Mot.M1, Dir.Left);
        tx.waitForInput(Unv.I1, true);
        tx.setMotor(Mot.M1, Dir.Off);
        tx.pause(1234);
        tx.setMotor(Mot.M1, Dir.Right);
        tx.waitForInput(Unv.C1, true);
        tx.setMotor(Mot.M1, Dir.Off);
        tx.pause(1234);
    } while(!tx.finish());
```

```
    System.out.println("--- Finito ---");
}
```

Der Motor an M1 wird abwechselnd bei Erreichen des zugehörenden Endtasters links (I1) und rechts (C1) geschaltet.

Motor für eine vorgegebene Zeit einschalten

```
public void motorTime(int mSek, int speed) {
    System.out.println("Motor gestartet");
    tx.speedMotor(Mot.M1, Dir.Left, speed);
    tx.pause(mSek);
    tx.setMotor(Mot.M1, Dir.Off);
    System.out.println("--- Finito ---");
}
```

Motor an M1 wird linksdrehend mit vorgebbarer Geschwindigkeit für die angegebene Zeit (in Millisekunden eingeschaltet.

Motor für eine vorgegebene Anzahl von Impulsen einschalten

```
public void motorCount(int iCount) {
    System.out.println("Motor gestartet");
    tx.startMotor(Mot.M1, Dir.Left, 512, iCount);
    System.out.println("Finito, Diff : " + tx.waitForMotor(Mot.M1));
}
```

Über startMotor wird der Motor an M1 mit einem Impulstaster an C1 gestartet. Er schaltet sich nach Erreichen der vorgegebenen Impulszahl selbstätig wieder ab. mit waitForMotor wird lediglich darauf gewartet.

Die Motoren M1 und M2 fahren simultan auf Home und Position

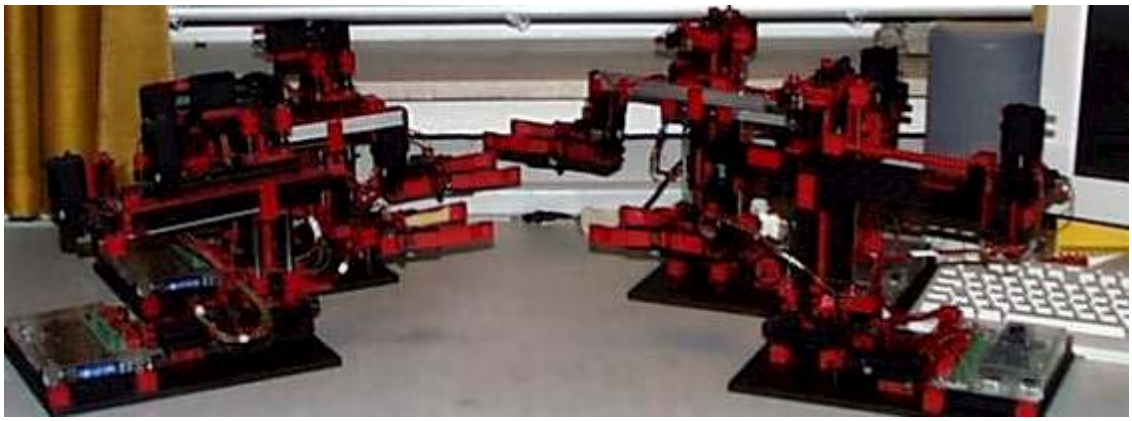
```
public void motorRob(int iCount1, int iCount2) {
   System.out.println("Motor Home, wartet auf I1 und I2");
   tx.startRobMotor(Mot.M1, Dir.Left, 512, 9999);
   tx.startRobMotor(Mot.M2, Dir.Left, 512, 9999);
   tx.waitForMotors(Mot.M1, Mot.M2);
   System.out.println("Motor M1 to : " + iCount1);
   System.out.println("Motor M2 to : " + iCount2);
   tx.startRobMotor(Mot.M1, Dir.Right, 400, iCount1);
   tx.startRobMotor(Mot.M2, Dir.Right, 512, iCount2);
   tx.waitForMotors(Mot.M1, Mot.M2);
   System.out.println("--- Finito ---");
}
```

Die Motoren M1 (Endtaster I1, Impulstaster C1) und M2(I2, C2) fahren zunächst simultan linksdrehend auf ihre Homeposition am zugehörenden Endtaster. Sie werden dort automatisch abgeschaltet. Der vorgegebene Counterwert wurde mit 9999 so groß gewählt, dass er außerhalb des Bewegungsraumes liegt, er dient nur dazu den Motor bis zum Erreichen des Endtasters in Betrieb zu halten.

Anschließend werden beide Motoren auf rechtsdrehend für die beim Start der Methode vorgegebene Impulszahl geschaltet.

In beiden Fällen werden die Motoren selbstätig bei Erreichen des Ziels abgeschaltet. Die vorhandenen waitForMotors Methoden dienen der Synchronisation mit dem Programm.

Über den Umgang mit den Industry Robots



Aufbau der Robots nach Anleitung : Säulenmotor an M1 mit Endtaster an I1 und Impulstaster an C1, Arm waagerecht an M1, I2, C2 und Arm senkrecht an M3, I3, C3. Greifer an M4, I4, C4 kommt später.

Klasse RobTXtut : Säule fährt Home und dann nach rechts

```
import ftcomputing.roboTX.*;
public class RobTXtut {
    private FishFaceTX tx;
    private int[] actPos = new int[4];
    public RobTXtut() {
        tx = new FishFaceTX();
        tx.openController("COM4");
    }
```

Import des genutzten Packages, in der Klasse die globale Variable tx mit dem FishFaceTX-Objekt. Im Konstruktor wird tx instanziiert. Zusätzlich wird eine Verbindung zum Robo TX Controller an USB - hier über COM4 (auf eigenen Gegebenheiten anpassen) - hergestellt.

Säule fährt linksdrehend an ihre Home-Position

```
public void goHome() {
    System.out.println("Motor M1 Home, wartet auf I1");
    tx.setMotor(Mot.M1, Dir.Left);
    tx.waitForInput(Unv.I1, true);
    tx.setMotor(Mot.M1, Dir.Off);
    System.out.println("--- Zu Hause ---");
}
```

System… zur Anzeige der aktuellen Aktivität auf der Konsole.
setMotor : Start des Säulenmotors
waitForInput : Warten auf das Erreichen der Home-Position
setMotor Off : Abschalten des Säulenmotors

Säule dreht rechts um die angegebene Schrittzahl

```
public void goRight(int Inc) {
   System.out.println("Säule dreht rechts um : " + Inc);
   tx.startRobMotor(Mot.M1, Dir.Right, 512, Inc);
   tx.waitForMotors(Mot.M1);
   System.out.println("--- Säule angekommen ---");
}
```

startRobMotor : Die Säule wird um Inc Impulse nach rechts (fullSpeed - 512) gedreht. Die Methode ist asynchron. Der Säulenmotor wird bei Erreichen der vorgegebenen Impulszahl automatisch abgeschaltet. Da hier nichts weiter zu tun ist, folgt mit waitForMotors eine Methode, die auf das Ende der vorher angestossenen Operation wartet.

Testen mit BlueJ

Auf das Klassensymbol von RobTXtut rechtsklicken und ein neues Objekt anlegen. Darauf klicken und die interessierende Methode aufrufen. Achtung : Nach Änderungen in der Source muß das BlueJ vorher noch zurückgesetzt werden (Rechtsklick auf das RUN-Symbol).

Alternativ können die interessierenden Befehle oder Methoden auch im Direkteingabe-Fenster eingegeben werden. Als erster Befehl muss dann eine RobTXtut = new RobTXtut(); Instanz angelegt werden.

Testen mit BlueJ : Methode main()

```
public class RobMain {
  public static void main() {
     RobTXtut rob = new RobTXtut();
     System.out.println("Robot gestartet");
     rob.goHome();
     rob.goRight(22);
     rob.goHome();
     rob.goRight(56);
     System.out.println("--- Das war's ---");
  }
```

Eine zweite Klasse mit der statischen Methode main() anlegen und in ihr die interessierenden Befehle unterbringen.

RobTXzwei : Der gesamte Robot wird gescheucht

```
import ftcomputing.roboTX.*;
public class RobTXzwei {
    private FishFaceTX tx;
    private int[] actPos = new int[4];
    static final int FullSpeed = 512;

    public RobTXzwei() {
        tx = new FishFaceTX();
    }

    public void start() {
        tx.openController("COM4");
    }
    public void ende() {
        tx.closeController();
    }
```

Es wird ein Array (actPos) angelegt, der die aktuelle Position der Robot-Komponenten in Impulsen ab Home (Endtaster) enthält. Die Verbindung zum Controller wird jetzt in einer eigenen Methode hergestellt. Hinzu kommt eine weitere Methode für das Beenden der Verbindung. Macht zwar im Testbetrieb zunächst etwas mehr Arbeit, erleichtert sie aber auch.

Anfahren der Home-Position

```
public void driveHome(Mot motNr) {
   System.out.println("Motor " + motNr + " -> Home");
   tx.setMotor(motNr, Dir.Left);
   tx.waitForInput(Unv.values()[motNr.ordinal()], true);
   tx.setMotor(motNr, Dir.Off);
   actPos[motNr.ordinal()] = 0;
   System.out.println(motNr + "--- Zu Hause ---");
}
```

Angefahren wird die Home-Position einer Komponente. Der zugehörende Motor wird als Term der Enumeration Mot übergeben. Die eigentliche Home-Funktion entspricht dem schon bekannten goHome. Hinzugekommen ist noch ein Null-Setzen der aktuellen Position.

Der Einsatz von Enumarationen entspricht zwar der Methodik von Java, macht aber auch manchmal zusätzliche Arbeit : Bei waitForInput muß der zugehörende Endtaster ermittelt werden (Mot.M1 -> Unv.I1, Mot.M2 -> Unv.I2 ..) das geschieht hier erstmal durch konvertieren des enum-Motornamens in eine Ordnungsnummer (0 - 3 für M1 - M4) über motNr.ordinal(). Anschliessend wird die so gewonnene Nummer wieder in einen Namen verwandelt : Unv.values()[motNr.ordinal()]. actPos ist ein Array der Länge 4 mit den Elementen 0 - 3, er wird auch über motNr.ordinal() adressiert.

Anfahren einer vorgegebenen Position - Impulse ab Home

```
public void driveTo(Mot motNr, int pos) {
  System.out.println("Motor " + motNr + " fährt auf Position : "
                     + pos);
  if (pos < actPos[motNr.ordinal()])
      tx.startRobMotor(motNr, Dir.Left, FullSpeed,
      actPos[motNr.ordinal()] - pos);
  else if (pos > actPos[motNr.ordinal()])
           tx.startRobMotor(motNr, Dir.Right, FullSpeed,
           pos - actPos[motNr.ordinal()]);
  tx.waitForMotors(motNr);
  actPos[motNr.ordinal()] = pos;
  System.out.println("Auf Position : " + pos);
}
```

Die Methode ist komfortabler als die bekannte goRight, aber auch komplizierter. Mit dem if-Konstrukt wird zunächst die erforderliche Drehrichtung bestimmt. Nach links, wenn die neue Position kleiner ist als die aktuelle, nach rechts, wenn sie größer ist. Gar nicht, wenn Zielposition und aktuelle Position übereinstimmen. In Verbindung mit der Drehrichtungsbestimmung wird dann der zugehörenden Fahrbefehl wien in goRight aufgerufen. Die zu fahrende Impulszahl wird als Differenz zwischen Soll und Ist übergeben. Anschließend wird wieder auf das Ende der Operation gewartet und dann actPos upgedatet.

Testen mit main()

```
import ftcomputing.roboTX.*;
public class RobTXmainZwei {
  static RobTXzwei rob = new RobTXzwei();
  static boolean COMon = false;
  public static void main() {
    System.out.println("Robot gestartet");
    if(!COMon) { rob.start(); COMon = true;}
      HinUndHer();
      //AlleDrei();
```

```
      //RobGo();
      //RobListe(new int[][]
      //         {{55, 21, 33},{10, 22, 22},{56, 11, 11}});
      rob.ende();
      System.out.println("--- Das war's ---");
  }
  private static void HinUndHer() {
    rob.driveHome(Mot.M1);
    rob.driveTo(Mot.M1, 22);
    rob.driveTo(Mot.M1, 11);
    rob.driveTo(Mot.M1, 56);
  }
```

Das main() ist etwas aufwändiger geworden. static : main() muß eine static Methode sein um startbar zu sein. Wenn main() auf globale Elemente zugreifen will, müssen diese auch static sein. Das gilt für Objekt rob und die Methoden der Klasse.
In main() wird hier mit rob.start(); die Verbindung zum TX Controller hergestellt umd mit rob.ende(); wieder aufgehoben. Dazwischen gibt es nur einen Aufruf der jeweils genutzten Methode. Hier HinUndHer().
In HinUndHer() wird zunächst das Home der Säule angefahren und dann wird die Säule ein wenig hin und her gedreht.

Da main beim Wiederholten Aufruf zickt (BlueJ IDE) wurde die Statusvariable COMon eingeführt, die das unterbindet. Ist getrickst, war aber das Einfachste.

Alternativ kann man natürlich auch weiterhin mit den direkten Testmethoden von BlueJ arbeiten (s.o.). Unter Menü Ansicht des Hauptfensters von BlueJ kann man zusätzlich ein Debugger-Fenster einschalten.

Und nun : AlleDrei

```
private static void AlleDrei() {
  for(int i = 0; i < 3; i++) rob.driveHome(Mot.values()[i]);
  rob.driveTo(Mot.M1, 56);
  rob.driveTo(Mot.M2, 22);
  rob.driveTo(Mot.M3, 33);
}
```

Zunächst werden die Home-Positionen aller beteiligten Komponenten nacheinander angefahren. Anschließend wird eine Komponente nach der anderen in gewünschte Position gebracht.

Simultan : Alles auf einmal

```
public void moveHome() {
  System.out.println("Es geht nach Hause");
  for(int i = 0; i < 3; i++)
      tx.startRobMotor(Mot.values()[i], Dir.Left, FullSpeed, 9999);
  tx.waitForMotors(Mot.M1, Mot.M2, Mot.M3);
  for(int i = 0; i < 3; i++) actPos[i] = 0;
  System.out.println("--- wir sind zu Hause ---");
}
```

Das Anfahren Home mit allen Komponenten gleichzeitig ist eigentlich recht simpel. Zunächst werden die beteiligten Motoren in Richtung links für 9999 Impulse geschaltet. Hier wird eine Eigenheit von startRobMotor genutzt : Der Motor wird abgeschaltet, wenn der zugehörende Endtaster erreicht wird - 9999 wird nie erreicht. Anschließend wird in Ruhe auf das Ende der Operation gewartet. waitForMotors hat hier eine feste Liste aller beteiligten Motoren als Parameter. Schluß : actPos = 0;

```
public void moveTo(int... pos) {
  System.out.println("Robot fährt auf neue Position");
  for(int i = 0; i < pos.length; i++) {
      if(pos[i] < actPos[i])
         tx.startRobMotor(Mot.values()[i], Dir.Left, FullSpeed,
         actPos[i] - pos[i]);
```

```
        else if(pos[i] > actPos[i])
                tx.startRobMotor(Mot.values()[i], Dir.Right,
                FullSpeed, pos[i] - actPos[i]);
    }
    tx.waitForMotors(Mot.M1, Mot.M2, Mot.M3);
    for(int i = 0; i < pos.length; i++) actPos[i] = pos[i];
    System.out.println("--- Mir san doa ---");
}
```

Auch hier Start aller beteiligten Motoren in einer for-Schleife. Hier wird aber nach links / rechts unterschieden wie schon in driveTo. Der Parameter int… pos von moveTo sthet für variable Anzahl von Argumenten. Es müssen also nur die Motoren von "vorne (ab M1)" angegeben werden, deren Position sich ändert. Gewartet wird aber auf alle Beteiligten.

Test : RobGo - RobListe

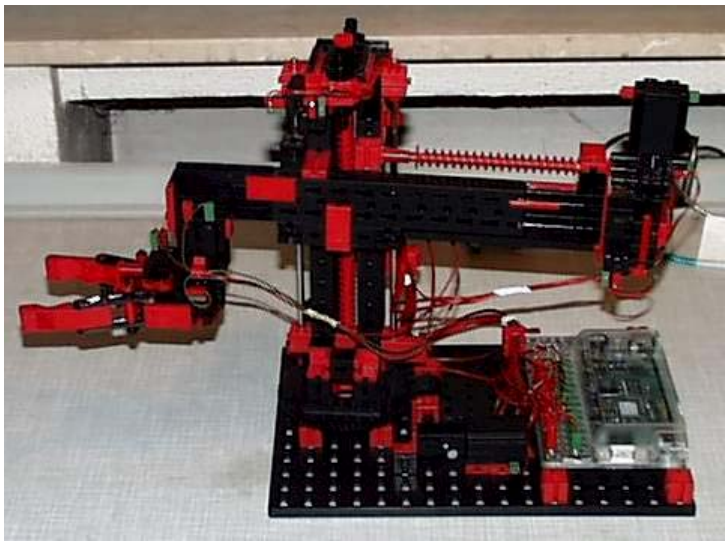
```
private static void RobGo() {
  rob.moveHome();
  rob.moveTo(56, 33, 22);
  rob.moveTo(50, 11, 16);
}
```

Ist eigentlich schon ein wenig mickrig.

```
private static void RobListe(int[][] posListe) {
  rob.moveHome();
  for(int[] pos : posListe) rob.moveTo(pos);
}
```

Deswegen nochmal, aber listengesteuert. Die Liste mit den Positionen wird als Parameter übergeben. Der Methodenrumpf ist eigentlich auch ein bischen wenig : moveHome() wie gehabt und dann eine Schleife über die Positionen. Damit auch Spaß macht eine "foreach"-Schleife in der die Werte einer Robot-Position abgearbeitet werden. An moveTo wird deswegen auch nur ein Array mit den Positionswerten übergeben, das kann er ab.

Klasse RobTXir2 : Betrieb des Säulenrobots



Ist primär eine Überarbeitung der Klasse RobTXzwei ohne deren Struktur deutlich zu verändern.

Import, Klass und Konstruktor

```
import ftcomputing.roboTX.*;
public class RobTXir2 {
    private FishFaceTX tx;
    private int[] actPos = new int[3];
    private int gripOpen;

    static final int FullSpeed = 512;
    static final int GripClosed = 10;
    static final Mot[] RobMot = new Mot[]{Mot.M1, Mot.M2, Mot.M3};
    static final Mot GripMot = Mot.M4;

    public RobTXir2() {
        tx = new FishFaceTX();
    }
    public void start() {
        tx.openController("COM4");
        for(int pos : actPos) pos = 0;
        gripOpen = 0;
    }
    public void ende() {
        tx.closeController();
    }
```

Neu ist hier die klare Trennung des eigentlichen Robot von seinen Werkzeugen : Liste RobMot mit einer Aufzählung der Robotmotoren und GripMot mit der Angabe des Greifermotors. Dazu gehören actPos mit der aktuellen Position des Robots und gripOpen für die Stellung des Greifers. In start() werden die Werte zurückgesetzt.

moveHome : Überarbeitet

```
public void moveHome() {
  System.out.println("Home simultan");
  for(Mot mot : RobMot)
      tx.startRobMotor(mot, Dir.Left, FullSpeed, 9999);
  tx.waitForMotors(RobMot);
  gripper(0);
```

```
  for(int pos : actPos) pos = 0;
}
```

In RobTXir2 nur noch die simultane Methode, die auch den Greifer berücksichtigt. Bei waitForMotors kann jetzt als Argument die Liste der beteiligten Motoren verwendet werden.

moveTo : Neuausgabe

```
public void moveTo(int... pos) {
  System.out.println("Robot fährt auf neue Position");
  for(int p : pos) System.out.print(" - " + p);
  for(int i = 0; i < actPos.length; i++) {
    if(pos[i] < actPos[i])
      tx.startRobMotor(Mot.values()[i], Dir.Left,
                       FullSpeed, actPos[i] - pos[i]);
    else if(pos[i] > actPos[i])
         tx.startRobMotor(Mot.values()[i], Dir.Right,
                          FullSpeed, pos[i] - actPos[i]);
  }
  tx.waitForMotors(Mot.M1, Mot.M2, Mot.M3);
  for(int i = 0; i < actPos.length; i++) actPos[i] = pos[i];
}
```

wie gehabt, jetzt mit Anzeige der anzufahrenden Position.

gripper : Extrawurst für den Greifer

```
public void gripper(int open) {
  if(open == 0) {
    System.out.println(" : Greifer öffnet");
    tx.startRobMotor(GripMot, Dir.Left, FullSpeed, 9999);
    tx.waitForMotors(GripMot);
    gripOpen = 0;
  }
  else if (gripOpen == 0) {
    System.out.println(" : Greifer schließt");
    tx.startRobMotor(GripMot, Dir.Right, FullSpeed, GripClosed);
    tx.waitForMotors(GripMot);
    gripOpen = GripClosed;
  }
}
```

Da der Griefer erst dann öffnet oder schließt, wenn der Robot sthet, auch eine separate Methode dafür. Sie lehnt sich an das bekannte driveHome/To an. Da der Parameter keine "Zwischentöne" kennt sondern nur "auf" / "zu" hier auch als Parameter die Werte 0 für auf und 1 für zu. Was das in Impulsen heißt, sagt die Konstante GripClosed. Zusätzlich wird bei "zu" abgefragt, ob der Greifer schon "zu" ist, so kann der Robot mit gleicher Greiferstellung zwischen mehreren Robot-Positionen verfahren werden.

Testen mit RobTXmainIR2

```
import ftcomputing.roboTX.*;
public class RobTXmainIR2 {
  static RobTXir2 rob = new RobTXir2();
  public static void main() {
    System.out.println("Robot gestartet");
    rob.start();
    RobListe(new int[][]{{55,  21,  33,  1},
                         {10,  22,  22,  0},
                         {56,  11,  11,  1}});
    rob.ende();
    System.out.println("--- Das war's ---");
  }
  private static void RobListe(int[][] posListe) {
    rob.moveHome();
    for(int[] pos : posListe) {
       rob.moveTo(pos);
       rob.gripper(pos[3]);
    }
  }
}
```

Eigentlich auch schon bekannt. Die Methode RobListe hat jetzt den Parameter mit der um die Greiferstellung erweiterten posListe bekommen. Nach Anfahren der Robot-Position mit moveTo wird der Greifer gesteuert. Die um einen Eintrag (Greifer) längere posListe mach dem moveTo nichts, da es sich intern an actPos orientiert.

FishFaceTX : Dreipunkt-Regelung (Console)

Eine Lampe an Out.O7 sitzt auf einem Schneckenantrieb mit Motor an Mot.M1. Sie soll den Photowiderstand an Unv.I4 mit einem vorgegeben SollWert so beleuchten, daß der Meßwert an Unv.I4 stets innerhalb vorgegebener Grenzen (LimitHigh / LimitLow) bleibt.

```
import ftcomputing.roboTX.*;

public class DreipMain {

   FishFaceTX tx    = new FishFaceTX();
   Mot ReglerMotor = Mot.M1;
   Unv LichtSensor = Unv.I4;
   Out Lichtquelle = Out.O7;

   public static void main(String[] args) {

          DreipMain dp = new DreipMain();
          System.out.println("Dreipunkt gestartet");
          System.out.println("FishFace-Version : " +
                             FishFaceTX.Version());
       try {dp.Action();}
      catch(FishFaceException eft) {System.out.println(eft);}
      finally {System.out.println("--- FINITO ---");}
   }

   private void Action() {

          int SollWert  = 700;
          int LimitHigh = SollWert + 100;
          int LimitLow  = SollWert - 100;
          int ActualValue;

       tx.openController("COM4");
       tx.setLamp(Lichtquelle, 512);
       tx.pause(1234);
       System.out.println("Sollwert : " + SollWert);
       System.out.println("Ende : ESC-Taste");
       do {
          ActualValue = tx.getAnalog(LichtSensor);
          System.out.println("Istwert : " + ActualValue);
          if(ActualValue < LimitLow)
                 tx.speedMotor(ReglerMotor, Dir.Right, 400);
          else if (ActualValue > LimitHigh)
                 tx.speedMotor(ReglerMotor, Dir.Left, 400);
          else tx.setMotor(ReglerMotor, Dir.Off);
          tx.pause(111);
       } while(!tx.finish());
       tx.closeController();
       System.out.println("Das war's");}}
```

Der Sollwert (und die Limits) wird hier im Programm festvorgegeben.

Mit openController / closeController wird die Verbindung zum TX Controller hergestellt. Auf eine Absicherung durch try - catch wurde hier verzichtet.

Innerhalb der do-Schleife wird zunächst der aktueller Wert des Photowiderstandes abgefragt und angezeigt. Danach erfolgt die eigentliche Regelung : akt. Wert kleiner LimitLow : Motor nach rechts, akt. Wert größer als LimitHigh : Motor nach links, sonst Motor Stopp.
Gemessen wird alle 111 mSek (wesentlich größere Zeiten können zu Fehlfunktionen führen, probieren mit dem eigenen Modell).