Übersichten und Hinweise zu

ftTeachCS

für die Industry Robots
und die ROBO & Intelligent Interfaces
C# Version

Ulrich Müller

Inhaltsverzeichnis

Bild : Einfügen | Graphik | Aus Datei... SEEPFRD2.WMF

Programm Funktionen

Allgemeines

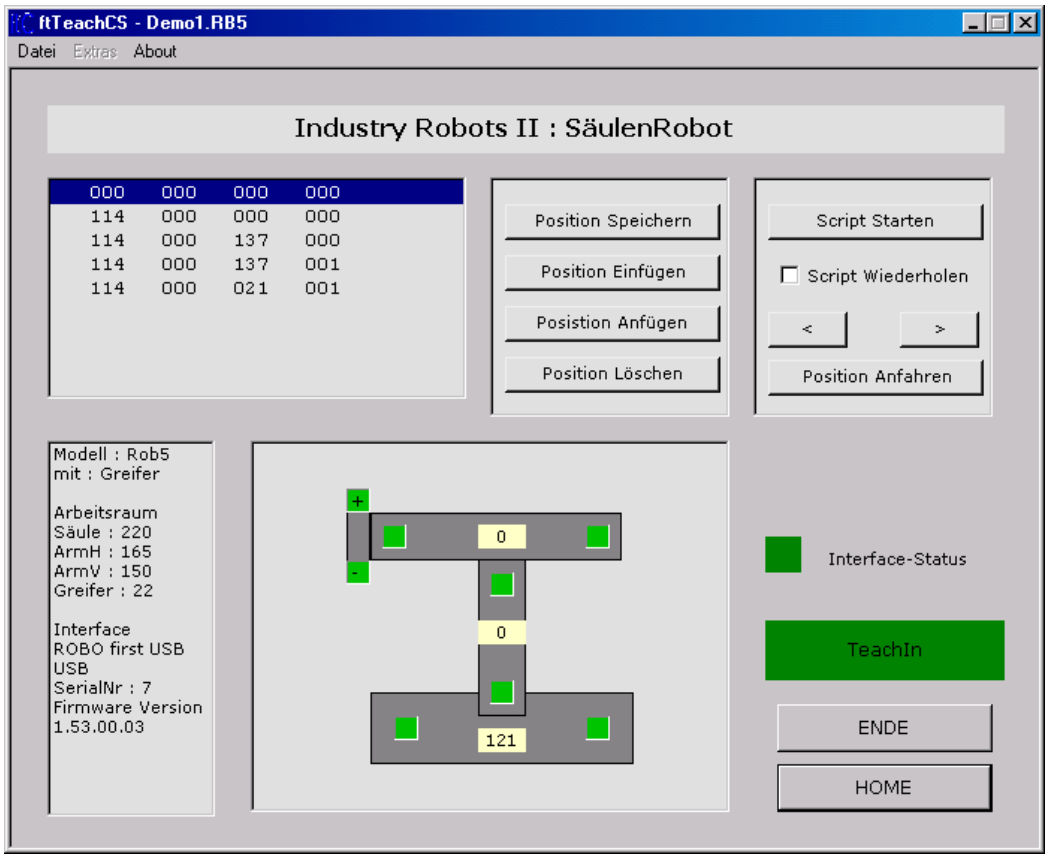
Das Programm ftTeachCS ermöglicht den TeachIn-Betrieb von Robots der fischertechnik Kästen Industry Robots I und II über das ROBO und das Intelligent Interface. Die durch das TeachIn gewonnenen Daten können in einem Scriptfile aufgezeichnet werden und können dann im Run-Modus des Programms als Ausgang für den selbständigen Betrieb des Robots genommen werden.

Hauptzweck des in C#.NET erstellten Programmes ist eine Einführung in ein komplexeres in C# geschriebenes Programm. Wenn man Wert auf ein fertiges TeachIn-Programm legt, sollte man ftTeach40.EXE nutzen (www.ftcomputing.de/ftteach.htm).

Das Programm wurde bewußt in C# Final Version geschrieben, es kann so mit der Final Version und – nach einer automatischen Konvertierung – auch mit Version 1.1 genutzt werden. Eine Nutzung zusammen mit SharpDevelop dürfte auch recht einfach möglich sein, hier ist ggf. ein Neuaufbau des Projektes erforderlich.

Programmfunktionen

Haupt-Form



Das Programm kann über die Robot-Graphik im TeachIn-Modus den Robot durch Klick auf die grünen Felder auf bestimmte Positionen verfahren werden und die erreichte Position über die Positions-Buttons abspeichern.

Über die Script-Buttons kann dann im Run-Modus das im TeachIn-Modus erstellte Script abgefahren werden.

Über das Datei-Menü kann ein Script abgespeichert werden und später auch wieder geladen werden.

Die Interface- und Modell-Einstellungen können über eigene Formen eingestellt und angepaßt werden.

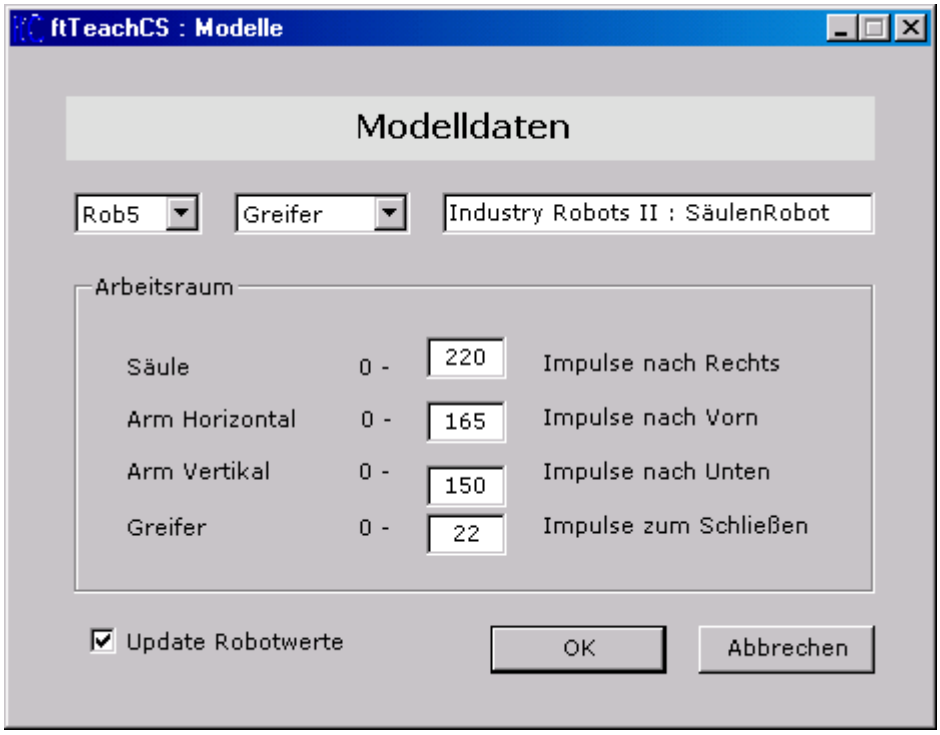
Der Abbruch des Run-Modus kann über einen mit HALT beschrifteten Button (aktuell im Bild mit HOME beschriftet) oder die ESC-Taste erfolgen.

Interface-Einstellungen

Aufruf über Menü | Extras | Interface erlaubt es die Auswahl des aktuell gewünschten Interfaces aus einer Liste der möglichen Interfaces. Bei einem Anschluß über USB kann zusätzlich noch die Serialnummer eingegeben werden. Bei Anschluß über COM kann der COM-Port ausgewählt werden.

Zusätzlich zur der Liste der möglichen Interfaces enthält Position 0 der Liste immer das aktuelle Interface. Es kann gewählt werden, ob SerialNr bzw. COM-Port auch in dieser Liste abgespeichtert werden sollen.

Modell-Einstellungen



Aufruf über Menü | Datei | Modell erlaubt die Auswahl des aktuell gewünschten Modells aus einer Liste der möglichen Modelle.

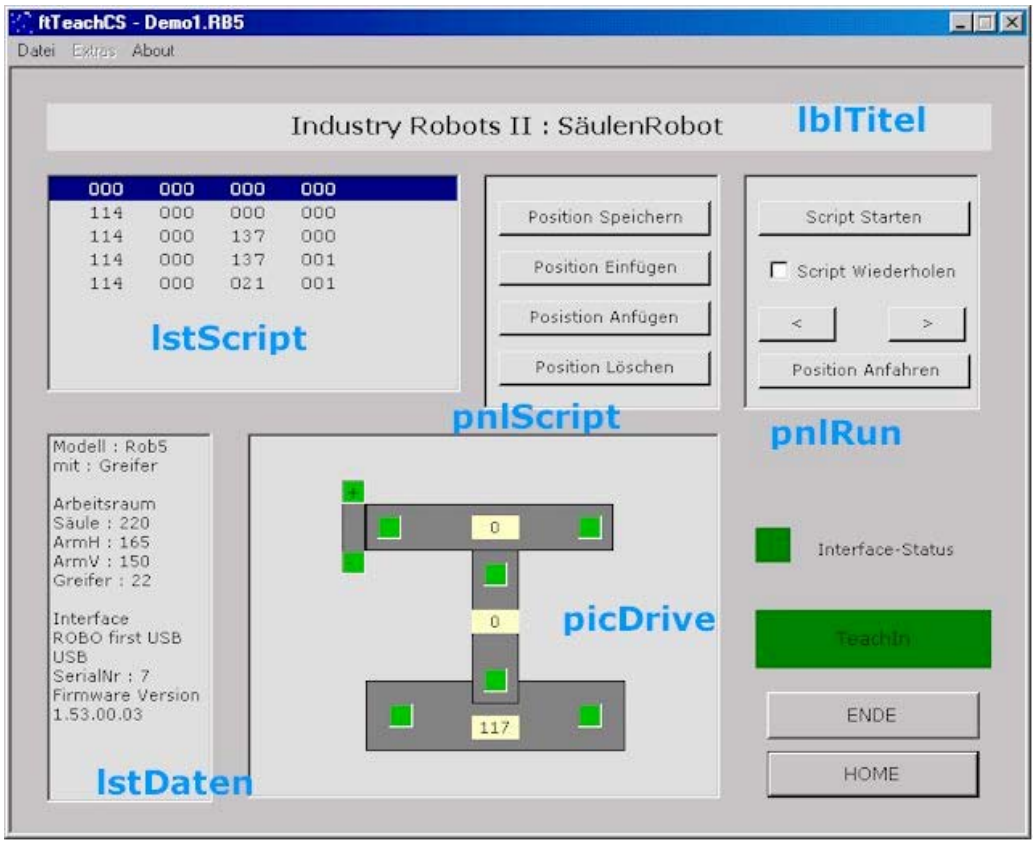
Zusatzgerät, beschreibender Text und Arbeitsraum können modifiziert werden und wahlweise auch in der Liste der möglichen Modelle gespeichert werden, Position 0 enthält immer das aktuelle Modell.

Source Programm

Source

TeachMain.CS :	Hauptform mit der wesentlichen Programmlogik
PerDaten.CS :	Klasse mit persistenten Daten (Modell, Interface)
Modell.CS :	Form zum Festlegen der Modelldaten
Schnitt.CS :	Form zum Festlegen der Interfacedaten
About.CS :	Form zur Anzeige von Programmdaten
FishFace40.CS :	Assembly mit Klassen zur Steuerung der Interfaces Source (2.12.05) entspricht der Assembly FishFace40.DLL

TeachMain



Aufbau : Controls, zugeordnete Aufgaben

lblTitel	Titelzeile mit dem Namen des aktuellen Robots noch offen : dyn. Titel in this.Text
lstScript	ListBox mit dem Daten des aktuellen Scriptfiles.
pnlScript	Panel mit den Buttons für Script Operationen cmdPosSpeichern : Speichern der aktuellen Position in markierte Zeile cmdPosEinfugen : Einfügen der aktuellen Position vor markierte Zeile cmdPosAnfugen : Hinzufügen der aktuellen Position ans Scriptende cmdPosLoschen : Löschen der markierten Position
pnlRun	Panel für Button für den RobotBetrieb mit ScriptDaten. cmdRunStart : Fahren gesamtes Scipt chkRunRepeat : Endlose Wiederholung des Scripts cmdRunBack : Fahren zur Position vor der Markierten cmdRunForward : Fahren zur Position hinder der Markierten cmdRunTo : Fahren zur markierten Position.
lstDaten	Auflistung der aktuellen Robot und InterfaceDaten
picDrive	TeachIn- und AnzeigePanel für den Robot Anzeige | Fahren 1 | Fahren 2 | Position lblSaule, lblSauleLinks, lblSauleRecht, lblSaulePos lblArmH, lblArmHVor, lblArmHRuck, lblArmHPos lblArmV, lblArmVHoch, lblArmVAb, lblArmVPos lblGreifer, lblGreiferAuf,
cmdAction	Click-Ereignis : EIN : Starten Interface

	STARTEN : Starten Robot-Betrieb mit HomeRun HALT : Anhalten Robot-Betrieb
cmdEnde	Click-Ereignis Beenden des Programms im Zusammenspiel mit dem Form_Closed Ereignis
lblStatus	Anzeige des Betriebsstatus
lblBetrieb	Anzeige des Interfacestatus
mnuModell	Aufruf der Modell-Form
mnuAbout	Aufruf der About-Form
mnuInterface	Aufruf der Interface-Form
mnuNeu	Starten mit neuem Script-File
mnuOffnen	Öffnen eines vorhandenen Script-Files
mnuSpeichern	Speichern des aktuellen Script-Files
mnuSpeichernU	Speichern des aktuellen Script-Files unter neuem Namen
tmrI	Überwachung des Interfaces
Positionsanzeige	Anzeige der aktuellen Robot-Position und der Robot-Aktivitäten Ereignis-Routine aus FishFace / FishRobot

C# Programmiertechniken

Serialisierung / Persistenz

Einige Programmdaten, die Interfacedaten und die Modelldaten werden über den Programmlauf hinaus in einem binären File gespeichert. Dazu wird die .NET Fähigkeit der Serialisierung genutzt (System.Runtime.Serialization.Formatters.Binary). Dazu müssen die zu speichernden Daten in besonders gekennzeichneten Klassen([Serializable()] im File PerDaten.CS) PerDaten mit den Listen für die Interface- und Modelldaten untergebracht werden. Außerdem kann die Adresse der aktuellen Instanz bequem als Konstruktor-Parameter an die Forms zur Modifikation der Daten übergeben werden.

Die Klasse PerDaten ist gleichzeitig ein Verzeichnis aller zulässigen Interfaces und Modelle. Das aktuelle Interface bzw. Modell befindet sich auf Position 0 der jeweiligen Liste.

Das File mit den binären, persistenten Daten liegt im Pfad der ftTeachCS.EXE.

```
private string IniName
     = new DirectoryInfo(".\\").FullName + "ftTeachCS.BIN";
```

Im frmMain_Load Ereignis wird eine Instanz (pd) der Klasse PerDaten erzeugt :

```
private void frmMain_Load(object sender, System.EventArgs e) {
  prgModus = PrgModus.None;
  if(File.Exists(IniName)) {
    BinaryFormatter ser = new BinaryFormatter();
    FileStream bin      = new FileStream(IniName, FileMode.Open);
    pd                  = (PerDaten)ser.Deserialize(bin);
    bin.Close();
  }
  else pd = new PerDaten();

  this.Left      = pd.FormLeft;
  this.Top       = pd.FormTop;
  this.Text      = Application.ProductName;
  ScriptName     = " [Neu]";

  Modell         = pd[0];
  Interface      = pd.IntModelle[0];
  ModellDaten();
  ModusWechsel(PrgModus.Ein);
}
```

Wenn kein ftTeachCS.BIN vorhanden ist, werden die Default-Daten aus der Klassen-Instanz pd von PerDaten genutzt. Ansonsten wird der BinaryFormatter (ser) und ein Hilfsfile (bin) aufgesetzt, der Formatter liest dann mit einem Befehl die gespeicherten Daten und deserialisiert sie dabei.

Im frmMain_Closed Ereignis werden die aktuellen Daten der Instanz pd von PerDaten dann serialisiert :

```
private void frmMain_Closed(object sender, System.EventArgs e) {
  pd.IntModelle[0]    = Interface;
  tmrI.Enabled        = false;
  BinaryFormatter ser = new BinaryFormatter();
  FileStream bin      = new FileStream(IniName, FileMode.Create);
  pd.FormLeft         = this.Left;
  pd.FormTop          = this.Top;
  ser.Serialize(bin, pd);
  bin.Close();

  DirtySpeichern();    // --- Speichern Scriptfile, wenn Dirty
```

```
  if(ft != null) ft.CloseInterface();
}
```

Zunächst werden die aktuellen InterfaceDaten in pd gespeichert, dann werden wieder ser und bin aufgesetzt und die pd-Daten mit ser.Serialize serialisiert und in ftTeachCS.BIN gespeichert. Die weiteren Befehle runden das Programmende ab, gehören aber nicht zum Serialisieren.

Script-Files

Die Script-Files sind einfache Text-Files, die die Liste der anzufahrenden Positionen enthalten. Reihenfolge : Säule, Arm waagerecht, - senkrecht, Zusatz / Greifer (Offen 0, geschlossen 1) :

```
private void mnuOffnen_Click(object sender, System.EventArgs e) {
  string Zeile;
  DirtySpeichern();
  openScript.InitialDirectory = Application.StartupPath;
  if(openScript.ShowDialog() == DialogResult.OK) {
    ScriptName = openScript.FileName;
    StreamReader sr = new StreamReader(ScriptName);
    Zeile = sr.ReadLine();
    lstScript.Items.Clear();
    while(Zeile != null) {
      lstScript.Items.Add(Zeile);
      Zeile = sr.ReadLine();
    }
    sr.Close();
    lstScript.SelectedIndex = 0;
    this.Text = Application.ProductName + " - " +
                ScriptName.Substring(ScriptName.LastIndexOf(@"\")+1);
    IsDirty = false;
  }
}
```

In mnuOffnen wird zum Öffnen der normale DateiDialog geführt und dann über den System.IO.StreamReader zeilenweise (ReadLine) gelesen und in die ListBox lstScript abgestellt. Der Name des gelesenen Files wird zusätzlich in die Titelzeile der Form abgestellt. Das Speichern geschieht analog über mnuSpeichern, mnuSpeichernU unter Nuzung der Routine Speichern (sr.WriteLine()).

In beiden Fällen wird ein IsDirty-Flag beachtet über das ggf. (IsDirty == true) eine Abfrage gestartet wird, ob das aktuelle Script-File gespeichert werden soll.

Die aktuelle Position (erreichte / anzufahrende) wird in einem Array MoveWerte gehalten (MoveToScript() / ScriptToMove()).

Der Aufruf der Routinen erfolgt über Menü.

Programm-Ablaufsteuerung

Um Fehlbedienungen, die zu einem Crash des Modells führen können, schon im Vorfeld ausschließen zu können, werden die infrage kommenden Controls der Bedieneroberfläche in Abhängigkeit eines "Programm-Modus" matrixartig gesteuert (Routine ModusWechsel). In der Routine werden alle betroffenen Controls auf dem zum Programm-Modus(Ein, Home,TeachIn, Run) passenden Stand gesetzt (z.B. Enabled = true;)

Der Programm-Ablauf selber wird durch den Button cmdAction gesteuert :

```
private void cmdAction_Click(object sender, System.EventArgs e) {
  if(cmdAction.Text == "EIN") {
    try {
      ft = new FishRobot(new int[,] {{1, Modell.Saule},
                                     {2, Modell.ArmH},
                                     {3, Modell.ArmV},
```

```
                                        {4, Modell.Greifer}});
      if (Interface.COM)
        ft.OpenInterface((IFTypen)Interface.Typ,
                (int)Interface.pNr, 999, true);
      else
        ft.OpenInterface((IFTypen)Interface.Typ,
                      Interface.Serial, true);
      ft.PositionChange  +=
                   new FishRobot.CommonDelegate(PositionsAnzeige);
      Interface.Firmware  = ft.ActDevice.Firmware;
      Interface.Serial    = ft.ActDevice.SerialNr;
      tmrI.Enabled        = true;
      ModellDaten();
      ModusWechsel(PrgModus.Home);
     }
    catch (FishFaceException eft) {
      lblStatus.Text = eft.Message;
      lblStatus.BackColor = Color.Red;
      ft.CloseInterface();
    }
  }
  else if(cmdAction.Text == "HALT") {
    ft.NotHalt = true;
    chkRunRepeat.Checked = false;
  }
  else {
    try {
      ModusWechsel(PrgModus.Home);
      HomeRun();
      ModusWechsel(PrgModus.TeachIn);
    }
    catch(FishFaceException eft) {
      lblStatus.Text = eft.Message;
      lblStatus.BackColor = Color.Red;
    }
  }
}
```

Bei Start des Programms hat er die Beschriftung EIN, es wird das zuvor ausgewählte Interface gestartet. Den Achsen des Modells werden dabei Motornummer und maximaler Fahrweg zugeordnet. Das OpenInterface geschieht in Abhängigkeit vom Interface-Anschluß. Nach erfolgreichem Open werden die aktuellen Interface-Daten ausgelesen und angezeigt. Die Beschriftung des Buttons wechselt auf HOME (in ModusWechsel).

Bei Beschriftung HOME gilt der else-Zweig. Die Home-Position wird über HomeRun() angefahren.

In den anderen Fällen ist der Button mit HALT beschriftet, ein Setzen von ft.NotHalt = true hält den Robot an.

TeachIn-Steuerung

Auf dem picDrive ist aus einer Reihe von Labeln ein Robot-Symbol zusammen gebaut worden. Der recheckige (grüne) Label zur Steuerung der einzelnen Robot-Komponeten über Maus genutzt werden :

```
private void lblRobMouseDown(object sender,
  System.Windows.Forms.MouseEventArgs e) {
  if(ft == null) return;
  Label l = (Label)sender;
  ft.NotHalt = false;
  try {
    switch(Convert.ToInt32(l.Tag)){
```

```
      case 1:             // --- Säule zum Endtaster
        ft.MoveTo(0, ft.MotCntl[1].actPos, ft.MotCntl[2].actPos);
        break;
      case 2:             // --- Säule weg vom Endtaster
        ft.MoveTo(ft.MotCntl[0].maxPos,
              ft.MotCntl[1].actPos, ft.MotCntl[2].actPos);
        break;
      case 3:            // --- Arm, horizontal, zurück
        ft.MoveTo(ft.MotCntl[0].actPos, 0, ft.MotCntl[2].actPos);
        break;
      case 4:            // --- Arm, horizontal, vor
        ft.MoveTo(ft.MotCntl[0].actPos, ft.MotCntl[1].maxPos,
              ft.MotCntl[2].actPos);
        break;
      case 5:           // --- Arm, vertikal, auf
        ft.MoveTo(ft.MotCntl[0].actPos, ft.MotCntl[1].actPos, 0);
        break;
      case 6:           // --- Arm, vertikal, ab
        ft.MoveTo(ft.MotCntl[0].actPos, ft.MotCntl[1].actPos,
               ft.MotCntl[2].maxPos);
        break;
      }
    }
    catch(FishFaceException eft) {
      lblBetrieb.BackColor = Color.Red;
      lblStatus.Text       = eft.Message;
    }
}

private void lblRobMouseUp(object sender,
           System.Windows.Forms.MouseEventArgs e) {
    if(ft != null) ft.NotHalt = true;
}

private void lblGreiferClick(object sender, System.EventArgs e) {
    if(ft == null) return;
    ModusWechsel(PrgModus.Run);
    ft.NotHalt = false;
    Label l = (Label)sender;
    int OnOff = Convert.ToInt32(l.Tag);
    if(OnOff == 7) OnOff = 1; else OnOff = 0;
    Zusatz(OnOff);
    ModusWechsel(PrgModus.TeachIn);
}
```

Genutzt werden die MouseDown- und MouseUp-Ereignisse der Labels. Bei MouseDown wird der zugehörende Motor gestartet, bei MouseUp wieder angehalten. Zum Motorbetrieb wird die Methode MoveTo von FishRobot genutzt. Es müssen deswegen alle Positionen des Robots angegeben werden. Für den betroffenen Motor (in der Numerierung der Instanzierung) wird für Fahren nach links (auf den Endtaster zu) als ZielPosition 0 angegeben, bei Fahren nach rechts die max. mögliche. Für die nicht betroffenen Motoren wird deren aktuelle Postion angegeben (keine Angst, intern wird das erkannt, der Motor wird nicht gestartet). Positionen hinter der betroffenen können entfallen.

Um mit einer Ereignis-Routine auszukommen, wurde der Motor und die anzufahrende Richtung in der Tag-Eigenschaft des jeweiligen Label codiert und hier in einem switch-Konstrukt ausgewertet.

Das Anhalten des Motors (aller Motoren) geschieht durch Setzen von ft.NotHalt = true; Deswegen auch zu Anfang von MouseDown ft.NotHalt = false.

Der Greifer bzw. ein Zusatzgerät wird stets vollständig geschaltet. Das geschieht in lblGreiferClick.

Status-Anzeige

Der Interface-Betrieb wird durch eine Timer-Routine (tmrI) überwacht (lblBetrieb grün/rot). Es ist darauf zu achten, dass der Timer vor Programmende abgeschaltet wird.

Der aktuelle Robot-Betrieb (einschließlich laufender Positionsangabe) wird in einer Ereignis-Routine von ft (FishRobot) angezeigt :

```
private void PositionsAnzeige(object sender, int[] actPos) {
   try {
      int EW = ft.Outputs;

      lblSaulePos.Text = actPos[0].ToString();
      if((EW & 0x3) > 0) lblSaule.BackColor = Color.GreenYellow;
      else lblSaule.BackColor = Color.Gray;

      lblArmHPos.Text  = actPos[1].ToString();
      if((EW & 0xC) > 0) lblArmH.BackColor = Color.GreenYellow;
      else lblArmH.BackColor = Color.Gray;

      lblArmVPos.Text  = actPos[2].ToString();
      if((EW & 0x30) > 0) lblArmV.BackColor = Color.GreenYellow;
      else lblArmV.BackColor = Color.Gray;

      if((EW & 0xC0) > 0) lblGreifer.BackColor = Color.GreenYellow;
      else lblGreifer.BackColor = Color.Gray;
   }
   catch(FishFaceException eft) {
      lblBetrieb.BackColor = Color.Red;
      lblStatus.Text       = eft.Message;
   }
}
```

Dazu wird die ft.Outputs-Eigenschaft entsprechend maskiert. Eine Anzeige in der Timer-Routine wäre genauso möglich gewesen, die Teilung ist eher Geschmacksache.