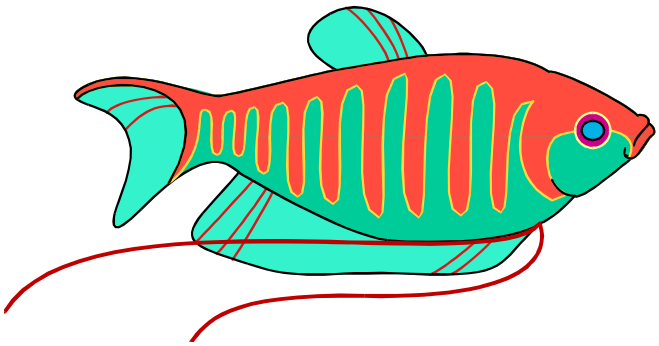
ftComputing

FishFaceTX für VB.NET

Programmierung des ROBO TX Controllers
mit VB 2005, 2008 und 2010

Ulrich Müller



Inhaltsverzeichnis

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873
eMail : UM@ftComputing.de
HomePage : www.ftcomputing.de und : www.ftcomputing.de/fishdllstx.htm

Dokumentname : FishFaTXVB.doc. Druckdatum : 04.10.2010
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish9.WMF

Einführung

Allgemeines

MIt der in C# geschriebenen Assembly FishFaceTX.DLL (Namensraum FishFaceTX) wird die Möglichkeit geboten, den fischertechnik ROBO TX Controller unter einer .NET 2.0 (und höher) Sprache zu programmieren. FishFaceTX.DLL setzt auf umFish50.DLL und der auf ftMscLib.DLL (die von fischertechnik gestellt wird) auf. Die zentrale Klasse FishFace von FishFaceTX erlaubt die Ansteuerung der TX Controller über USB und Bluetooth.

Angeboten werden Befehle zur Schaltung der M-Ausgänge und zur Abfrage der Eingänge eines Interfaces. Der TX Controller wird durch einen MultiMediaTimer in umFish50.DLL überwacht. Dabei werden die Counter der C-Eingänge mit ihren Sollwerten verglichen. Bei Erreichen eines Sollwertes wird der Counter auf Null zurückgesetzt und der zugeordnete Motor abgeschaltet. Zusätzlich werden die Motoren beim Erkennen der ESC-Taster abgeschaltet.

Universal-Eingänge (I-Eingänge), die im Modus Analog laufen, liefern Raw-Werte im Bereich von 0 – 5000. Die M-bzw.O-Ausgänge können mit einer Power (PWM) im Bereich von 0-512 betrieben werden.

Die mit FishFaceTX erstellten Programme laufen im sog. "Online"-Betrieb, d.h. das auf dem Windows PC laufende Programm muß über USB bzw. Bluetooth mit dem Controller verbunden sein.

Getestet wurde mit : C# 2005 und VB2005 Express unter Vista mit .NET 3.5. Ein Ablauf unter .NET 2.0 ist auch möglich ebenso ist ein Upgrade auf VB2008 leicht durchzuführen.

Installation

Vorausgesetzt wird ein Windows System ab Windows 2000 mit einem installierten C# 2005 (ab Express Edition, C# 2008 ist auch möglich) mit dem .NET-Framework 2.0. Bezug der Express Edition siehe Literaturübersicht

Zusätzlich erforderlich ist die Installation des Programmpaketes "PC-Programming-12.zip" (oder höher) von www.fischertechnik.de Downloads. Es enthält, neben ftMscLib.DLL, die erforderlichen USB/Bluetooth Treiber und das Testtool "ROBO TX Test" für den Controllertest. Dort werden auch die erforderlichen COM-Namen für den Verbindungsaufbau mit dem TX Controller angezeigt (ROBO Pro tut das leider nicht).

Zusätzlich sollte das fischertechnik ROBO Pro installiert sein. Es ermöglicht die schnelle Erstellung einfacher Testprogramme.

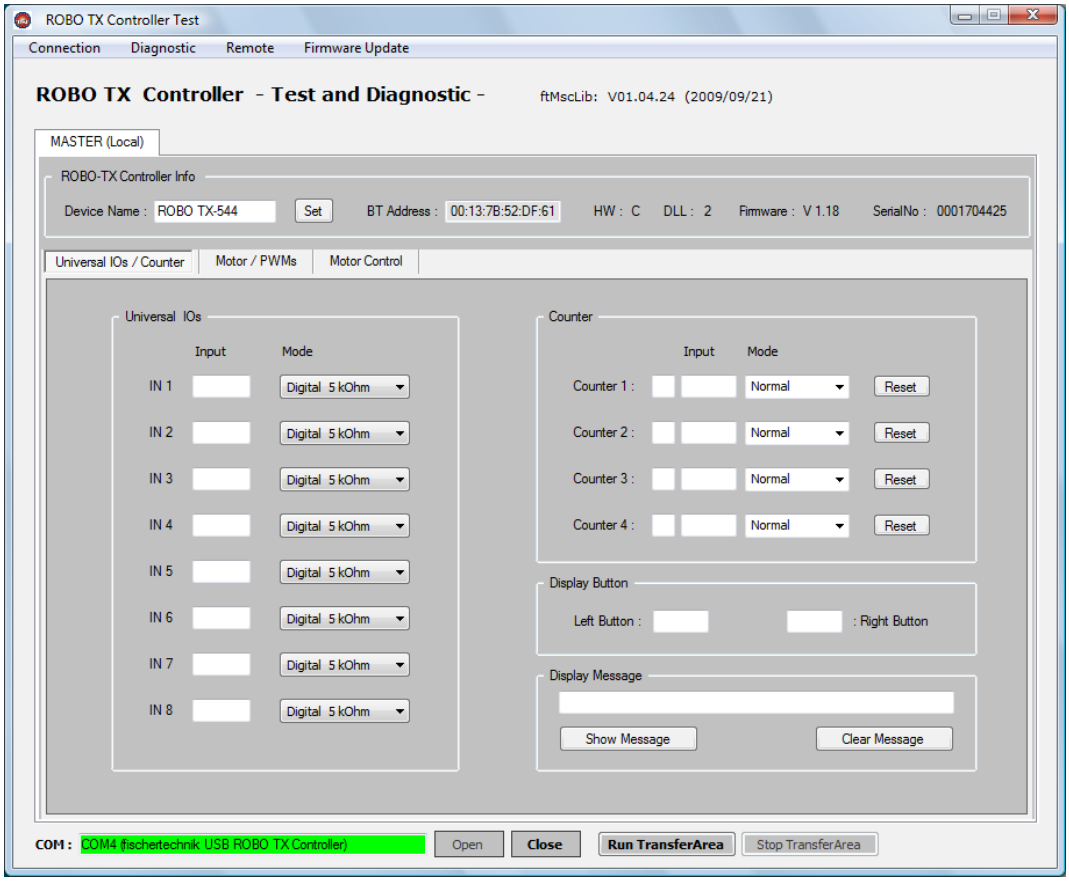
Das Paket FishFaceTX.zip enthält die Assembly FishFaceTX.DLL und die umFish50.DLL, nicht aber die ebenfalls erforderliche ftMscLib.DLL, sie ist in "PC-Programming-12.ZIP und in ROBO Pro enthalten. Zusätzlich im Paket enthalten sind Programmbeispiele und Programmvorlagen(Templates). FishFaceTX.DLL sollte an einem zentralen Ort untergebracht werden um die erforderlichen Verweise leicht einrichten zu können. Hinweis : Es kann vorkommen, daß in den BeispielProjekten die Verweise auf FishFaceTX.DLL nicht stimmen, nach dem Laden des Projektes einfach im Projektmappen Explorer (Verweise) richtigstelllen. umFish50.DLL bringt man am besten im Verzeichnis \Windows\System32 unter.

Robo TX Controller

Allgemein

Ein Master und 0 - 8 an den Master angeschlosse Extensions. Master und Extension sind baugleich, sie müssen aber über ihr Display entsprechend konfiguriert werden. Pro Rechner kann nur ein Master betrieben werden. Parallel dazu können aber ROBO Interfaces betrieben werden. Die Programme laufen stets im "online" Modus auf dem PC, sie können über einen USB-Anschluß oder Bluetooth betrieben werden. Der Bezeichnung für einen Ein- oder Ausgang muß bei den entsprechenden FishFaceTX-Methoden der Name der jeweiligen Extension vorangestellt werden, beim Main-Controller kann der Name entfallen.

ROBO TX Test Panel



Das Test Tool aus dem fischertechnik Download-Päckchen (PC-Programming-12.zip)

Belegung bei den Code-Beispielen im weiteren Text meist :

M1 Motor mit 4er Impulsrad
C1 Taster als Impulszähler
I1 Endtaster

M2 Motor mit 4er Impulsrad
C2 Taster als Impulszähler
I2 Endtaster

I3 Taster
I4 Farbsensor
I5 Phototransistor
I6 Photowiderstand
I7 Spursensor links
I8 Spursensor rechts

Anstelle eines einfachen Motors mit Impulstaster kann auch ein Encodermotor angeschlossen werden. Dabei ist auf die deutlich größere Impulszahl zu achten.

Sensoranschlüsse

I1 - I8 Universalanschlüsse

Beim TestPanel müssen sie entsprechend konfiguriert werden, bei FishFace gibt es pro Anschlußtyp eine entsprechende Methode. Bei den einzelnen Sensortypen werden die von fischertechnik angebotenen Sensoren aufgeführt, weitere dürften möglich sein.

D5K Digital 5 kOhm : **GetInput**
Taster, Reedkontakt, PhotoTransitor
Anschluß : 2polig an I-Eingang

D10V Digital 10 V : **GetTrack**
SpurSensor
Anschluß : rot an +9V, grün an Masse, gelb / blau an zwei verschiedene I-Eingänge

A5K Analog 5kOhm : **GetAnalog**
NTC, Photowiderstand, Potentiometer
Anschluß : 2polig an I-Eingang

A10V Analog 10 V : **GetVoltage**
FarbSensor, Spannung allgem.
Anschluß : rot an +9V, grün an Masse, schwarz an I-Eingang

C1 - C4 Zählereingänge

Zählereingänge : Schnelle Zähler für den EncoderMotor, können aber auch genausogut mit normalen Motoren und der Kombination Impulsrad/Taster genutzt werden. Außerdem können sie wie D5K-Eingänge genutzt werden.

Aktorenanschlüsse

M1 - M4 : Motor, Lampe, Magnetventil, Elektromagnet, Summer, EncoderMotor. Power einstellbar im Bereich 0 - 512

Alternativ

O1 - O8 : Einpolig + Masse, einzelne M-Eingänge können als zwei O-Eingänge genutzt werden. Anschließbare Devices wie M-Eingänge, bei Motoren ist dann aber kein Drehrichtungswechsel möglich.

Struktur der Assembly FishFaceTX.DLL

Basis sind die ftMscLib.DLL und darauf aufsetzend die umFish50.DLL.

Immer Erforderlich : Verweis auf FishFacecTX.DLL und `Imports FishFaceTX`

Programmiermodell FishFace

Kontrollierter Zugriff auf die Ein- und Ausgänge des TX-Controllers über die Klasse FishFace. Zusätzlich FishFaceException und Enumerationen.

FishFace

Klasse für den kontrollierten Zugriff auf den TX-Controller (Open… / Close…, Get… / Set…), zusätzlich : komplexere Methoden (Wait… / …Motor).

FishFaceException

Allgemeine Exception, die bei Fehlern und Fehlfunktionen ausgelöst wird.

Enumerationen

Primär zur Definition der Parameter von Eigenschaften und Methoden (Ctr, Mot, Out, Unv, Cnt, Dir, AnalogState, TrailState)

Programmiermodell Devices

Zugriff auf die Ein- und Ausgänge des TX-Controllers über spezifische Klassen, die dann zusätzliche - devicespezifische - Methoden bieten. Hinzukommen "Combined Devices", Klassen, die mehrere Ein- und Ausgänge zu einer Funktionseinheit zusammenfassen (z.B. TwinMotor : simultaner Betrieb von zwei Motoren -> Explorer).

FishControl

Verwaltung der zu einem TX-Controller (einschl. etwa angeschlossener) Extensions) gehörenden Devices sowie allgemeine Methoden (Connect, DisConnect, Finish, Pause) zu Umgang mit dem TX-Controller

FishFaceException - Enumerationen

wie oben.

Device-Klassen

DeviceBase

- **AnalogInput** Auslesen von Eingängen, die als Ergebnis einen numerischen Wert liefern
    - **A10VInput** Anschluß für analoge SpannungsSensoren.
        - **ColorSensor**
    - **A5KInput** Anschluß für WiderstandsSensoren
        - **NTC**
        - **PhotoResistor**
        - **PotentioMeter**

    - **DistanceSensor**
- **BinaryInput** Auslesen von Eingängen, die als Ergebnis True/False liefern
    - **D5KInput** Anschluß von WiderstandsSensoren
        - **PhotoTransistor**
        - **PushButton**
        - **ReedContact**
    - **HalfTrack** Anschluß für einen halben Spursensors
- **CounterInput** Zählereingang
- **DualOutput** Zweipoliger Ausgang
    - **Motor**
- **MonoOutput** Einpoliger (Pol + Masse) Ausgang
    - **Buzzer**
    - **Lamp**
    - **Magnet**
- --- Combined Devices ---
- **Drive2NXT** (2x EncoderMotor)
- **EncoderMotor** (Motor, CounterInput)
- **LightBarrier** (Lamp, PhotoTransitor)
- **Lights** (nx Lamp)
- **LimitedMotor** (Motor, 2x BinaryInput)
- **RobMotor** (Motor, D5KInput, CounterInput)
- **RobMotors** (nx RobMotor)
- **TrailSensor** (2x HalfTrack)
- **TwinMotors** (2x Motor)

Auf das Programmiermodell Devices wird hier nicht weiter eingegangen (z.Zt. das FishFaceTXdev Dokument für C# 2005 nutzen). Im Abschnitt KlassenAusflug finden sich aber zwei Beispiele dazu.

Literatur zu Visual Basic .NET

- Microsoft : Visual Basic 2005 Express Edition. ISBN 3-86063-567-0.
  Kurzer Überblick der Möglichkeiten von Visual Basic 2005.
  CD mit der Visual Basic 2005 Express Edition (19,90 €).
- Peter Bloch : Einstieg in Visual Basic 2005. ISBN 3-89842-641-6
  Für Programmieranfänger.
  Mit **CD Visual Basic 2005 Express Edition**. (24,90 €)
- O'Reilly : Programming Visual Basic.NET, ISBN 0-596-00093-6, als Übersicht.
- O'Reilly : VB.NET in a Nutshell, 0-596-00308-0, als Referenz neben der recht ansprechenden Hilfe des Visual Studio.NET.

Und dann gibt es jetzt auch von den altbekannten VB-Autoren eine VB.NET Version :

- Michael Kofler : Visual Basic 2005 - Grundlagen, Programmiertechniken, Windowsanwendungen - Addison-Wesley ISBN 3-8273-2338-X
- Doberenz / Kowalski : Grundlagen und Profiwissen - Visual Basic.NET, Hanser ISBN 3-446-22024-0. Auch in einer Neuausgabe für VB2005 erhältlich, diese ist aber nur noch "Profi", eine Grundlagen-Ausgabe ist in Arbeit.

In den beiden letztgenannten Büchern wird auch ausführlich auf die Programmierung mit Windows.Forms eingegangen.Die Doberenz Bücher sind wirklich für Profis gedacht.

Inzwischen sind die angeführten Bücher auch in VB2008 Versionen erhältlich.

Die StartAmpel

So geht's los :

- TX Controller anschließen, dessen Funktion mit dem ROBO TX Test kontrollieren (COM-Namen merken)
- An den TXcontroller anschließen O5 : rote, O6 gelbe, O7 rote Lampe
- Console-Projekt StartAmpel aus öffnen
- In der Projektübersicht Verweis (Referenz / Verweise) FishFaceTX.DLL (Assemblies) ggf. korrigieren.
- F5 : Starten.

Code des Console-Programms :

```
Imports FishFaceTX
Module StartAmpelVB
   Dim tx As New FishFace()
   Sub Action()
         Do
               tx.SetLamp(Out.O5, Dir.On)
               tx.Pause(1000)
               tx.SetLamp(Out.O6, Dir.On)
               tx.Pause(500)
               tx.SetLamp(Out.O5, Dir.Off)
               tx.SetLamp(Out.O6, Dir.Off)
               tx.SetLamp(Out.O7, Dir.On)
               tx.Pause(2000)
               tx.SetLamp(Out.O7, Dir.Off)
         Loop Until tx.Finish()
   End Sub
#Region "--- ProgramControl ---"
Sub Main()
  Try
    tx.OpenController("COM4") ' --- den COM-Namen anpassen
    Console.WriteLine("--- Bei der Arbeit, Abbruch ESC-Taste ---")
    Action()
  Catch txe As FishFaceException
     Console.WriteLine(txe.Message)
  Finally
     tx.CloseController()
      Console.WriteLine("--- FINITO : RETURN drücken ---")
      Console.Read()
  End Try
End Sub
#End Region
```

Das Programm steht im Verzeichnis StartAmpel.

Zu den Elementen :

- Das (Console) Programm befindet sich im Module StartAmpelVB
  ```
  Dim tx As New FishFace()
  ```

anlegen einer neuen Instanz der Klasse FishFace (Teil von FishFaceTX.DLL) mit dem Namen tx. Unter diesem Namen werden dann die Methoden (Funktionen) der Klasse FishFace angesprochen.

- Im Module befindet sich eine Methode Main mit der die Anwendung gestartet wird. Sie enthält eine Try .. Catch … Finally Klammer mit der Fehler, die vom TX Controller herrühren, abgefangen werden.
- `tx.OpenController("COM4")` : Herstellen einer Verbindung zum TXcontroller (siehe ROBO TX Test : Linke untere Ecke).
- `tx.SetLamp(Out.O5, Dir.On)` : Einschalten der roten Lampe
  Die Methoden von FishFace bieten meist einen Auswahlliste (enum) möglicher Parameterwerte. Hier aus der Aufzählung Out und Dir. Es können aber auch eigene Konstanten angegeben werden (z.B. Const Out LampeRot = Out.O5).
- `tx.Pause(1000)` : Das Programm wird für 1000 MilliSekunden (1 Sekunde) angehalten.
- `tx.SetLamp(Out.O6, Dir.On);` die gelbe Lampe wird für 500 MilliSekunden zugeschaltet. und dann werden beide aus und die grüne Lampe an O7 wird für 2000 MilliSekunden angeschaltet.
- Das läuft dann in einer Endlos-Schleife, die durch die Esc-Taste abgebrochen werden kann.
- Danach und der Ordnung halber : `tx.CloseController()`, die Verbindung zum Interface gekappt.

Siehe auch www.ftcomputing.de/vb7ecke.htm .

Beispiele

Dreipunktregelung

Eine Lampe an Out.O7 sitzt auf 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.


```
Imports FishFaceTX
Public Class DreipunktunvVB
  Dim tx As New FishFace()
  Dim ReglerMotor As Mot = Mot.M1
  Dim LichtSensor As Unv = Unv.I4
  Dim LichtQuelle As Out = Out.O7
  Dim Sollwert, LimitHigh, LimitLow, ActualValue As Integer

  Private Sub numSollWert_ValueChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
                                  Handles numSollWert.ValueChanged
    Sollwert = numSollWert.Value
    LimitHigh = Sollwert + 100
    LimitLow = Sollwert - 100
  End Sub
```

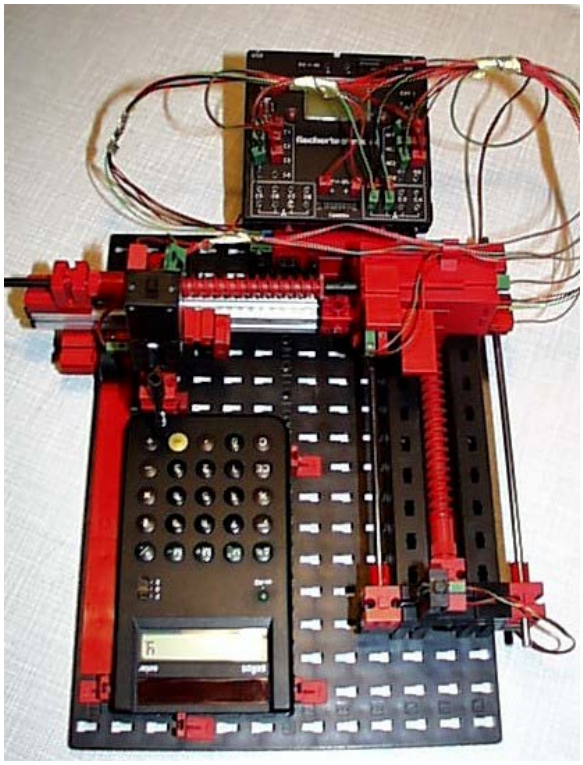
```
Private Sub Action()
  numSollWert.Value = 700
  tx.SetLamp(LichtQuelle, Dir.On)
  tx.Pause(1234)
  Do
    ActualValue = tx.GetAnalog(LichtSensor)
    lblIstwert.Text = ActualValue.ToString()
    If ActualValue < LimitLow Then
      tx.SetMotor(ReglerMotor, Dir.Right, 400)
    ElseIf ActualValue > LimitHigh Then
           tx.SetMotor(ReglerMotor, Dir.Left, 400)
    Else
      tx.SetMotor(ReglerMotor, Dir.Off)
    End If
    tx.Pause(111)
  Loop Until tx.Finish()
End Sub
```

Der Sollwert wird durch das Control numSollwert (NumericUpDown) in der Ereignisroutine numSollwert_ValueChanged samt Grenzwerten eingestellt.

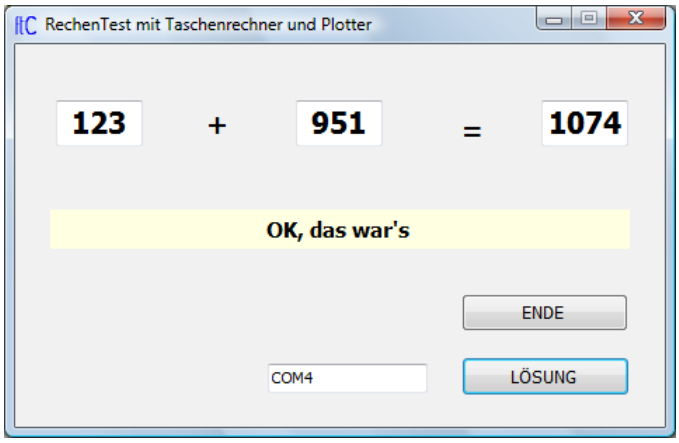
In der Click-Routine des Buttons cmdAction läuft in einer Endlosschleife das eigentliche Programm :
- tx.OpenController / tx.CloseController : Verbindung zum TX Controller (Name COM4 anpassen).
- die Do-Schleife über die Messungen kann durch die ESC-Taste beendet werden.
- in der Schleife wird zuerst der aktuelle Wert am Photowiderstand gemessen und im Label-Control lblIstWert angezeigt.
- Danach erfolgt die eigentliche Regelung : akt. Wert kleiner LimitLow : Motor nach Rechts
  akt. Wert größer LimitHigh : Motor nach links, sonst Motor Stopp
- gemessen wird alle 111 mSek.

RobRechner

Ein plotterähnlicher Robot löst einfache Rechenaufgaben auf dem Taschenrechner



RechenPlotter : Die im Bild senkrechte Schnecke wird als X-Achse bezeichnet und durch einen Encodermotor an M1, C1 und I1 betrieben. Waagerecht Y-Achse mit Encodermotor (M2, C2, I2) An der Y-Schnecke hängt an einer Zahnstange mit Hubgetriebe ein Minimotor (M3 und I3) mit einem gefederten Taster. Der benutzte Taschenrechner muß mit Geschick auf der Bauplatte arretiert werden (Display nach oben ist eigentlich schöner). In jedem Fall müssen die Koordinaten für die Tasten 0 - 9 und "+", "=", "Clear" für den speziellen Aufbau mit Geschick ermittelt und in eine Tabelle eingetragen werden.



Auf START-Button drücken um den Rechenplotter auf Grundstellung zu fahren. Die Tastenbeschriftung wechselt dann auf Lösung. Eine Aufgabe eingeben (ohne Lösung) und auf Lösungsbutton Klicken. Der Rechenplotter marschiert los. Taschenrechnerergebnis mit hier angezeigten Ergebnis vergleichen.
Achtung COM4 ggf. anpassen.

Deklarationen

```
Imports FishFaceTX
Public Class RobRechnerSimpleVB
  Dim tx As New FishFace()
  Const xMotor As Mot = Mot.M1
  Const yMotor As Mot = Mot.M2
  Const taster As Mot = Mot.M3
  Const tEnd As Unv = Unv.I3

  Const txFull As Integer = 512
  Const xMax As Integer = 1040
  Dim xStart As Integer = 0
  Dim xDest As Integer = 0
  Const yMax As Integer = 1040
  Dim yStart As Integer = 0
  Dim yDest As Integer = 0
```

Tabelle mit Tastenpositionen

```
Dim TasPos(,) As Integer = {{940, 310}, {820, 310}, {820, 590},…..
  {940, 180}}
```

Taste betätigen

```
Private Sub TasterGo()
  tx.SetMotor(taster, Dir.Right)
  tx.WaitForInput(tEnd)
  tx.SetMotor(taster, Dir.Left)
  tx.Pause(555)
  tx.SetMotor(taster, Dir.Off)
End Sub
```

Tastermotor fährt nach unten bis Endtaster (auf die Rechnertaste) und dann 555 mSek wieder nach oben um freizukommen.

Anfahren der Home-Position

```
Private Sub MoveHome()
  xDest = 0
  yDest = 0
  tx.StartRobMotor(xMotor, Dir.Left, txFull, 9999)
  tx.StartRobMotor(yMotor, Dir.Left, txFull, 9999)
  tx.WaitForMotors(xMotor, yMotor)
  xStart = 0
  yStart = 0
End Sub
```

Hier mit Trick 17b : Die Motoren für X und Y werden nach links mit dem Ziel 9999 Schritte gestartet, das kann nie erreicht werden, der zugehörende Endtaster schlägt eher zu.

Anfahren einer Tastenposition

```
Private Sub MoveTo(ByVal x As Integer, ByVal y As Integer)
  Dim diff As Integer = 1
  Dim vorzX As Integer = 1
  Dim vorzY As Integer = 1
  If x > xMax Then xDest = xMax Else xDest = x
  If xDest > xStart Then
        tx.StartMotor(xMotor, Dir.Right, txFull, xDest - xStart)
        vorzX = 1
  ElseIf xStart - xDest > 0 Then
        tx.StartRobMotor(xMotor, Dir.Left, txFull, xStart - xDest)
```

```
        vorzX = -1
    End If
    If y > yMax Then yDest = yMax Else yDest = y
    If yDest > yStart Then
        tx.StartMotor(yMotor, Dir.Right, txFull, yDest - yStart)
        vorzY = 1
    ElseIf yStart - yDest > 0 Then
        tx.StartRobMotor(yMotor, Dir.Left, txFull, yStart - yDest)
        vorzY = -1
    End If
    diff = tx.WaitForMotor(xMotor)
    xStart = xDest + diff * vorzX
    diff = tx.WaitForMotor(yMotor)
    yStart = yDest + diff * vorzY
End Sub
```

Zunächst wird die maximal mögliche Position festgestellt und der Destwert ggf. gekürzt. Dann wird die zu fahrende Richtung bestimmt und aus aktueller (Start) Position und Dest-Position die Anzahl der zu fahrenden Schritte bestimmt und der Fahrauftrag gegeben. Die Motoren werden bei Erreichen des Ziels von einer zentralen Routine (MultiMediaTimer) gestoppt. Das WaitForMotor ist also nur eine Feststellung : Ziel erreicht. Eventl. Abweichungen vom Ziel werden in die aktuelle Position eingebracht.

Die Steuer-Routine

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles cmdAction.Click
    Dim OP1, OP2 As Integer
    Dim i, ptrTas As Integer
    Dim Tasten(24) As Integer
    Try
      If cmdAction.Text = "START" Then
        tx.OpenController(txtComName.Text)
        cmdEnde.Text = "&HALT"
        lblStatus.Text = "--- Fährt auf Home-Position ---"
        TasterGo()
        MoveHome()
        MoveTo(TasPos(12, 0), TasPos(12, 1))
        TasterGo()
        lblStatus.Text = "Gestartet"
        cmdEnde.Text = "ENDE"
        cmdAction.Text = "Lösung"
        cmdEnde.Focus()
       ElseIf cmdAction.Text = "Lösung" Then
        OP1 = txtOP1.Text
        OP2 = txtOP2.Text
        ptrTas = 0
        Tasten(ptrTas) = 12      ' --- Clear
        For i = 0 To txtOP1.Text.Length - 1
          ptrTas += 1
          Tasten(ptrTas) = txtOP1.Text.Substring(i, 1)
        Next
        ptrTas += 1
        Tasten(ptrTas) = 10          ' --- Plus
        For i = 0 To txtOP2.Text.Length - 1
          ptrTas += 1
          Tasten(ptrTas) = txtOP2.Text.Substring(i, 1)
        Next
        ptrTas += 1
        Tasten(ptrTas) = 11          ' --- Gleich
        cmdEnde.Text = "HALT"
        lblStatus.Text = "Kontrolle mit dem Taschenrechner"
        For i = 0 To ptrTas
```

```
            MoveTo(TasPos(Tasten(i), 0), TasPos(Tasten(i), 1))
            TasterGo()
          Next
          txtErgebnis.Text = OP1 + OP2
          lblStatus.Text = "OK, das war's"
          cmdEnde.Text = "ENDE"
        End If
      Catch tx As FishFaceException
        MsgBox(tx.Message)
      Catch ee As Exception
        lblStatus.Text = ee.Message
      End Try
    End Sub
```

OpenController / CloseController wie gewohnt, hier in einer try - catch Konstruktion um evtl. Fehler von TX-Controller / Modell und bei der Eingabe abzufangen.
Die größte Teil des Programms beschäftigt sich mit dem Ermitteln der anzufahrenden Tasten für die aktuelle Aufgabe.
Das tatsächliche Anfahren der Tasten geschieht in einer unscheinbaren, kleinen Schleife über die Methode MoveTo, die über die Liste der anzufahrenden Tasten (Tasten[]) auf die Tabelle der Tastenpositionen (TasPos[,]) zugreift.

Referenz

Programmrahmen

Anwendungen, die die FishFaceTX.DLL verwenden, enthalten eine Reihe von immer wiederkehrenden Elementen :

```
// ---(0)----
Imports FishFaceTX

// ---(1)---

Dim Tx As New FishFace()

// ---(2)---

Try
// ---(3)---
  tx.OpenController(COM-Name)
              .... Anwendungs-Code hier, bei größeren Anwendungen
           Methoden-Aufrufe .......
Catch txe As FishFaceException
// ---(4)---
       ... Ausgabe von Fehlermeldungen ...
       .... txe.Message;
Finally
// ---(5)---
  tx.CloseController()
End Try
```

(0) Die FishFace-Funktionen befinden sich im Namespace FishFaceTX der FishFaceTX.DLL. Für die FishFaceTX.DLL ist ein entsprechender Projektverweis erforderlich.

(1) Die Klasse FishFace muß entsprechend installiert werden.

(2) Der Try – Block fängt mögliche Fehler aus der Klasse FishFace ab (aber nur diese), man kann bei Bedarf weitere Catch-Blöcke hinzufügen, wenn man das tx.CloseController direkt hinter den catch-Block stellt, geht es auch ohne finally.

(3) Mit OpenController wird die Verbindung zum Interface hergestellt (am einfachsten gibt man hier den eigenen Anschluß angeben fest ein). (5) ft.CloseController() hebt die Verbindung wieder auf.

(4) Im catch-Block können Fehlernachrichten ausgegeben werden, wenn FishFace-Methoden eine Exception ausgelöst haben.

Auf Basis dieses Programmrahmens kann man nette kleine Testprogramme erstellen, z.B. zum probieren mit den Beispielen der Referenz.

Verwendete Variablenbezeichnungen

Die Variablen sind durchweg enums. Einige Angabe erfolgen als int, das wird besonders gekennzeichnet

Die Parameter-Angaben erfolgen – soweit nicht extra notiert – By Value

ctrId	Bezeichnung des zutreffenden Controllers (Ctr)
Direction	Drehrichtung eines Motors (Dir)
CounterNr	Name eines C-Einganges (Cnt)
InputNr	Name eines (Universal)I-Einganges (Unv)
LampNr	Name eines O-Ausganges ("halben"-M-Ausganges) (Out)
MotorNr	Name eines M-Ausganges (Mot)
mSek	Zeitangabe in MilliSekunden
OnOff	Ein/Ausschalten eines M-Ausganges (Dir)
Value	allgemeiner Integer-Wert
bool	Wahrheitswert true/false
Power	Leuchtstärke (0 - 512)
Speed	Motorgeschwindigkeit (0 - 512)

enum's

Verwendung zur Eingaben von Parametern bei den FishFace-Methoden.

Dir	Angabe der Drehrichtung ...
Ctr	Angabe des zugehörenden Controllers (Main oder Ext1…, Main kann entfallen)
Unv	Angabe der Nummer eines I-Einganges
Cnt	Angabe der Nummer eines C-Einganges
Mot	Angabe der Nummer eines M-Ausganges
Out	Angabe der Nummer eines O-Ausganges

Exceptions

FisFaceException

Allgemeine Exception in Zusammenhang mit Zugriffen auf den TX Controller.
MessageAufbau : Verursachende Methode.Fehler

Klasse FishFace

Enthalten in der Assembly FishFaceTX.DLL (Source-File : FishFaceTX.CS).
FishFace ist die Basisklasse der Assembly FishFaceTX.DLL. Verwendet wird der Namespace FishFaceTX.

Konstruktor

FishFace()
Ohne Parameter

Eigenschaften

bool	**NotHalt** Anmelden eines Abbruchwunsches (Default = false).
string	**Version** (get, static) Version der FishFaceTX.DLL

Methoden

ClearCounter

Löschen (0) des angegebenen Counters an einem C-Eingang

tx.**ClearCounter**([devId,] CounterNr)

Siehe auch : GetCounter

CloseController

Schließen der Verbindung zum Controller

tx.**CloseController**()

Siehe auch : OpenController

Finish

Feststellen eines Endewunsches (NotHalt, Escape [, Digitaler-Eingang])

bool = tx.**Finish**([InputNr])

Exception : ControllerProblem, KeinOpen. DoEvents

Siehe auch : GetInput

Beispiel :

```
Do
  Console.WriteLine("läuft")
  tx.Pause(2345)
Loop Until tx.Finish(Unv.I1)
```

Der Do .. Until Loop wird solange duchlaufen, bis entweder tx.NotHalt = true, die ESC-Taste gedrückt oder I1 = true wurde. Die Schleife wird mindestens einmal durchlaufen.

Alternativ :

```
Do Until tx.Finish(Unv.I1)
   Console.WriteLine("läuft")
  tx.Pause(2345)
Loop
Console.WriteLine("--- FINIS ---")
```

Die Schleife wird ggf. übersprungen.

GetAnalog

Feststellen eines Analogwertes an einem I-Eingang (NTC, Photowiderstand, Potentiometer).
Es wird der intern vorliegende (Raw)Wert ausgegeben.

Value = tx.**GetAnalog**([ctrId.] InputNr)

Exception : ControllerProblem, KeinOpen, DoEvent

Siehe auch : GetVoltage, GetDistance

Beispiel

```
Console.WriteLine(" NTC : " + tx.GetAnalog(Unv.I1).ToString())
```

WriteLine gibt den aktuellen Wert eines NTX am Universal-Eingang I1 aus.

GetCounter

Auslesen des Wertes des angegebenen Counters an einem C-Eingang

Value = tx.**GetCounter**([ctrId.] CounterNr)

Siehe auch : ClearCounter

Beispiel

```
Console.WriteLine("Counter für C2 : " & _
tx.GetCounter(Cnt.C2).ToString())
```

Der aktuelle Zählerstand, der dem C-Eingang C2 zugeordnet ist, wird ausgegeben.

GetDistance

Auslesen eines zu einem I-Eingang (UltraschallSensor) gehörenden Distanzwertes [cm].

Value = tx.**GetDistance**([ctrId.] InputNr);

Exception : ControllerProblem, KeinOpen, DoEvent.

Siehe auch

Beispiel

```
Dim Abstand As Integer = tx.GetDistance(Dev.Ext1, Unv.I1)
```

Der aktuelle Abstand eines DistanceSensors am ersten Extension Controller in cm zu einem Hindernis wird bestimmt.

GetInput

Auslesen des Zustandes des angegebenen I-Einganges(Taster, Reedkontakt,
PhotoTransistor(auch Eingänge C1 - C4, als I9 - I12)

bool = tx.**GetInput**([ctrId.] InputNr)

Exception : ControllerProblem, KeinOpen. DoEvents

Siehe auch : Finish, WaitForInput

Beispiel

```
If tx.GetInput(Unv.I1) Then
  ...
```

```
Else

End If
```

Wenn der I-Eingang I1 (Taster, PhotoTransistor, Reedkontakt ...) = True ist, wird der erste Block durchlaufen. Bei `Not ft.GetInput(Unv.I1)` wird der else-Zweig durchlaufen. Möglich ist auch If `(tx.GetInput(Unv.I1) = False)` Then ... oder
`If Not tx.GetInput(Unv.I1)) Then ...`

GetTrack

Auslesen des Zustandes des angegebenen I-Einganges(SpurSensor)

bool = tx.**GetTrack**([ctrId.] InputNr)
true bedeutet auf Spur (schwarzem Grund), False von der Spur (weißer Grund)

Exception : ControllerProblem, KeinOpen. DoEvents

Beispiel:
```
If tx.GetTrack(Unv.I1) Then
  ...
Else
  ...
End If
```

Wenn der I-Eingang I1 (einer der beiden Ausgänge eines SpurSensors) = True (auf Spur) ist, wird der erste Block durchlaufen. Bei `Not tx.GetTrack(Unv.I1)` wird der Else-Zweig (von Spur) durchlaufen.

GetVoltage

Feststellen des Spannungswertes des angegebenen I-Einganges.

Value = tx.**GetVoltage**([ctrId.] InputNr);

Exception : ControllerProblem, KeinOpen; DoEvents

Siehe auch : GetAnalog

Beispiel :
```
lblVolt.Text = tx.GetVoltage(Unv.I1).ToString()
```

Dem Label lblVolt wird der aktuelle Wert von I1 zugewiesen.

OpenController

Herstellen der Verbindung zum TX Controller. OpenController muß als erste Methode aufgerufen werden.

ft.**OpenInterface**(string COMname);

Exception : InterfaceProblem

Siehe auch : CloseController

Beispiel
```
Try
  ft.OpenController("COM4")
......
Catch txe As FishFaceException
  Console.WriteLine(txe.Message)
Finally
  ft.CloseController()
End Try
```

Herstellen der Verbindung zum TX-Controller am COM-Port COM4 (Ermittlung des richtigen COM-Ports über ROBO TX Test) ImFehlerfall wird (nach einigen Sekunden) der Text `'ControllerProblem.Open'` ausgegeben

Pause

Anhalten des Programmablauf für mSek MilliSekunden

tx.**Pause**(mSek)

Exception : KeinOpen; DoEvents; Abbrechbar

Beispiel
```
tx.SetMotor(Mot.M1, Dir.Left)
tx.Pause(1000)
tx.SetMotor(Mot.M1, Dir.Off)
```

Der Motor am M-Ausgang M1 wird für eine Sekunde (1000 MilliSekunden) eingeschaltet.

SetLamp

Setzen eines O-Ausganges (eines 'halben' M-Ausganges). Anschluß einer Lampe oder eines Magneten ... an einen Kontakt eines M-Ausganges und Masse.

tx.**SetLamp**([Dev,] Out, OnOff, Power)

- Power : Intensität des "Leuchtens", optional, default = 512.

Exception : ControllerProblem, KeinOpen

Siehe auch :

Beispiel
```
Const Gruen As Out = Out.O1, Gelb = Out.O2, Rot = Out.O3

  tx.SetLamp(Gruen, Dir.On)
  tx.Pause(2000)
  tx.SetLamp(Gruen, Dir.Off)
  tx.SetLamp(Gelb, Dir.On)
```

Die grüne Lampe an O1 und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an O2.

SetMotor

Setzen eines M-Ausganges (Motor). Die Motordrehzahl kann gewählt werden (Default = 512 (Full)), ebenso die Fahrstrecke in Anzahl Impulsen.

tx.**SetMotor**([ctrId.] MotorNr, Direction [, Speed])

Speed default 512

Exception : ControllerProblem, KeinOpen;

Siehe auch : SetLamp

Beipiel 1
```
tx.SetMotor(Mot.M1, Dir.Right, 512)
tx.Pause(1000)
tx.SetMotor(Mot.M1, Dir.Left, 400)
tx.Pause(1000)
tx.SetMotor(Mot.M1, Dir.Off)
```

Der Motor am M-Ausgang M1 wird für 1000 Millisekunden rechtsdrehend, volle Geschwindigkeit eingeschaltet und anschließend für 1000 MilliSekunden linksdrehend, etwa halbe Geschwindigkeit.

StartMotor

Starten eines "Encoder"Motors (echter oder normaler + Impulstaster) am M-Ausgang und zugehörenden C-Eingang. Der Motor läuft asynchron für die angegebene Anzahl von Zählimpulsen und wird dann selbsttätig abgeschaltet.

tx.**StartMotor**([ctrId.] MotorNr, Direction, Speed, ICount)

Beispiel

```
tx.StartMotor(Mot.M1, Dir.Left, 512, 123)
……
Dim diff As Integer = tx.WaitForMotors(Mot.M1)
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am C-Eingang C1 mit Geschwindigkeitsstufe Full(512) eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Schließlich wird auf das Erreichen von 123 Impulsen gewartet (der Motor kann dann schon abgeschaltet sein). Die Variable diff enthält eine evtl. Abweichung in Impulsen von der Zielvorgabe

StartRobMotor

Starten eines "Encoder"Motors (echter oder normaler + Impulstaster) am M-Ausgang und zugehörenden C-Eingang (gleiche Nummer). Der Motor läuft asynchron für die angegebene Anzahl von Zählimpulsen und wird dann selbsttätig abgeschaltet. Gegenüber StartMotor ist dem Motor ein Endtaster festzugeordnet (gleiche Nummer), der bei Dir.Left ausgewertet wird, Taster = true führt zum vorzeitigen Abbruch des Motors.

tx.**StartRobMotor**([ctrId,] MotorNr, Direction, Speed, ICount)

Beispiel

```
tx.StartMotor(Mot.M1, Dir.Left, 512, 9999)
tx.StartMotor(Mot.M2, Dir.Left, 512, 9999)
……
tx.WaitForMotors(Mot.M1, Mot.M2)
```

Die Motoren an M-Ausgang M1 / M2 werden für 9999 Impulse an C-Eingang C1 / C2 (also praktisch "endlos") mit Geschwindigkeitsstufe Full(512) eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Schließlich wird auf das Erreichen der Position Taster I1 / I2 gewartet (die Motoren können dann schon abgeschaltet sein). Genutzt wird das Zum Anfahren der "Home"Position eines Modells.

WaitForHigh

Warten auf einen false/true-Durchgang an einem digitalen I-Eingang

ft.**WaitForHigh**([ctrId.] InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForLow, WaitForCount, WaitForInput.

Beispiel

```
tx.SetMotor(Mot.M1, Dir.On)
tx.SetMotor(Mot.M2, Dir.Left)
tx.WaitForHigh(Unv.I1)
tx.SetMotor(Mot.M2, Dir.Off)
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband aus der Lichtschranke ausgefahren ist (die Lichtschranke wird geschlossen), dann wird abgeschaltet. Die Lichtschranke muß vorher false sein (unterbrochen).

WaitForInput

Warten, daß der angegebene digitale I-Eingang den vorgegebenen Wert annimmt.
(Default = true)

tx.**WaitForInput**([ctrId.] InputNr, OnOff)

OnOff ist optional, default = true

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar

Siehe auch : WaitForCount, WaitForLow, WaitForHigh.

Beispiel
```
tx.SetMotor(Mot.M1, Dir.Left)
tx.WaitForInput(Unv.I1)
tx.SetMotor(Mot.M1, Dir.Off)
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf I-Eingang = true gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer EndPosition.

WaitForLow

Warten auf einen true/false-Durchgang an einem ditigalen I-Eingang

tx.**WaitForLow**([ctrId.] InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForCount, WaitForInput, WaitForHigh.

Beispiel
```
tx.SetMotor(Mot.M1, Dir.On)
tx.SetMotor(Mot.M2, Dir.Left)
tx.WaitForLow(Unv.I1)
tx.SetMotor(Mot.M2, Dir.Off)
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband in die Lichtschranke einfährt (sie unterbricht), dann wird abgeschaltet. Die Lichtschranke muß vorher true sein (nicht unterbrochen).

WaitForMotor

Warten auf ein MotorReadyEreignis

tx.**WaitForMotor**([ctrId,] MotorNr)

MotorNr : Motor auf den gewartet werden soll. Zusätzlich wird eine evtl. Differenz zur Zielvorgabe zurückgegeben. Die kann dann in die Bestimmung der aktuellen Position einbezogen werden.

Exception : IControllerProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : StartMotor, StartRobMotor

Beispiel
```
Dim actPos As Integer = 112
tx.SetMotor(Mot.M4, Dir.Right, 400, 50)
------
int diff = tx.WaitForMotor(Mot.M4)
actPos = actPos + 50 + diff
```

Der Motor am M-Ausgang M4 wird rechtsdrehend mit halber Geschwindigkeit für 50 Impulse gestartet. Der Motor beendet sich selber. Die aktuelle Position ergibt sich aus der alten, der vorgegebenen Fahrstrecke und einer evtl. Überschreitung der Fahrstrecke.

WaitForMotors

Warten auf ein MotorsReadyEreignis

tx.**WaitForMotors**([ctrId.] MotorNr, ...)

MotorNr(Nr) : Liste von M-Ausgängen in beliebiger Reihenfolge auf die gewartet werden soll. Gewartet wird auf MotorStatus = Aus für die betreffenden M-Ausgänge gewartet.

Exception : IControllerProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : StartMotor, StartRobMotor

Beispiel

```
tx.SetMotor(Mot.M4, Dir.Left, 400, 50)
fx.SetMotor(Mot.M3, Dir.Right, 512, 40)
-----
tx.WaitForMotors(Mot.M4, Mot.M3)
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehend mit voller Geschwindigkeit für 40 Impulse. Mit WaitForMotors wird auf das Erreichen der Zielposition gewartet. Das kann schon vorher geschehen sein, die Motoren beenden sich selber.

Allgemeine Anmerkungen

Die Methoden erwarten ein vorhergehendes OpenController. Ggf. wird eine entsprechende Exception ausgelöst. Sie enthalten meist ein **DoEvents** um das Programm unterbrechbar zu machen. Wird im Ablauf ein InterfaceProblem festgestellt, wird eine entsprechende **Exception** ausgelöst.

Das Schalten von M- bzw. O-Ausgängen über SetMotor / SetLamp gilt stets bis zum nächsten SetMotor/SetLamp. Also SetLamp(Out.O1, Dir.On) … SetLamp(Out.O1, Dir.Off), entsprechend für SetMotor.

Die StartMotor/StartTwin-Methoden sind **asynchron** d.h. der oder die angesprochenen Motoren werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter und beenden sich nach Erreichen der vorgegebenen Position selber.

Die Wait-Methoden koordinieren den Motorlauf mit dem Ablauf des Programms.Sie halten den weiteren Progammablauf an, bis das Waitziel (Ablauf Zeit, erreichte Position, Tasterstellung ...) erreicht ist d.h. sie synchronisieren den Programmablauf wieder.

Die längerlaufenden Methoden sind abbrechbar. Das geschieht manuell durch Drücken der ESC-Taste oder im Programm durch Setzen der Eigenschaft NotHalt = true (z.B. über einen Button).

Bei der Beschreibung der Methoden wird das unter dem Stichwort Exception angegeben.

Anmerkungen zu VB

Programmrahmen

Aufbau eines ftComputing-Programms

1. Try – Catch(FishFaceException e) (- Finally) Block
   sollte die gesamte Anwendung umfassen. Wenn mehr Detailierung erforderlich ist, natürlich mehr, ggf. auch weitere Catch – Klauseln.
2. OpenController – CloseController
   umschließt die Anwendung. Häufig reicht der feste "COMxx" Eintrag für den einzigen ROBO TX Controller an USB. Bei Betrieb über Bluetooth ist ein anderer COM-Name fällig. Hilfestellung bei der Namensfindung bieten RoboTxText oder das mitgelieferte RoboTXdevs.
3. Programmabbruch bei Fehlfunktionen, das Modell kann sonst "gegen die Wand fahren" Die Wait.. Methoden können durch die ESC-Taste am Keyboard abgebrochen werden. Das gleiche tut die Eigenschaft NotHalt = true, der man auf der Form eine Button zuordnen sollte.
4. Sperren von Buttons und des (x) rechts oben um ein Programmende erst nach Beenden aller Interna zu erlauben.
5. `Do ... Loop Until ft.Finish()`
   "Endlos"-Schleife, die durch Esc-Taste und NotHalt = true abgebrochen wird, ist nützlich bei Anwendungen, die auf Taste am Interface warten oder einen Funktions-Block wiederholen. Ein in Finish eingebautes Application.DoEvents sorgt für die Unterbrechbarkeit der Bedieneroberfläche.

Anlegen eines Console-Programmes

Beschrieben wird der Ablauf für VB 2005 :

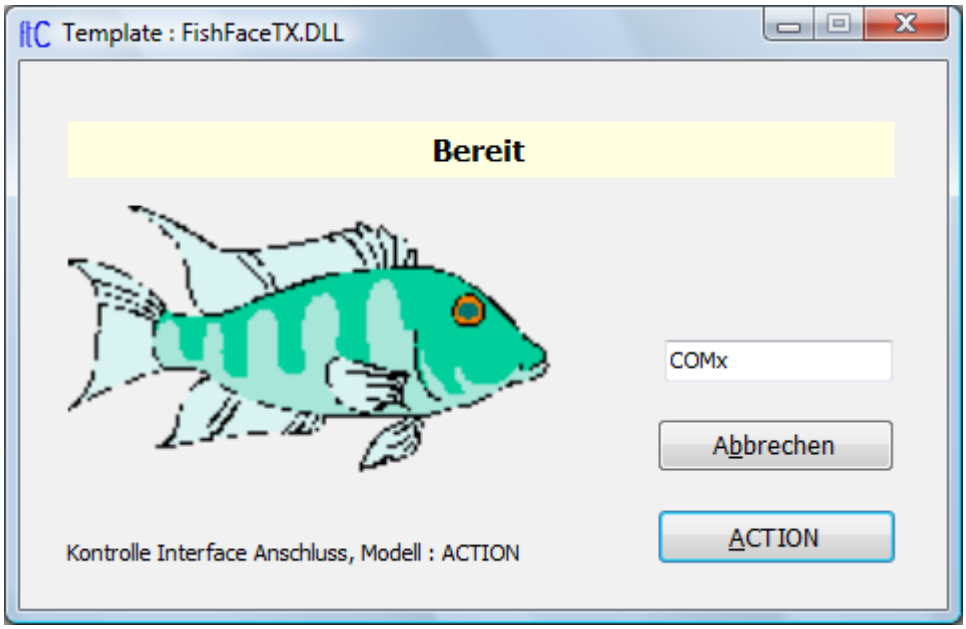
1. Menü Datei | Neues Projekt... | Konsolenanwendung
2. Projektnamen wählen : FishTXConsole
3. Ggf. Program.cs umbenennen
   Projektmappen Explorer : FishTXConsole
4. Projektmappen Explorer : Verweise
   Hinzufügen : FishFaceTX.DLL (über Tab Durchsuchen oder Aktuell)
5. In FishTXConsole.vb
   `Imports FishFaceTX` ergänzen
6. Speichern
   Projektmappenverzeichnis anlegen, nicht erforderlich, wenn nur Einzelprojekt (man spart dann eine Verzeichnis-Ebene).

Anlegen eines Windows-Programmes

Beschrieben wird der Ablauf für VB 2005 Express :

1. Menü Datei | Neues Projekt .. | Windows Anwendung auswählen
2. Projektnamen wählen : BeispielFishFace
3. Projektmappen Explorer : Form1.vb umbenennen im Feld Eigenschaften Name : BeispielFishFace.vb
   Achtung : vb muß klein geschrieben werden.
4. Projektmappen Explorer : Verweise
   Hinzufügen : FishFaceTX.DLL (über Tab Durchsuchen oder Aktuell)
5. In der Source :
   `Imports FishFaceTX` ergänzen.
7. Speichern
   Projektmappenverzeichnis anlegen, nicht erforderlich, wenn nur Einzelprojekt (man spart dann eine Verzeichnis-Ebene).

Vorlagen



Vorlage : FishTXWindows

Das Paket FishFaTXVB.zip enthält ein Verzeichnis Templates in dem sich eine Reihe von Projekten befinden, die sich gut als Programmrahmen für das Ausprobieren von Codeausschnitten dieses Handbuchs, aber auch für das Ausprobieren eigener Ideen eignen. Um sie als Vorlagen auf der eigenen VB 2005 Installation nutzen zu können, sind sie vorher noch als Vorlagen zu speichern. Sie erscheinen dann anschließend in dem Auswahlfenster für neue Projekte.

FishTXConsole

```
Imports FishFaceTX
Module FishTXConsoleVB1
   Dim tx As New FishFace()
' Routine mit der eigentlichen Anwendung, hier ein einfacher Blinker
'       Weitere Sub's sind möglich
Sub Action()
  Do
        tx.SetLamp(Out.O5, Dir.On)
        tx.Pause(555)
        tx.SetLamp(Out.O5, Dir.Off)
        tx.Pause(333)
   Loop Until tx.Finish()
End Sub
#Region "--- ProgramControl ---"
……
#End Region
End Module
```

Ist eine ganz einfache Konsolen-Anwendung. Mit einem Try – Catch – Finally Block (In Region ProgramControll) zum Abfangen von FishFace-Fehlern und einem Console.WriteLine für Programmausgaben.
Das OpenController auf eigenen Bedarf anpassen.

FishTXWindows

Für Anwendungen mit der Klasse FishFace. Für Status-Anzeigen ist lblStatus.Text vorgesehen. Der ComName kann über eine Textfeld eingegeben werden.

Programm-Struktur

```
Imports FishFaceTX
Public Class FishTXWindowsVB1
   Dim tx As New FishFace()
  Private Sub Action()
' Routine für den Modellbetrieb, kann weitere Subs aufrufen ----
    Do
      ' Endlosschleife, Abbruch durch HALT-Button oder ESC-Taste,
      ' kann gelöscht werden
    Loop Until tx.Finish()
  End Sub
#Region "--- ProgramControl ---"
…..
#End Region
End Class
```

Hier ist die enthaltene #region zusammengeklappt, das Programm sieht dann verblüffend übersichtlich aus.

Methode **Action** : nimmt die eigentliche Anwendung auf. Die do-Schleife kann auch gelöscht werden.

#region --- Programm-Kontrolle --- (jetzt aufgeklappt) : mit dem Steuerungs-Code für Start und Beenden der Anwendung.

Das Programm verfügt über folgende Controls :

- lblStatus : Label zur Anzeige des Programm-Status
- txtCOM : Zur Eingabe des aktuellen COM-Namens für tx.OpenController.

- cmdEnde : Button zu Abbrechen / Beenden des Programms. Die Beschriftung wechselt entsprechend.
- cmdAction : Button zum Start der Anwendung, während des Ablaufs der Anwendung disabled. Mit Click-Routine cmdAction_Click.
- lblHinweis : Label mit Bedienungshinweisen.

```
#Region "--- ProgramControl ---"
Private Sub cmdAction_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs)Handles cmdAction.Click
' --- Open ROBO TX Controller an USB oder über Bluetooth ---
'     während des Ablaufs von Action gesperrt
  Try
        tx.OpenController(txtComName.Text)
        cmdAction.Enabled = False
        cmdEnde.Text = "&HALT"
        lblStatus.Text = "--- Bei der Arbeit ---"
        Action()                ' --- Nutz-Routine für die Anwendung
  Catch tx As FishFaceException
        MsgBox(tx.Message)
  Finally
        tx.CloseController()
        cmdAction.Enabled = True
        cmdEnde.Text = "&ENDE"
        lblStatus.Text = "--- Auf ein Neues ---"
        cmdEnde.Focus()
  End Try
End Sub

……
#End Region
```

cmdAction enthält einen Try – Catch – Finally Block, der die Fehler aus FishFaceTX abfängt. Das OpenController öffnet, nach Vorgabe von txtComName, den ROBO TX Controller. Nach erfolgreichem Open wird die eigentliche Anwendung in Action(); gestartet.

Bei Open-Fehlern und Fehlern in Action() wird der Catch-Teil des Blocks aufgerufen , die Fehlernachricht wird angezeigt, cmdAction wird beendet.

Finally schließt die Interface-Verbindung wieder und rückt die Buttons gerade.

```
Private Sub cmdEnde_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles cmdEnde.Click
' --- Programmende nur, wenn Action nicht läuft ---
  If cmdEnde.Text = "&HALT" Then tx.NotHalt = True Else Me.Close()
End Sub
```

Die Click-Routine zu cmdEnde löst bei Beschriftung mit &HALT ein NotHalt = True aus, d.h. die in Action() oft vorhandene do-Schleife "rauscht durch" d.h. alle Wait.. Methoden und das Finish der do-Schleife werden beendet, die Schleife und damit Action() ist zu Ende.

Bei einer anderen Beschriftung von cmdEnde (Abbrechen, ENDE) wird das gesamte Programm beendet.

Ein Betätigen der ESC-Taste hat die gleiche Wirkung wie die Betätigung von cmdEnde.

```
Private Sub frmClosing(ByVal sender As System.Object, _
  ByVal e As System.Windows.Forms.FormClosingEventArgs) _
  Handles MyBase.FormClosing
' --- Wenn Action läuft, wird ein Programmabbruch über (x)
'     unterbunden ---
  If cmdEnde.Text = "&HALT" Then e.Cancel = True
End Sub
```

formClosing wird z.B. bei einem Click auf (x) rechts oben aufgerufen. Es läßt ein Programmende nur zu, wenn der cmdEnde-Button nicht mit &HALT beschriftet ist.

Erstellen Vorlagen

Am einfachsten ist es, die obenaufgezählten Beispielprogramme (Programmrahmen) in Vorlagen zu konvertieren :

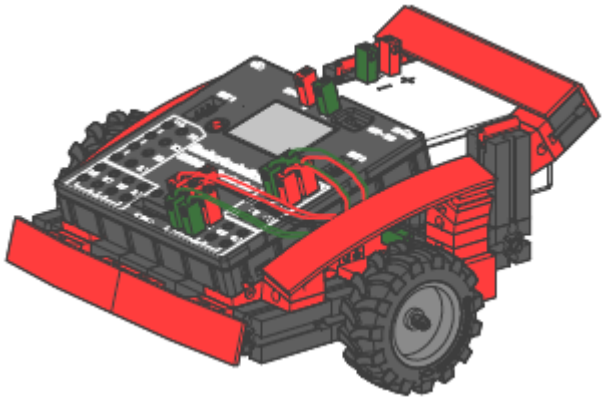
1. Das Projekt wie normal laden
2. An den eigenen Geschmack anpassen, testen
3. Menü Datei | Volage Exportieren
   Name z.B. FishFaceProjekt
4. Details :
   Symbol vergeben
   Beschreibung eingeben
   Fertigstellen

Klassenausflug

Allgemeines

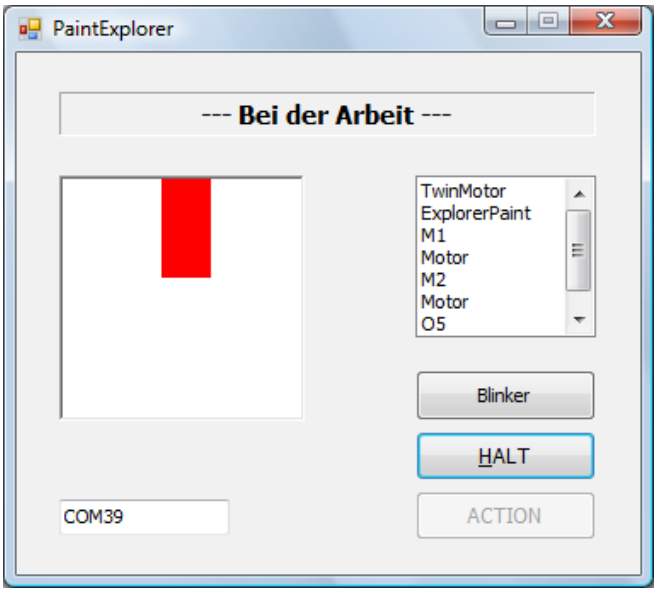
Hier einige Anmerkungen zum Umgang mit eigenen und importierten Klassen.

Das Modellbeispiel dazu ist ein Robo Explorer, gesteuert über den Robo TX Controller.
Linker Motor M1,
Rechter Motor M2,
Lampe (manchmal) O5



Für die ersten Tests nimmt man aber lieber eine Grundplatte mit zwei Motoren und Rädern irgendwie am Motor und dann noch eine Lampe. Das reicht.

Ziel des Abschnitts ist eine Einführung in den Umgang mit Klassen am Beispiel der Steuerung des Explorers durch Mausbewegungen auf einem PictureControl



einer WindowsForm. Dazu wird die PictureBox in neun Steuerungsbereiche geteilt : LinksVor, Vor, RechtsVor, LinksDrehen, Halt, RechtsDrehen, LinksRück, Rück, RechtsRück. Die Position innerhalb eines Bereiches bestimmt die Drehzahl der Motoren.

Das Programm selber basiert auf dem mitgelieferten WindowsTemplate.

Eingesetzt wird die FishFaceTX.DLL, Basis ist die Methode SetMotor(Mot, Dir, Speed)

Gezeigt werden fünf Varianten zum gleichen Thema, sie sind in der Projektmappe MouseExplorer untergebracht :

1. **MouseExplorer** : Alles wird in der Hauptform erledigt. Zugriff über die Klasse FishFace
2. **KlasseExplorer** : Die direkten Zugriffe auf die Motoren wurden in der Klasse Explorer zusammengefasst (Zugriff über Klasse FishFace)
3. **FishExplorer** : Wie 2. Hier wird aber die Klasse ExplorerFish von der Klasse FishFace abgeleitet (Inherited). Gesamt Zugriff auf den TX Controller über Klasse ExplorerFish.
4. **DevExplorer** : Alles wird in der Hauptform erledigt (wie 1.), aber es werden die Klassen FishControl (Steuerung des gesamten TX Controllers) und die Klasse TwinMotor (Simultane Steuerung von zwei Motoren mit Methoden wie RunForward, TurnLeft) eingesetzt.
5. **PaintExplorer** : Mit der Klasse ExplorerPaint (Inherited TwinMotor), die auch den Zugriff auf die aktuell Mausposition in die Klasse einbezieht (einschl. EventRoutinen).

MouseExplorer

Alles wird in der Klasse MouseExplorer der Hauptform erledigt.

Mitspieler

```
  Dim tx As New FishFace()
  Dim canvas As Graphics
```

Das Objekt tx für den Zugriff auf den TX Controller (Verweis auf FishFaceTX.DLL und `Imports FishFace` erforderlich). Mit `tx.OpenController` und und `tx.CloseController` in dem Click-Routine des ACTION-Buttons

canvas als Spielwiese für die Maus.

```
Private Sub MouseExplorer_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
        canvas = picMouse.CreateGraphics()
        mitte = picMouse.Height / 5
        mitteOben = (picMouse.Height - mitte) / 2
        MitteUnten = (picMouse.Height + mitte) / 2
        mitteLinks = (picMouse.Width - mitte) / 2
        mitteRechts = (picMouse.Width + mitte) / 2
        mouseX = mitteLinks + 1
        mouseY = mitteOben + 1
  End Sub
```

zuweisen des Graphics-Objektes der PictureBox picMouse zu canvas. Besetzen der Koordinaten für die Steuerungsbereiche. mouseX, mouseY erhalten hier einen Platz im Mittelfeld (Halt) um einen Sofortstart zu unterbinden.

```
Private Sub picMouse_MouseMove(ByVal sender As System.Object, ByVal
e As System.Windows.Forms.MouseEventArgs) Handles picMouse.MouseMove
        mouseX = e.X
        mouseY = e.Y
End Sub
```

```
Private Sub picMouse_MouseLeave(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles picMouse.MouseLeave
        mouseX = mitteLinks + 1
        mouseY = mitteOben + 1
End Sub
```

Im MausMove-Ereignis wird die aktuelle Mausposition in globale Variable überstellt. Beim Verlassen (MouseLeave-Ereignis) wird die Maus auf Halt gestellt.

```
Private Function Speed(ByVal pos) As Integer
  pos = IIf(pos > mitteRechts, pos - mitteRechts, mitteLinks - pos)
  If pos > mitteOben * 0.9 Then Return 512
  If pos > mitteOben * 0.6 Then Return 444
  Return 255
End Function
```

Eine allgemeine Routine um die aktuellen Mauskoordinaten in Geschwindigkeitswerte 255 - 512 umzurechnen (langsamer macht wenig Sinn).

```
Private Sub Action()
  Do
        canvas.Clear(Color.White)
        If mouseY < mitteOben And mouseX < mitteLinks Then 'LinksVor
              tx.SetMotor(Mot.M1, Dir.Left, Speed(mouseX))
              tx.SetMotor(Mot.M2, Dir.Left, Speed(mouseY))
              canvas.FillRectangle(Farbe, 0, 0, mitteLinks, _
                                   mitteOben)
        ElseIf mouseY < mitteOben And mouseX < mitteRechts Then 'Vor
              tx.SetMotor(Mot.M1, Dir.Left, Speed(mouseY))
              tx.SetMotor(Mot.M2, Dir.Left, Speed(mouseY))
              canvas.FillRectangle(Farbe, mitteLinks, 0, _
                                             mitte, mitteOben)
        ElseIf mouseY < mitteOben Then ' RechtsVor
--------
        End If
        tx.Pause(333)
    Loop Until tx.Finish()
  End Sub
```

Die Action Routine in einer "EndlosSchleife". Sie fragt alle 333 msec die aktuelle Mausposition ab und bestimmt daraus den zugehörenden Steuerungsbereich. Die Motoren werden dann entsprechend gestartet und der erkannte Steuerungbereich wird rot markiert.

Resümee

Für das anstehende Problem eine angemessene Lösung. Bequem, da alle benötigten Komponenten im direkten Zugriff sind. Beim Ausbau des Explorers auf Hindernis-Erkennung….. kann es aber unübersichtlich werden.

KlasseExplorer

Die direkten Zugriffe (SetMotor) auf die Explorermotoren wurden in die Klasse Explorer gelegt.

Mitspieler

```
  Dim tx As New FishFace()
  Dim ex As New Explorer(tx)
  Dim canvas As Graphics
```

Das Object tx wie bisher. Neu das Objekt ex zum Betreiben der Explorermotoren, canvas wie gehabt.

```
Private Sub Action()
  Do
    canvas.Clear(Color.White)
    If mouseY < mitteOben And mouseX < mitteLinks Then
      ex.LinksVor(Speed(mouseX), Speed(mouseY))
      canvas.FillRectangle(Farbe, 0, 0, mitteLinks, mitteOben)
    ElseIf mouseY < mitteOben And mouseX < mitteRechts Then
       ex.Vor(Speed(mouseY))
       canvas.FillRectangle(Farbe, mitteLinks, 0, mitte, mitteOben)
    ElseIf mouseY < mitteOben Then
--------
    End If
    tx.Pause(333)
  Loop Until tx.Finish()
End Sub
```

Aufbau wie gehabt, anstelle der direkten Motorbefehle hier aber Methoden der Klasse Explorer. Die Action wird dadurch deutlich dünner.

```
Public Class Explorer
  Private ff As FishFace
  Sub New(ByVal tx As FishFace)
        ff = tx
  End Sub
  Public Sub LinksVor(ByVal speedX As Integer, _
      ByVal speedY As Integer)
        ff.SetMotor(Mot.M1, Dir.Left, speedX)
        ff.SetMotor(Mot.M2, Dir.Left, speedY)
  End Sub
  Public Sub Vor(ByVal speed As Integer)
        ff.SetMotor(Mot.M1, Dir.Left, speed)
        ff.SetMotor(Mot.M2, Dir.Left, speed)
  End Sub
---------
End Class
```

Mit einer eigenen Objektvariablen für den Zugriff auf das FishFace-Objekt. Sie wird im Konstruktor (Sub New) übergeben. Die Methoden entsprechen den im "alten" Action genutzten. Speed wird weiterhin in KlasseExplorer bestimmt und durch gereicht.

Resümee

Darstellung der Programmoberfläche und Steuern des Explorers wurden getrennt. Macht die Sache klarer, aber bei der vorliegenden Programmgröße auch komplexer.

FishExplorer

Überarbeitung der Lösung des KlasseExplorers. Die Klasse Explorer (jetzt ExplorerFish) erbt (`Inherits FishFace`) damit kann auf alle FishFace-Funktionen über ein Object zugegriffen werden (Dim ex As New ExplorerFish).

Die Verbindung zum TX Controller wird jetzt über ex.OpenController / ex.CloseController aufgebaut.

Die Klasse ExplorerFish ist weitgehend so geblieben. Der Zugriff auf eine externe FishFace-Instanz entfällt so :

```
Public Class ExplorerFish
  Inherits FishFace

  Dim lichtAn As Boolean = False

   Public Sub LinksVor(ByVal speedX As Integer, _
       ByVal speedY As Integer)
     SetMotor(Mot.M1, Dir.Left, speedX)
     SetMotor(Mot.M2, Dir.Left, speedY)
   End Sub
   Public Sub Vor(ByVal speed As Integer)
     SetMotor(Mot.M1, Dir.Left, speed)
     SetMotor(Mot.M2, Dir.Left, speed)
   End Sub
-----------
   Public ReadOnly Property IsFahrLicht() As Boolean
     Get
       Return lichtAn
     End Get
   End Property
   Public Sub FahrLicht(ByVal OnOff As Boolean)
     If OnOff Then
        SetLamp(Out.O5, Dir.On)
        lichtAn = True
     Else
        SetLamp(Out.O5, Dir.Off)
        lichtAn = False
     End If
   End Sub
 End Class
```

Hinzugekommen ist da noch eine Methode zum Schalten des Fahrlichtes am Explorer (FahrLicht) und eine Eigenschaft (IsFahrLicht, ReadOnly), über den Schaltzustand Auskunft gibt. Dafür gibts dann auf der MainForm einen extra Button und eine Click-Routine

```
Private Sub cmdFahrlicht_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles cmdFahrlicht.Click
  ex.FahrLicht(Not ex.IsFahrLicht)
End Sub
```

Ausgebildet als Wechselschalter.

DevExplorer

Nutzung weiterer Klasse aus FishFaceTX.DLL ansonsten wird alles im MainWindow erledigt wie in MouseExplorer :

Mitspieler

```
Dim fc As New FishControl()
Dim tm As New TwinMotor(fc, Ctr.Main, Mot.M1, Mot.M2)
Dim canvas As Graphics
```

Das Object fc der Klasse FishControl zur Überwachung des Zugriffs auf den TX Controller

Das Object tm der Klasse TwinMotor für den Zugriff auf die Explorer-Motoren.

Und canvas wie gehabt.

In cmdAction_Click gibt es jetzt ein fc.Connect / fc.DisConnect sonst bleibt es da ruhig.

Die Sub Action entspricht weitgehend der Lösung von KlasseExplorer. Die Methoden haben jetzt englische Namen (zwecks internationalem Einsatz).

```
Private Sub Action()
      Do
        canvas.Clear(Color.White)
        If mouseY < mitteOben And mouseX < mitteLinks Then
          tm.RunForward(Speed(mouseX), Speed(mouseY))
          canvas.FillRectangle(Farbe, 0, 0, mitteLinks, mitteOben)
        ElseIf mouseY < mitteOben And mouseX < mitteRechts Then
          tm.Forward(Speed(mouseY))
          canvas.FillRectangle(Farbe, mitteLinks, 0, mitte, _
                                                  mitteOben)
        ElseIf mouseY < mitteOben Then
      ------
        End If
        fc.Pause(333)
      Loop Until fc.Finish()
End Sub
```

Resümee

Eine recht bequeme Lösung, da durch die Klasse TwinMotor einiges gekapselt wird.

PaintExplorer

Mit einer Klasse ExplorerPaint (`Inherits TwinMotor`), die auch die Positionsbestimmung der Maus in picMouse übernimmt :

```
Imports FishFaceTX

Public Class ExplorerPaint
  Inherits TwinMotor
  Private bl As Lamp
  Private blinkt As Boolean = False
  Private WithEvents steuerFlaeche As PictureBox
  Private canvas As Graphics
  Private mouseX, mouseY As Integer
  Private mitte, mitteOben, MitteUnten, mitteLinks, _
                                  mitteRechts As Integer
  Private Farbe As New SolidBrush(Color.Red)

 Public Sub New(ByVal steuer As PictureBox, _
                             ByVal fish As FishControl)
   MyBase.New(fish, Ctr.Main, Mot.M1, Mot.M2)
   bl = New Lamp(fish, Ctr.Main, Out.O5)
   steuerFlaeche = steuer
   canvas = steuer.CreateGraphics()
   mitte = steuer.Height / 5
   mitteOben = (steuer.Height - mitte) / 2
   MitteUnten = (steuer.Height + mitte) / 2
   mitteLinks = (steuer.Width - mitte) / 2
   mitteRechts = (steuer.Width + mitte) / 2
   mouseX = mitteLinks + 1
   mouseY = mitteOben + 1
 End Sub
```

Für picMouse gibt es hier eine Objectvariable steuerFläche, die über den Konstruktor durch gereicht wird, sie wird dann noch für canvas angezapft.

Das zugehörende FishControl-Object für den TX-Controller wird über den Parameter fish durchgereicht und bei der Instanzierung der BlinkerLampe bl verwendet. Die Klasse selber wird in der WindowsForm instanziert.

```
 Private Function Speed(ByVal pos) As Integer
   pos = IIf(pos > mitteRechts, pos - mitteRechts, mitteLinks - pos)
   If pos > mitteOben * 0.9 Then Return 512
   If pos > mitteOben * 0.6 Then Return 444
   Return 255
 End Function
```

wie gehabt, jetzt hier angesiedelt.

```
 Private Sub picMouse_MouseMove(ByVal sender As System.Object, _
   ByVal e As System.Windows.Forms.MouseEventArgs) _
   Handles steuerFlaeche.MouseMove
     mouseX = e.X
     mouseY = e.Y
 End Sub
 Private Sub picMouse_MouseLeave(ByVal sender As System.Object, _
   ByVal e As System.EventArgs)  Handles steuerFlaeche.MouseLeave
     mouseX = steuerFlaeche.Width / 2 + 2
     mouseY = steuerFlaeche.Height / 2 + 2
 End Sub
```

Auch von der WindowsForm verlagert.

```
Public Sub DriveExplorer()
  canvas.Clear(Color.White)
  If mouseY < mitteOben And mouseX < mitteLinks Then
    RunForward(Speed(mouseX), Speed(mouseY))
    canvas.FillRectangle(Farbe, 0, 0, mitteLinks, mitteOben)
  ElseIf mouseY < mitteOben And mouseX < mitteRechts Then
    Forward(Speed(mouseY))
    canvas.FillRectangle(Farbe, mitteLinks, 0, mitte, mitteOben)
  ElseIf mouseY < mitteOben Then
    RunForward(Speed(mouseX), Speed(mouseY))
    canvas.FillRectangle(Farbe, mitteRechts, 0, mitteLinks, _
         mitteOben)
  ElseIf mouseY < MitteUnten And mouseX < mitteLinks Then
  ------
  End If
End Sub
```

Für die gesamte Fahrerei bietet ExplorerPaint jetzt nur noch eine Methode an, sie entspricht in etwa der Sub Action aus KlasseExplorer

```
Public Sub Blinklicht(ByVal OnOff As Boolean)
  If OnOff Then
    bl.BlinkingOn(333)
    blinkt = True
  Else
    bl.BlinkingOff()
    blinkt = False
  End If
End Sub
Public ReadOnly Property IsBlinking() As Boolean
  Get
    Return blinkt
  End Get
End Property
End Class
```

Damit es nicht zu langweilig wird, auf Basis des Objektes bl der Klasse `Lamp` einen Gelblicht-Blinker und zugehörende Property. Da VB.NET nur die einfache Vererbung kennt, muß hier instanziert werden (im Konstruktor).

Die MainForm sieht dann geradezu mickrig aus (wenn man die ProgrammControl Region einklappt) :

```
Public Class PaintExplorer

  Private fc As New FishControl()
  Private ex As ExplorerPaint

Private Sub Action()
  Do
    ex.DriveExplorer()
    fc.Pause(333)
  Loop Until fc.Finish()
End Sub
"--- ProgramControl ---"
Private Sub cmdBlinklicht_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdBlinklicht.Click
  ex.Blinklicht(Not ex.IsBlinking)
End Sub
End Class
```

Es wird hier nur noch in einer Schleife gefahren. Und dann gibt es da noch einen Button zum Schalten des Gelbblinkers.

Als Nachtrag noch die Liste der beteiligten Komponenten, die schon im Bild ganz oben zu sehen war :

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdAction.Click
  Try
    fc.Connect(txtComName.Text)
    ex = New ExplorerPaint(picMouse, fc)
    cmdAction.Enabled = False
    cmdEnde.Text = "&HALT"
    cmdBlinklicht.Enabled = True
    lstDev.Items.Clear()
    For i As Integer = 0 To fc.DeviceCount - 1
      lstDev.Items.Add(fc.Item(i).ConnectorName)
      lstDev.Items.Add(fc.Item(i).DeviceName)
    Next
  -----
  End Try
End Sub
```

Zuerst einen Blick auf den Start :
- fc.Connect stellt die Verbindung zum TX Controller her
- ex = New ExplorerPaint(picMouse, fc) instanziert den Explorer
- Das Blinklicht wird freigegeben.

Und dann wird gelistet über Item Eigenschaft des fc-Objects.

Zu sehen sind in der Liste zunächst der Name des angeschlossenen Device-Objects TwinMotor und dessen Erbe (ExplorerPaint) sowie die belegten Anschlüsse am Interface : M1 mit Device-Object Motor und M2 dto. Etwas gequetscht O5 mit dem Device-Object Lampe.

Resümee

Eine technisch Interessante Lösung, die praktisch wohl nur sinnvoll ist, wenn diese Kombination tatsächlich häufiger genutzt wird.