

Kommunikation mit externen Geräten

Carsten Knoll

TU Dresden, Institut für Regelungs- und Steuerungstheorie

Interdisziplinärer Python "Sommerkurs"

27. Juni 2011

- Ziel:
 - Schnittstellen ↔ Module
 - Empfehlungen

- Aufbau:

Einleitung

Schnittstellen

- Serielle Schnittstelle
- parallele Schnittstelle
- GPIB
- Ethernet

DLL-Treiber nutzen

Anwendungsbeispiele

- USB Missile Launcher
- Mobiler Roboter (Arduino-Plattform)



Umsetzungshinweise

- 1 Vorbemerkungen
- 2 Einleitung**
- 3 Schnittstellen
- 4 Fertige Treiber Nutzen (DLLs)
- 5 Anwendungsbeispiele
- 6 Umsetzungshinweise

Kommunikation womit?

- Messgeräte
- Funktionsgeneratoren
- Steuerungen für Positioniertische
- Optische Komponenten (Lampen, Laser, Filter, ...)
- Temperaturregler
- Mikrocontroller („ μ C“)
- Anderer Rechner
- ...

Kommunikation womit?

- Messgeräte
 - Funktionsgeneratoren
 - Steuerungen für Positioniertische
 - Optische Komponenten (Lampen, Laser, Filter, ...)
 - Temperaturregler
 - Mikrocontroller („ μ C“)
 - Anderer Rechner
 - ...
-
- Hintergrund: Eigenen Erfahrungen ( Fraunhofer_{ILR},  iapp)

- Platzhirsch: LabVIEW (National Instruments)
- Umfangreiche (Treiber-)Bibliothek
- Datenfluss orientiert → grafisches Programmieren
- Intuitive Parallelisierung, sehr einfaches Erzeugen von GUIs

Nachteile (persönliche Meinung!)

- Haupteingabe per Maus (=Informations-Nadelör verglichen mit Tastatur)
- Modularisierung/Kapselung aufwendiger
- Tendiert schnell zu Unübersichtlichkeit
- Permanente Platzprobleme → Sparszwang auch bei Kommentaren
- Wartbarkeit ↘, Erweiterbarkeit ↘
- ...

So soll es aussehen

The image displays a LabVIEW project window titled "Read and Display" and its corresponding graphical user interface (GUI).

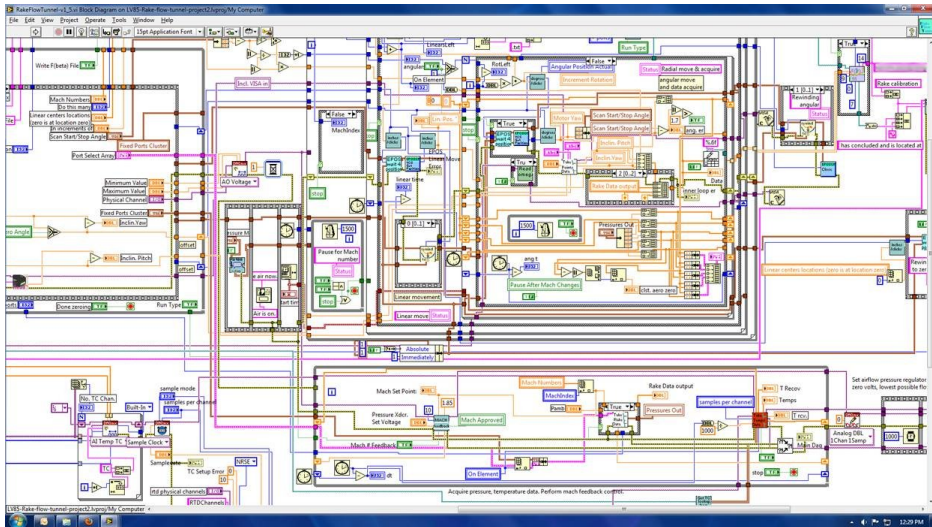
Block Diagram (Bottom): The block diagram shows the logic for the GUI. It starts with a "Simulate Signal" block that takes "amplitude of signal" and "frequency of sine signal" as inputs and outputs "error out", "Amplitude", "error in (no error)", "Phase", and "Reset Signal". The "Amplitude" and "Phase" outputs are connected to a "Spectral Measurements" block (containing FFT, RMS, and Phase sub-blocks). The "error in (no error)" output is connected to a "Numeric" display. The "error out" output is connected to a "Display Message to User" block. A "Boolean" control is connected to a "STOP" button. The "Spectral Measurements" block outputs "Spectrum" and "FFT - (RMS)" to a "Waveform Graph 2" display. A "1000" constant is connected to the "Time" input of the "Waveform Graph 2".

GUI (Top): The GUI is titled "Read and Display" and features a menu bar (File, Edit, View, Project, Operate, Tools, Window, Help) and a toolbar. It contains two waveform graphs: "Waveform Graph" showing a sine wave over time (0 to 0.01) and "Waveform Graph 2" showing the FFT magnitude spectrum (0 to 2500). The interface includes a "Numeric" display set to 200.00, a "Boolean" control, a "Slider" for "amplitude of signal" (0 to 10), a "Knob" for "frequency of sine signal" (0 to 1000), and a "STOP" button. An "OK Button" is also present.

Project Explorer (Top Right): The Project Explorer shows the project structure for "Project: windowing.lvproj", including folders for "My Computer", "windowing.vi", "Dependencies", "Build Specifications", and "windowing".

Message Dialog (Middle Right): A message dialog box is displayed with the text: "Hi, this is Forrest. I am enjoying designing graphics interface in LabVIEW. You will soon see how I made an explicit diagnostic report generator using LabVIEW. And, I am working on Linux now. I like LabVIEW that the code can run seamlessly from Linux, Mac, to Solaris and Windows." Below the text is a "Click me to continue" button.

„LabVIEW Horror“



Wann ist Python eine mögliche Alternative?

- \nexists (fast) fertige LabVIEW-Lösung
- Keine speziellen LabVIEW-Features benötigt (FPGA, harte Echtzeit)
- Aufwendige Algorithmen zu implementieren (nicht nur Daten verschieben)
- Lizenzkosten spielen eine Rolle

- \exists „harte“ und „weiche“ Echtzeitbedingungen
- Hart: Verletzung nicht hinnehmbar (Steuerung der Flugzeughydraulik)
- Weich: Verletzung unschön aber nicht tragisch (DVD-Player)
- Ca. 95% der Anwendungen: weich
- Hauptkriterium: Determiniertheit
- Python nicht-deterministisch (autom. Garbage-Collector)
→ nur weiche Echtzeit
- Ausreichend schnell? → abhängig von Aufgabe

- 1 Vorbemerkungen
- 2 Einleitung
- 3 Schnittstellen**
- 4 Fertige Treiber Nutzen (DLLs)
- 5 Anwendungsbeispiele
- 6 Umsetzungshinweise

Serielle Schnittstelle

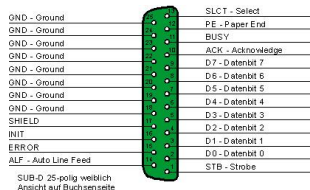
- „Seriell“: Daten (Bitfolge) werden hintereinander übertragen
- Meist RS-232 gemeint
- Heute oft über **USB** emuliert
- **Windows**: COM1, ... **Unix**: /dev/ttyS1...
- Viele (Mess-)Geräte damit ausgestattet
- Paket `serial` (Projekt „pySerial“)



```
import serial
ser = serial.Serial(0, 19200, timeout=2.5) # open first serial port
print ser.portstr                        # check which port was really used
ser.write("hello")                       # write a string
response = ser.readline()                # wait for response
ser.close()
```

Parallele Schnittstelle

- „Parallel“: Mehrere Bitfolgen gleichzeitig über parallele Leitungen übertragen
- Auch als „Druckerausgang“ bekannt
- Einfachste Möglichkeit für digitale E/A
- Heute kaum noch vorhanden
- Empfindlich gegen Überspannung und Kurzschluss
- Paket `parallel` (Projekt „pySerial“)



```
import parallel
p = parallel.Parallel() # open LPT1
p.setData(0x57) # write 0101 0111
responseBit = p.getInPaperOut()
```

- General Purpose Interface Bus (auch:HP-IB, IEC-625-Bus,...)
- Bis zu 15 Geräte parallel

- Paket `visa` (Projekt „pyvisa“)



```
import visa
keithley = visa.instrument("GPIB::12")
ident = keithley.ask("*IDN?")
assert ident.startswith('KEITHLEY INSTRUMENTS INC.') # Konsistenztest
keithley.write(':CONF:VOLT:DC')
v = keithley.ask(':READ?')
print float(v)
```

- Modernere Geräte mit Netzwerkanschluss
- Interner Webserver für die Kommunikation
- *Socket* (engl. für „Sockel“):
A socket object represents one endpoint of a network connection.
- → Client-Server-Architektur
- Paket `socket` (python standard library)



```
# Echo client program
import socket

HOST = '141.30.61.152'      # The remote host
PORT = 50007                # The same port as used by the server
s = socket.socket()
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

- 1 Vorbemerkungen
- 2 Einleitung
- 3 Schnittstellen
- 4 Fertige Treiber Nutzen (DLLs)**
- 5 Anwendungsbeispiele
- 6 Umsetzungshinweise

- Gerätetreiber vom Hersteller oft als kompilierter Code mitgeliefert (`.dll` („Dynamic Link Library“) oder `.so`; „Shared Object“)
- Paket `ctypes` (python standard library)
- Benötigt: Kenntnis über Namen u. Signatur der Funktionen
- Beachtung der Datentypen erforderlich

```
# Optischen Zeilen-Sensor (Linien-Kamera) auslesen
# (Kommunikation ber USB mit unbekanntem Protokoll)

import ctypes

camdll = ctypes.windll.LoadLibrary('D:/devices/cam123.dll')
camdll.setIntegrationTime(5) # 5ms

valuearray_type = ctypes.c_ushort * 512
valuearray = valuearray_type()
pixels = camdll.ReadData(ctypes.byref(valuearray))
assert pixels == 512 # Konsistenztest
values = list(valuearray) # ctypes array -> python list
```

- 1 Vorbemerkungen
- 2 Einleitung
- 3 Schnittstellen
- 4 Fertige Treiber Nutzen (DLLs)
- 5 Anwendungsbeispiele**
- 6 Umsetzungshinweise

USB Missile Launcher als optischer Strahl-Shutter

- Aufgabe: Lichtstrahl abschatten (ohne zeitliche Anforderungen)
- Professionelles Equipment: >500€
- Idee: Zweckentfremden von ... Spielzeug
- → Zweiachsig einstellbarer Raketenwerfer
- Steuerung vom PC aus mittels Windows-GUI-Programm
 - egal
- \exists DLL \Rightarrow \exists python-wrapper
- Aufgabe gelöst

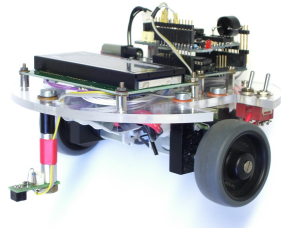


Mobiler Roboter mit Arduino μ C

- Projekt für Schülerpraktikum (7./8. Klasse)
- Kann: Fahren, Blinken, Hupen, „Sehen“ (optische Entfernungssensoren), kommunizieren (Display, RS-232)

Arduino:

- Open-Source μ C-Entwicklungsplattform einfach \cup leistungsfähig
- Atmel AVR-Mikrocontroller + Platine
- USB, Spannungsversorg., LEDs, Reset-Knopf
- Plug'nPlay
- Digital + Analog I/O (AO = PWM),
- Software: IDE + Bibliotheken + Beispiele
- \rightarrow Stark vereinfachte C++-Programmierung



Ziel:

- Vom PC aus den Roboter steuern

Umsetzung:

- C++-Programm auf μ C wartet auf Befehle und führt dann bestimmte Aktionen aus
- Python Programm um die Befehle zu senden
- Jeder Befehl besteht aus zwei Byte:
 - Kommando ('F'=Forward, 'B'=Backward, ...)
 - Argument (numerischer Wert des Bytes, [0, 255])
- Einbindung von `ipython` \Rightarrow interaktives Ausprobieren

- 1 Vorbemerkungen
- 2 Einleitung
- 3 Schnittstellen
- 4 Fertige Treiber Nutzen (DLLs)
- 5 Anwendungsbeispiele
- 6 Umsetzungshinweise**

- Automatisierungsaufgabe schriftlich durchdenken
- Möglichst viel bestehendes nutzen

- Objektorientierung: ein Gerät \leftrightarrow eine Klasse (\leftrightarrow eine Datei)
- Modularisierung: Eigenes Paket („`devices`“)
- Initialisierungsphase und ggf. ordnungsgemäßes Abschalten
- Simulationsmodus vorsehen (Programmtest ohne angeschlossene Geräte)
- Konsistenztest (Überprüfen eigentlich bekannter Informationen)
- Überprüfung zulässiger Wertebereiche
- Logging-Funktionalität

Vorschläge für Übungsaufgaben

- 1 Eine Funktion schreiben, die die Docstrings folgender builtin-Funktionen auf dem Bildschirm ausgibt: `bin`, `hex`, `oct`, `ord`, `chr`, `int`
Hinweis: Jeweils das `.__doc__`-Attribut nutzen.
- 2 Eine Funktion schreiben, die 8 Bit in ein entsprechendes Byte umwandelt und auf dem Bildschirm ausgibt (Dezimal, Hex-Code, und als ASCII-Zeichen)
- 3 Ein Programm schreiben welches die Bit-Folge (1,0,1,1,0,0) auf Bit Nr. 3 der parallelen Schnittstelle wiederholt ausgibt. Taktfrequenz: 0,5Hz.
Hinweis: `time`-Modul

- <http://pyserial.sourceforge.net>
- <http://pyserial.sourceforge.net/pyparallel.html>
- <http://pyvisa.sourceforge.net/>
- <http://docs.python.org/library/socket.html>
- <http://docs.python.org/library/ctypes.html>

- <http://www.arduino.cc>
- <http://www.arduino.cc/playground/Interfacing/Python>

- <http://wiki.python.org/moin/BitwiseOperators>