Kommunikation mit externen Geräten

Carsten Knoll

TU Dresden, Institut für Regelungs- und Steuerungstheorie

Interdisziplinärer Python "Sommerkurs"

27. Juni 2011

Vorbemerkungen

- Ziel:
  - Schnittstellen $\leftrightarrow$ Module
  - Empfehlungen
- Aufbau:

  Einleitung
  Schnittstellen
  - Serielle Schnittstelle
  - parallele Schnittstelle
  - GPIB
  - Ethernet

  DLL-Treiber nutzen
  Anwendungsbeispiele
  - USB Missile Launcher
  - Mobiler Roboter (Arduino-Plattform)

  Umsetzungshinweise

Gliederung

Kommunikation womit?

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

Kommunikation womit?

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

- Hintergrund: Eigenen Erfahrungen (Fraunhofer IIS , iapp)

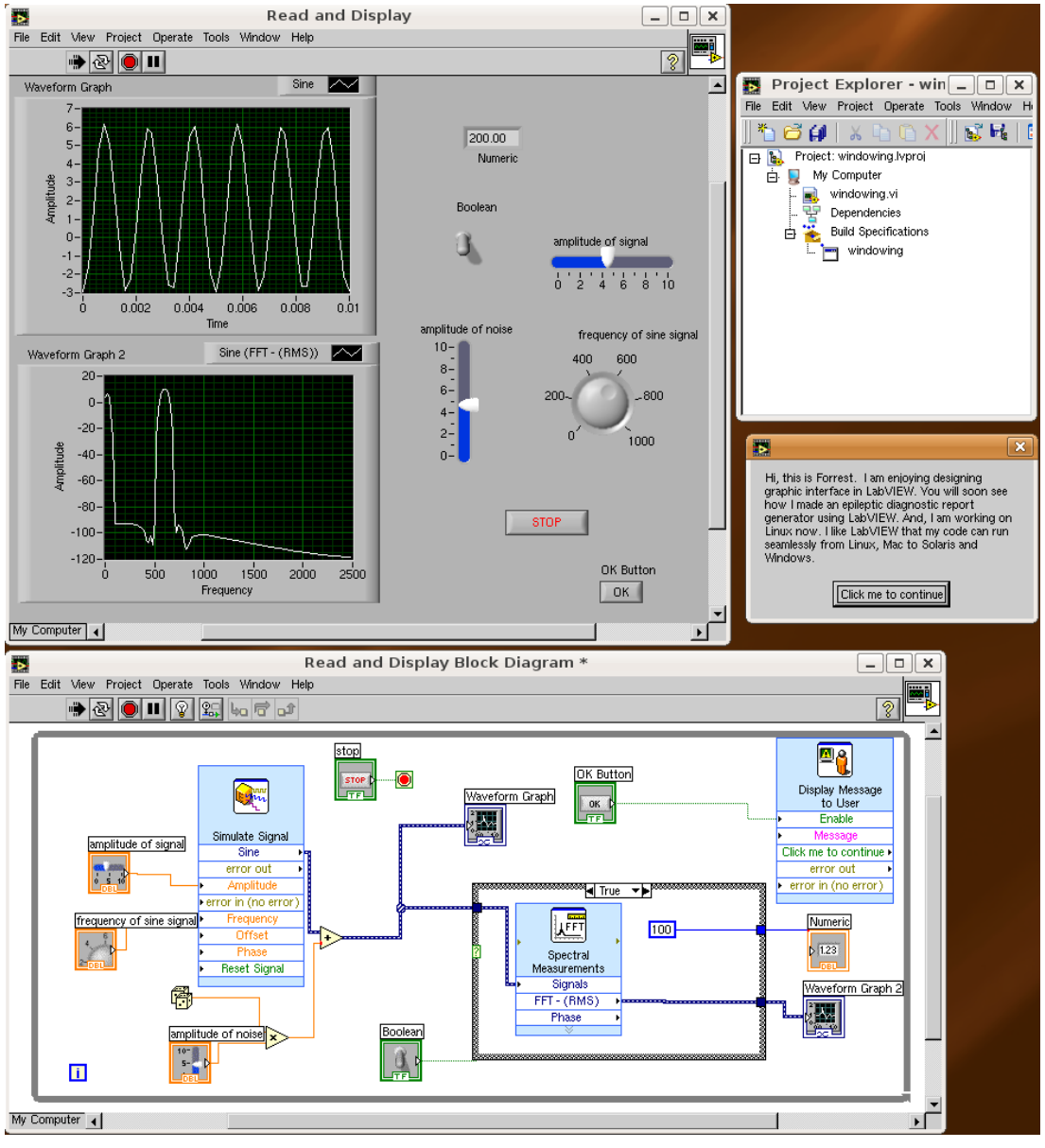
Python im Labor? (I)

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

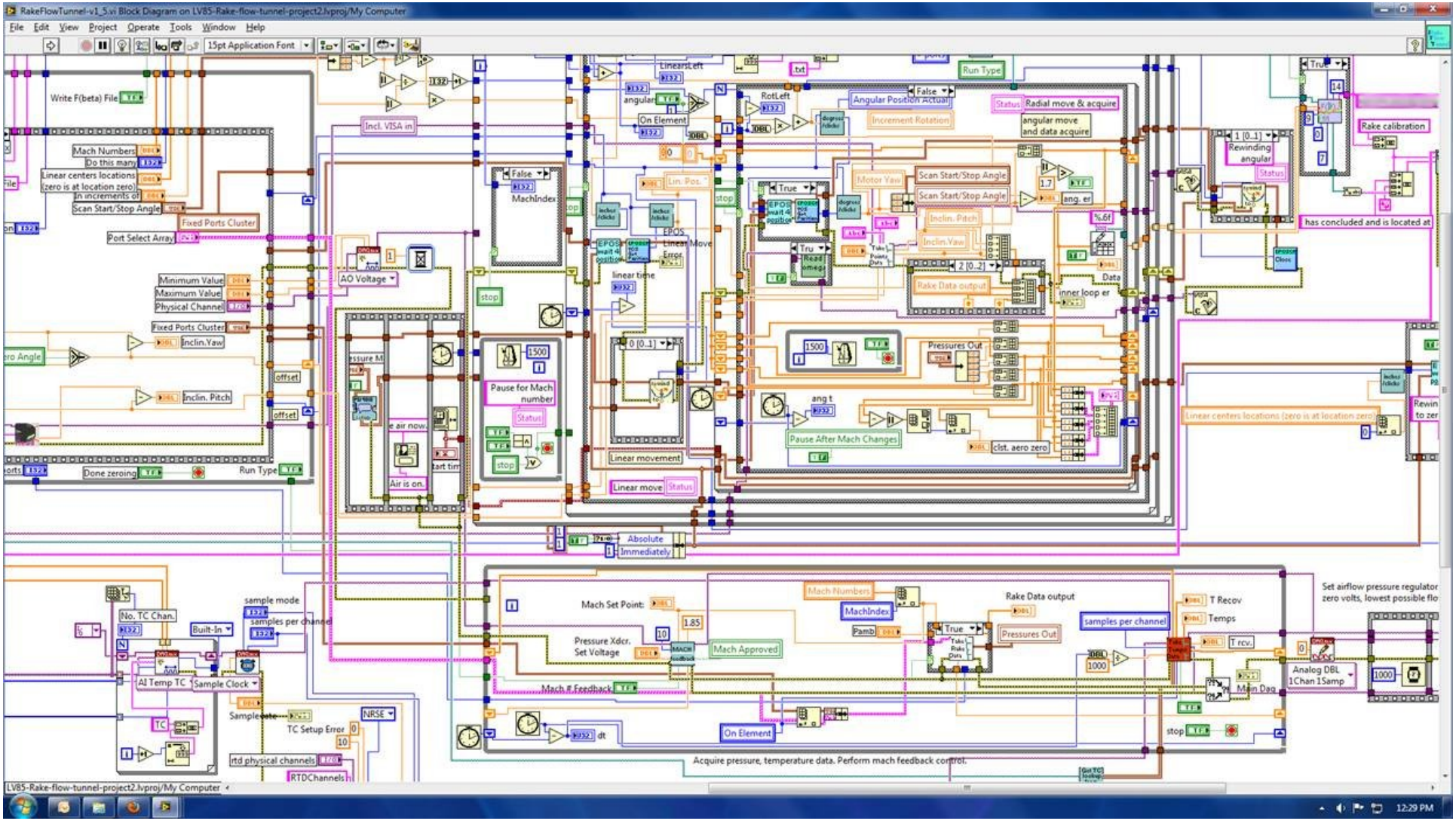
Nachteile (persönliche Meinung!)

- Haupteingabe per Maus (=Informations-Nadelör verglichen mit Tastatur)
- Modularisierung/Kapselung aufwendiger
- Tendiert schnell zu Unübersichtlichkeit
- Permanente Platzprobleme $\rightarrow$ Sparzwang auch bei Kommentaren
- Wartbarkeit $\searrow$, Erweiterbarkeit $\searrow$
- ...

So soll es aussehen



„LabVIEW Horror“



Python im Labor? (II)

Wann ist Python eine mögliche Alternative?

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

Echtzeitfähigkeit

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

Gliederung

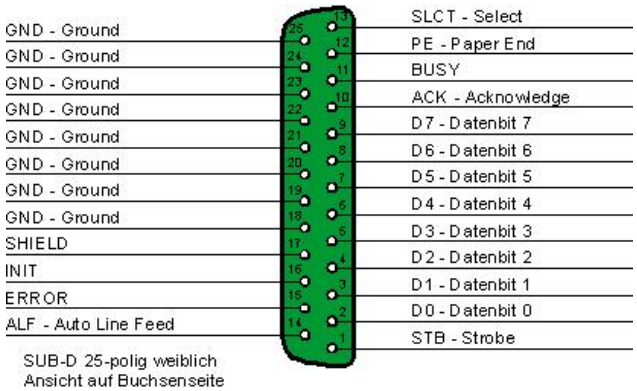
Serielle Schnittstelle

- „Seriell“: Daten (Bitfolge) werden hintereinander übertragen
- Meist RS-232 gemeint
- Heute oft über U**S**B 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()
```

GPIB

- 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)
```

Ethernet (Netzwerk, Sockets) *beispiel01_*.py*

- 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 libray)

```
# 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)
```

Gliederung

DLL Treiber

- Gerätetreiber vom Hersteller oft als kompilierter Code mitgeliefert (`.dll` („Dynamic Link Libray“) oder `.so`; „Shared Object“)
- Paket `ctypes` (python standard libray)
- 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
```

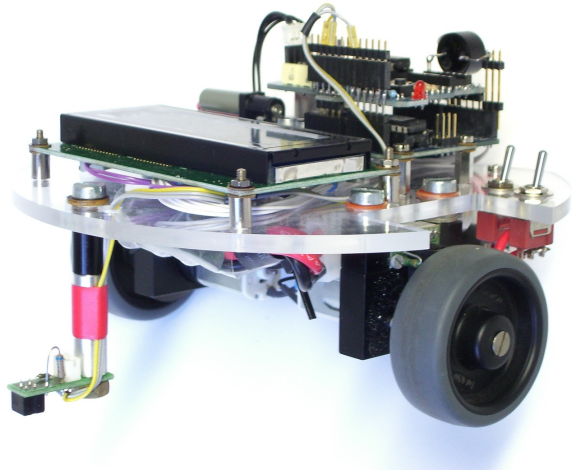
Gliederung

USB Missile Launcher als optischer Strahl-Shutter

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

Mobiler Roboter mit Arduino µC

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



Arduino:

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

Python-Anwendung: Fernsteuerung

beispiel02.py

Ziel:

- Vom PC aus den Roboter steuern

Umsetzung:

- `C++`-Programm auf µC 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

Gliederung

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

Links

- 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