Inverted Pendulum

Leonhard Brunauer Wolfgang Kreil

University of Salzburg

January 25, 2007

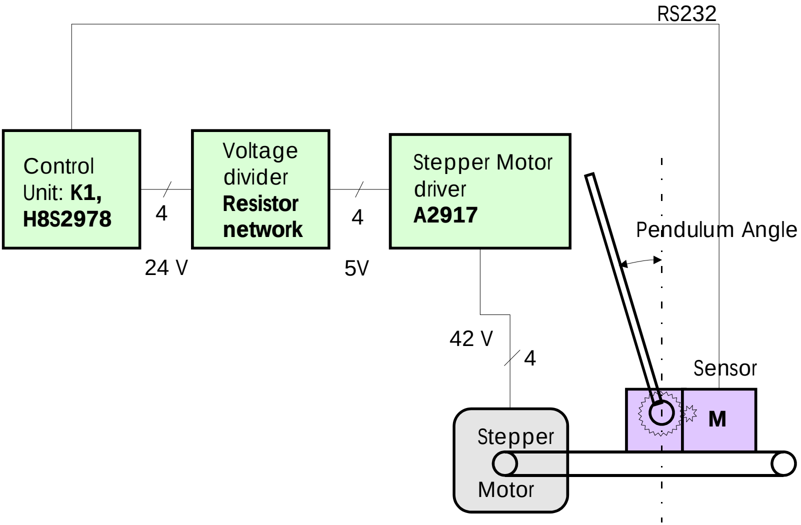
Outline

Introduction

- Control an inverted pendulum built onto a printer.
- Inverted pendulum is the de facto standard example in control engineering.
- Hard real-time constraints.
- Controller is written in C
    - IDE: High-performance Embedded Workshop
    - Flash Programming Toolkit

Hardware Setup

- Cart position is controlled using the printer's built in stepper motor.
- Pendulum angle is measured using a serial mouse.
- Control Unit: K1 (Motion Clinic) **Made in Austria**.


RS232
Control Unit: **K1, H8S2978**
4
24 V
Voltage divider **Resistor network**
4
5V
Stepper Motor driver **A2917**
42 V
4
Stepper Motor
Pendulum Angle
Sensor
M

Control Unit

- Renesas H8S/2378
- 20 MHz
- 32 KB RAM
- 512 KB ROM
- 16 16bit general purpose registers
- Watchdog timer
- A/D converter
- D/A converter
- I^2C bus

Sensor

- Standard serial mouse
- Protocol

	D7	D6	D5	D4	D3	D2	D1	D0
1st byte	1	1	LB	RB	Y7	Y6	**X7**	**X6**
2nd byte	1	0	**X5**	**X4**	**X3**	**X2**	**X1**	**X0**
3rd byte	1	0	Y5	Y4	Y3	Y2	Y1	Y0

- 1200 baud
- 8 data bits
- 1 stop bit
- no parity

Actuator

- Stepper motor
- Motor driver: Allegro A2917
- Driver is actuated using 4 digital output ports
- Output Voltage must be decreased to 5V

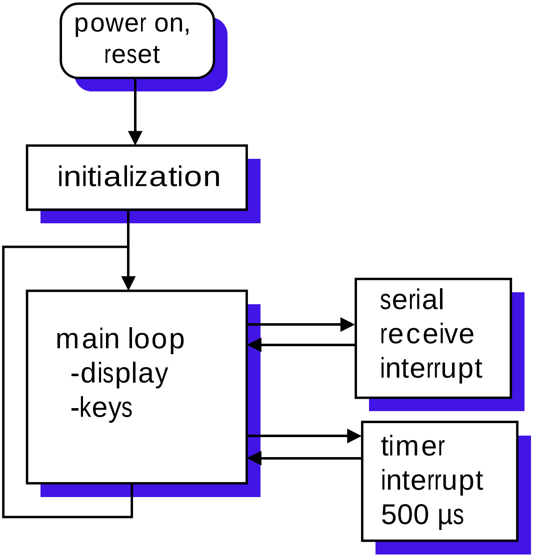
Stepper Motor

- 4 digital outputs to control stepper motor

	coil 1a	coil 1b	coil 2a	coil 2b
step 1	+	-	+	-
step 2	+	-	-	+
step 3	-	+	-	+
step 4	-	+	+	-

Software

- Serial line
    - Event triggered
    - Gives a relative offset
    - No interrupt must be lost
    - Current angle is stored in an integer variable
- Timer interrupt
    - Triggered every 500 μs
    - Read sensor data
    - Run controller to get motor signals
    - Control motor speed
- Watchdog timer
    - Reset on system hang
- Main loop updates display


power on,
reset
initialization
main loop
-display
-keys
serial
receive
interrupt
timer
interrupt
500 µs

Controller Design

- Controller runs in timer interrupt handler (time-triggered)
- System is Single-Input-Single-Output (SISO)
- $\Rightarrow$ use a PID Controller

PID Controller

- Set desired sensor value (setpoint)
- For every time instance t
  - Calculate error $e(t) = setpoint - SensorData(t)$
  - Calculate $P(t) = K_P * e(t)$
  - Calculate $I(t) = K_I * \int_0^t e(\tau)d\tau$
  - Calculate $D(t) = K_D * \frac{de}{dt}$
  - $Output(t) = P(t) + I(t) + D(t)$

Discrete Time PID Controller

- Set desired sensor value (setpoint)
- Set $errorSum = 0$
- For every discrete time instance t
    - Calculate error $e(t) = setpoint - SensorData(t)$
    - Set $errorSum = errorSum + e(t)$
    - Calculate $P(t) = K_P * e(t)$
    - Calculate $I(t) = K_I * errorSum$
    - Calculate $D(t) = K_D * (e(t) - e(t-1))$
    - $Output(t) = P(t) + I(t) + D(t)$

Controller Tuning

- Proportional Gain K_P
  - Set faster or slower response to error.
- Integral Gain K_I
  - Converge to setpoint accurately. Large values may cause overshooting.
- Derivative Gain K_D
  - Smooth overshoot caused by Integral Gain.

Controller Tuning (cont.)

- Set $K_I = K_D = 0$
- Increment K_P until cart is oscillating.
- Set $K_P \approx \frac{K_P}{2}$
- Increment K_I until balancing is stable.
- Increment K_P until overshoots disappear.

Outlook

- Pendulum controller should be easily portable to other micro controller architectures (AVR).
- Problems:
  - Abrupt change of cart direction causes stepper motor drift.
  - Stepper motor is too slow to handle large pendulum deviation.

Demo

Thanks for your attention!