Inverted Pendulum

Leonhard Brunauer
lbrunau@cosy.sbg.ac.at

Wolfgang Kreil
wkreil@cosy.sbg.ac.at

February 9, 2007

Abstract

In this work we have tried to create an inverted pendulum using standard PC hardware. The pendulum in this project has been mounted onto a printer. A serial mouse serves as an angle sensor. The software controller has been implemented bare metal using a common micro controller architecture.

1 Introduction

The inverted pendulum is the de-facto standard example in control engineering. Its aim is to balance a pendulum, which is mounted onto a moving cart. By adjusting the cart's velocity and direction, a controller manages to keep the pendulum in an upright position. In addition to this actuating, the controller needs a sensor to get an idea of the pendulum's current position relative to the cart's normal vector. Similar to other control systems, this is an iterative procedure, since moving the cart in turn changes the pendulum's angle.

One obvious property of this example is that it requires the software controller to run in real time. To make the setup even more challenging, the hardware used in this project has various shortcomings as reported in Section 2. These hardware constraints cause additional requirements for the control software. In particular, the tolerable maximum length of a control period is rather short.

For the sake of simplicity and performance, no operating system is used to run the controller software. Instead, the controller runs on bare metal. This prevents from nondeterminism and bugs introduced by the OS. In particular, running the most important part - the control loop - does not depend on some scheduling policy. Even better, it cannot be preempted. The software architecture is described in Section 3.

To perform the actual control, some basic understanding of control theory is necessary. An inverted pendulum can be controlled by a single-input-single-output loop. That is, a controller that reads a single sensor value and computes a single actuator value. This property has some convenient consequences for the controller's design as described in Section 4.

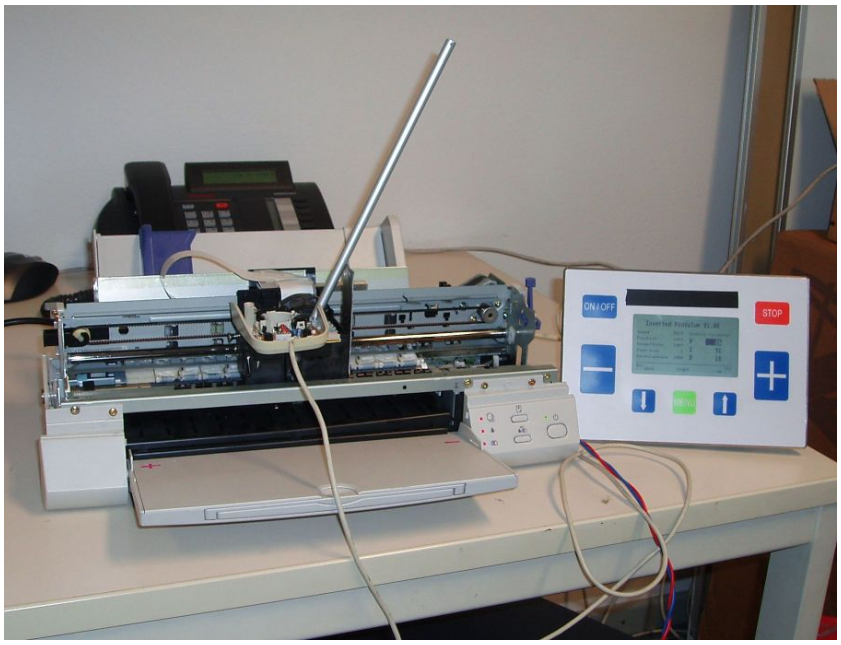


Figure 1: Hardware

2 Hardware

2.1 Hardware Overview

The hardware that form the basis of our Project consists of a ink jet printer Epson Stylus Color 640 as actuator and a Microsoft serial mouse as sensor. We use a serial mouse, because almost every microcontroller provides a RS232 compatible interface and the PS/2 and the USB interface require a more complex implementation. A better sensor solution would be a precision potentiometer, because our Control Unit provides several AD inputs, but we considered the mouse to be cheapest and quickest alternative.

2.2 Control Unit

To control the application we have used the embedded control board K1 from *Motion Clinic, Inc.* This control unit is applied to a swimming pool control. It consists of a 240×128 black and white display, some buttons, some digital and analog inputs and outputs and a serial interface to download the software. The main chip is the microcontroller *Renesas* (former *Hitachi*) H8S 2378 F.

Control Unit Microcontroller

- Renesas H8S/2378 F
- 20 MHz
- 32 KB RAM

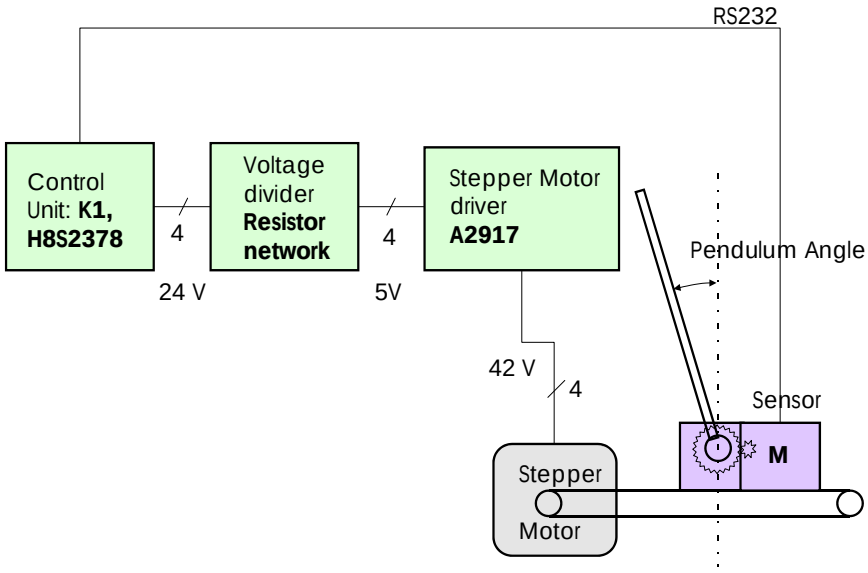


Figure 2: Hardware Overview

- 512 KB ROM
- 16 16bit general purpose registers
- Watchdog timer
- A/D converter
- D/A converter
- I^2C bus

The input/output voltages are 24V DC industrial standard.

2.3 Mechanical Setup

The mechanical reconstruction on the printer are the following:
A **aluminium rod** with 8 mm diameter and 25 cm length is the inverted pendulum. It is mounted on the cart where usually the ink tanks are mounted. It is hold by 2 precision bearings. To obtain enough preciseness of the angle, we have fixed a cog wheel to the sensor.

A **2 button serial mouse** sends the x position, a y position and the button state every time a movement occurs. Only the transmitted x position is relevant for us.

The ***Epson* Stylus Color 640 printer** comes along with 2 built-in stepper motors, one for the paper feed and ink nozzle cleaner and the other one for

moving the cart in the left and right direction. A printer does not allow direct control of the motor and ink nozzles, at the (parallel) interface are only transmitted ASCII codes, control commands like page feed and bitmap patterns. Thus we have made some changes on the printer's main board and mount our own (4 bit parallel) interface at it. A stepper motor requires a special manner to control: one has to set the polarity +/- to force the magnetic field in rotation in several steps.

2.4 Electronic Setup

We have done the following changes at the consisting hardware:

- Stepper Motor
  - Type bipolar, 4 connectors
  - supply voltage 42V DC
- Stepper Motor Driver
  - Allegro A2917 [1]
  - INTEGRATED BRIDGE DRIVER FOR DC AND BIPOLAR STEPPER MOTORS
  - PWM Current-Controlled Dual Full Bridge B1 1.5 A 45 V 2917
  - 44 PIN PLCC case
- Power Supply
  - The C640 contains a switching power supply which provides 5V DC for the logic part and 42 V DC for the power segment and at least 1.5 Ampere.

1. The following input PINs of the driver IC A2917 have been opened:

Pin Number	Name
PIN2	I_{10}
PIN1	I_{11}
PIN44	V_{REF1}
PIN43	$PHASE_1$
PIN42	$ENABLE_1$
PIN23	I_{20}
PIN24	I_{21}
PIN25	$V_{REF}2$
PIN26	$PHASE_2$
PIN27	$ENABLE_2$

2. I_{10}, I_{11}, I_{20}, I_{21} have been connected with GND. With these PINs the Stepper Current is divided as following (see datasheet):

I_0	I_1	Output Current
L	L	$V_{REF}/10 * R_S = I_{TRIP}$
H	L	$V_{REF}/15 * R_S = 2/3 I_{TRIP}$
L	H	$V_{REF}/30 * R_S = 1/3 I_{TRIP}$
H	H	0

It should be obvious that the stepper current I_{TRIP} is adjusted maximal in our case.

3. We have adjusted the current for the stepper motor with V_{REF1} and V_{REF2} to 3 V, this control voltage may vary from 0V to 7 V. If the current is too high, the thermal load of the IC and also the motor would exceed the limits.

4. With $PHASE_1$, $ENABLE_1$ and $PHASE_2$, $ENABLE_2$ it is now very easy to control the stepper motor. A table of the necessary steps to control a stepper in half and full step mode can be found at [2]
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	-	+	+	-

   - Truth Table

Enable	Phase	Out_A	Out_B
L	H	H	L
L	L	L	H
H	X	Z	Z

   X = Don't care
   Z = High impedance

5. The next problem we had to manage was to switch down the 24V DC output of our K1 Control to 5V DC input voltage of the driver IC. We chose the simple way and used a 2 resistor voltage divider with 1 kOhm and 220 Ohm.

6. 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

2.5 Speed Limitations

The speed limits in controlling the pendulum consist of the following constraints:

- The mouse sends at 1200 baud three bytes, so at minimum every 30 msec a new position is received
- The 24V Driver IC at the control unit have a Turn-on and Turn-off delay of minimum 80 μsec and maximum 400 μsec, specified at 12 Ohm R_L, (we use 1 kOhm for thermal,price and size reasons)
- the weight of the pendulum cart has a mass inertia

We currently have a step size of 500 μsec (2 kHz). You could accelerate the stepper motor to higher speed, but this would imply a hardware redesign and a more sophisticated control implementation.

3 Software

The software of this project is split into several components. Maybe the most important one is the controller code. This piece of software is responsible for adjusting the cart's movements, thus put the pendulum into an upright position. For reasons mentioned in Section 4, execution of the controller code must be time-triggered. Therefore, it is executed by the timer interrupt handler, which is triggered every 500μs in our setup.

Another piece of software is responsible for keeping track of the sensor state. This is due to the architecture of a serial mouse. A mouse does not send absolute values, but relative changes to its last position. Moreover, a mouse is event triggered, i.e. it asynchronously sends data as soon as the mouse moves. The pendulum's state, i.e. its current angle, is therefore stored in an integer variable, which is readjusted by the serial line's interrupt handler. It is necessary to calibrate the mouse when the system is initialized to start with a correct state. Obviously, the system is vulnerable in that loosing an interrupt on the serial line means loosing the global state.

The third piece of software is not important for control. It performs user interaction such as updating the display and reading key-press events. Since this is the least important piece of code, it runs only when the system is idle, i.e. when no interrupt handler is currently executing.

Since the controller runs on bare metal, initialization of hardware must be done at system startup. This includes initialization of the timer interrupt and the serial line. In addition to that, a watchdog timer is initialized to recover from system hangs. However, recovering is not possible in the current setup, since the cart is not able to get the pendulum back to an upright position if a certain angle has been exceeded. Moreover, the global state is lost and recalibration would be required.

4 Controller

Balancing a pendulum in a setup such as the one described above requires an angle sensor - a serial mouse, in our case - and a motor to control the cart's position. Hence the system has a single sensor and a single actuator. Fortunately, a setup like this can be controlled by one of the simplest controller architectures: a PID controller.

Basically, the controller adjusts the actuator until the sensor reaches some *setpoint*. In this example the setpoint is a pendulum angle of zero. If the setpoint is reached at some time instant t, the controller is idle. Otherwise it will try to actuate the motor $u(t)$ until the angle $\varphi(t)$ settles at the desired value.

Three parameters are used to tune the controller. The proportional gain K_P determines the magnitude of actuator changes with respect to setpoint errors. That is, small errors will cause the cart to move slow, while larger ones will cause it to move faster. To reach the setpoint more accurately, the second parameter K_I must be tuned. The last parameter K_D causes smoothing of overshoots. The motor signals are generated by

$$\begin{aligned} e(t) &= \varphi(t) - setpoint \\ u(t) &= K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de}{dt} \end{aligned} \tag{1}$$

Putting the above equations into a discrete time setup requires us to perform some numerical simplifications. A simple sum is used as an approximation for the error integral. Usually, the sum is stored in a fixed size integer variable. Hence, care must be taken to prevent from over- and underflows. The differential, on the other hand, is approximated by the gradient of the last two error estimates. Discretization of Equation 1 yields

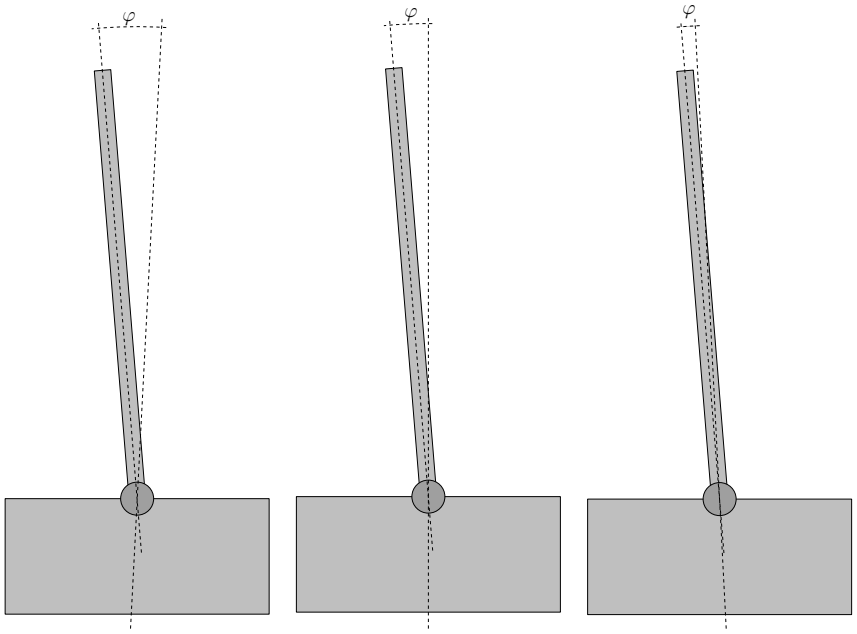
$$u(t) = K_P e(t) + K_I \sum_{i=0}^{t} e(i) + K_D(e(t) - e(t-1)) \tag{2}$$

One outstanding property of PID controllers is that its three parameters K_P, K_I, and K_D can be adjusted empirically, i.e. online. This procedure has been used in this project.

4.1 Design Considerations

Motor signals generated by the controller must be translated to actual triggering of the stepper motor. The direction of movement is simply determined by a signal's sign. The cart's velocity, on the other hand, is determined by a signal's absolute value. A motor signal must be translated to a reasonable value such as steps per second. In our implementation, a simple software timer is used. If the timer reaches a certain limit, it emits a command to perform one step. The counter is reset afterwards. Adjusting the timer's limit corresponds to adjusting the cart's velocity. Unfortunately, this procedure can only be performed up to a certain extend, due to hardware constraints. It is impossible to perform more than one step per millisecond.

Another problem is introduced by the limited distance the cart may move. Its important to keep the cart near the middle to prevent it from reaching one of



(a) Left end (b) Middle (c) Right end

Figure 3: Calculating the setpoint error φ at different cart positions

the ends. This can be achieved by adding complexity to the controller. It would require two sensor values - angle and cart position - to be read. The resulting controller would be required to reach two setpoints simultaneously. For the sake of simplicity, we have chosen a different approach.

A simple workaround is to adjust the setpoint according to the cart's position. That is, if the cart moves towards either of the two ends, the setpoint is shifted towards the middle (see Figure 3).

5 Conclusion

We managed to build and control an inverted pendulum using standard PC hardware and a simple microcontroller. It turned out that our setup - printer, mouse, and microcontroller - is close to the minimal requirements for balancing the pendulum. We have been able to implement an appropriate controller using the simple PID approach.

References

[1] Unknown. Allegro A2917 datasheet, Availbable at: `http://www.alldatasheet.com/`.

[2] Unknown. Schrittmotoren, Available at: `http://www.roboternetz.de/`.