ELM Home Page

DC Servomotor Controller

July 21, 2001
Updated: November 12, 2005

This is an experiment on the closed loop DC servomotor control system (*SMC*). It will able to be used for practical use with/without some modifications. The closed loop servo mechanism requires real-time servo operations, such as position control, velocity control and torque control. It will be suitable for implementation to any embedded 32 bit RISC processors as a middleware. In this project, these operations are processed with only a cheap 8 bit microcontroller.

Recently, most servo systems are using brushless motors called "AC servo motor" to reduce maintanance cost. The AC servo motor with PM rotor is a kind of the DC motor, difference between AC servo and DC servo is only motor driver. Any other block includes theory on the servo control is same as DC servo system.

Hardware

Figure 1. Block diagram

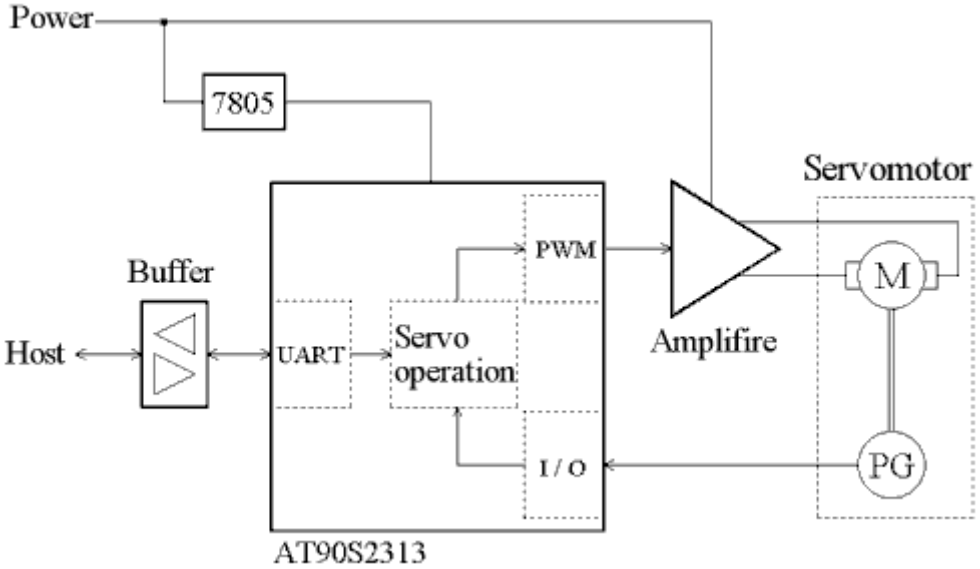


Figure 2. Circuit diagram (Obsolated)
Figure 3. Built servo controller board

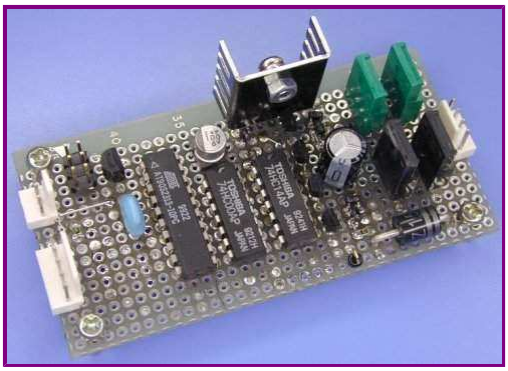


Figure 1 shows the block diagram for SMC. This is built with only an AVR microcontroller and a PWM mode power amplifire. Whole of servo operation is processed by software implemented servo processor. Any analog component for servo operation is not used.

Figure 2 shows the circuit diagram for SMC. The AVR processes most servo function, such as position capture, servo operation, PWM output and motion command from host controller. The power amplifire is high efficiency PWM mode H-blidge motor driver. It can drive a motor rated up to 50W at 12V supply voltage.

JP1 is an ISP connector to program the AVR, and it can also used to attach an LED display board. P2 is a host interface to be controlled by any host controller, it can be connected to PC's serial port directly. P4 is a pulse commanded interface like most servo controllers.

Figure 3 shows the built servo controller board. It is built on a proto board 44mm by 85mm.

Software

Figure 4. Positioning servo operation

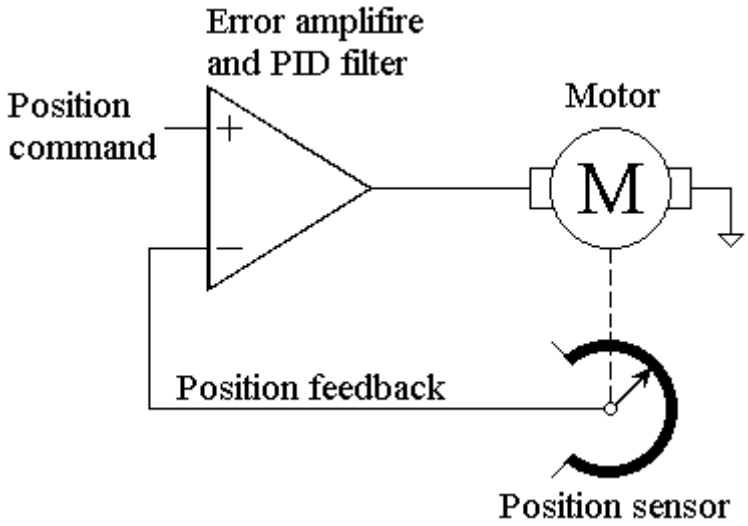


Figure 4 shows the theory on the positioning servo mechanism. Current motor position is fed back to the servo controller with a potentiometer or a rotary encoder. The motor position is compared to commanded position, the motor is driven according to the position error, and the motor is moved and held to the commanded position.

The servo operator consists of error amplifires and PID filters. Until 1980's, the servo operator is realized with op-amps, F-V converter, differential counter, D-A converter and many analog components. After 1990's, digital signal processing is popularized for the servo controller, and new servo algolithms, such as AI control, robust control and fuzzy control, have also been developped and applied.

The single loop servo control shown in figure 4 is called "Conventional servo mechanism". Because its servo performance is worse, it is not used for high performance servo controller.

Figure 5. Operation diagram for the SMC (Cascaded control)

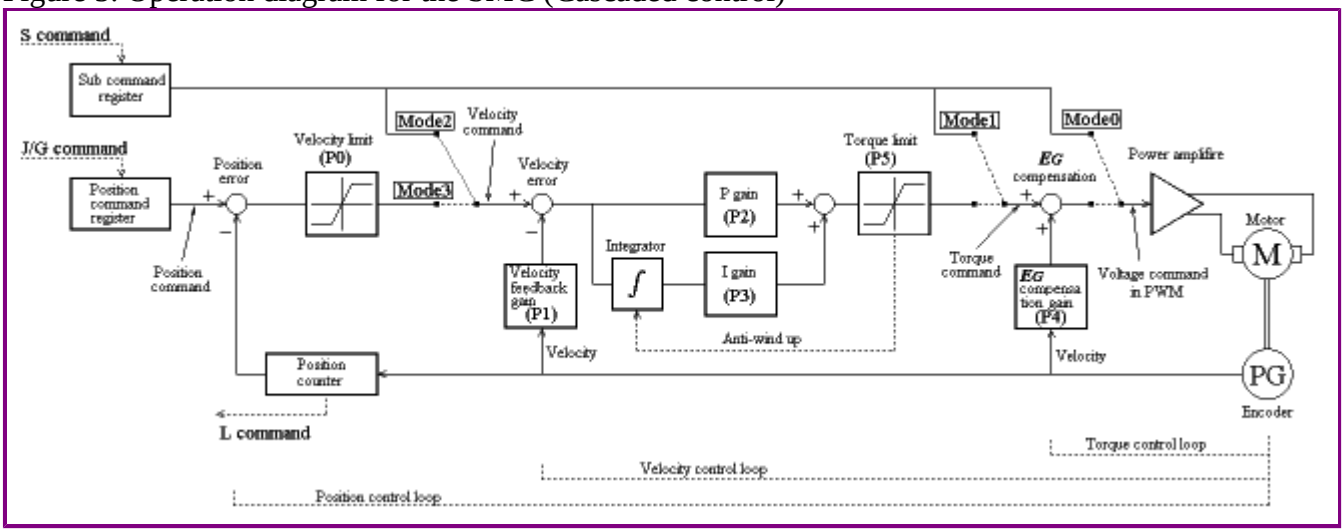


Figure 5 shows the servo operation for SMC. It is multiple feedback configuration which is most popular and fundamental servo mechanism at now. The servo operation which has multiple feedback, is called "Cascaded control". It is a kind of "Advanced servo mechanism". At the cascaded control, one or more control term whoes response is faster than major loop is chosen, and put it into the major control loop as minor control loop, the overall servo performance is increased.

Now, these servo operations are processed by software implemented digital servo operator. To make discrete approximation of the servo operation, the update period which is fully faster than mechanical response is required. Normally, the update rate from 1 kHz to several kHz is selected. It will able be processed with most microcontroller without DSP at now. SMC processes the servo operation at 1 kHz update rate, every operation is processed within 92 µsec, so that AVR can be used for servo processing with ease.

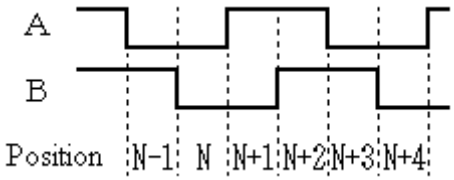
SMC can operate at *four different servo mode*, positioning (tracking) mode, constant velocity mode, constant torque mode and constant voltage mode. The operation modes are determined by data path of servo process. Each operateion can be controlled by a host controller via SMC's serial interface. Additionally, it can also operate as a stand alone motion controller with any update of its firmware.

Mode 0 (voltage controlled mode) drives motor at constant voltage. *Mode 1 (torque controlled mode)* drives motor at constant torque independnt of rotation velocity. *Mode 2 (velocity controlled mode)* drives motor at constant veocity independnt of load torque. *Mode 3 (positioning mode)* drives motor for the commanded position and hold there. This is called "tracking servo" which is typical usage for motion control system.

The servo parameters, such as any gains and limits, can be changed dynamically. However, SMC doesn't have automatic tuning feature, so that you have to tuneup the servo parameters manually for the properties of actual motor and load. AVR has 128 bytes of non-volatile data memory, the servo parameters can be saved or loaded to the memory.

Position capture

Figure 6. Encoder output



To capture the motor position, 52 kHz timer interrupt is used to sample the quadrature output from incremental rotaly encoder and update current position register. Normally, a hardware buffer counter is used for the encoder interface to reduce load of the capturering process. However, SMC samples the input signals directly with only

software process to reduce external components.

The maximum count rate of the software sampling method is less than its sampling rate. If input count rate exceeds the sampling rate, count error will occur and the correspondence between actual motor position and current position register will be lost. At the SMC, to increase over speed tolerance, a special process is applied to the error code. It can accept input count rate up to two times faster than its sampling rate. Therefore, the maximun input count rate is up to 104k cps. When a 400 ppr (1600 cpr) incremental encoder is used, the maximum rotation velocity becomes:

```
2 * 52k[sps] / 1600[cpr] * 60 = 3900[rpm]
```

Actually, the maximum count rate will little decrease from above value due to the jitter of interrupt response time and mechanical accuracy of the encoder.

Internal processes for SMC

Figure 7a. Foreground process

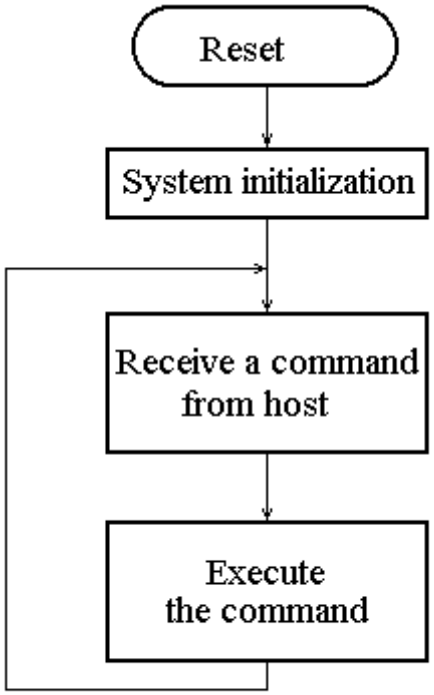
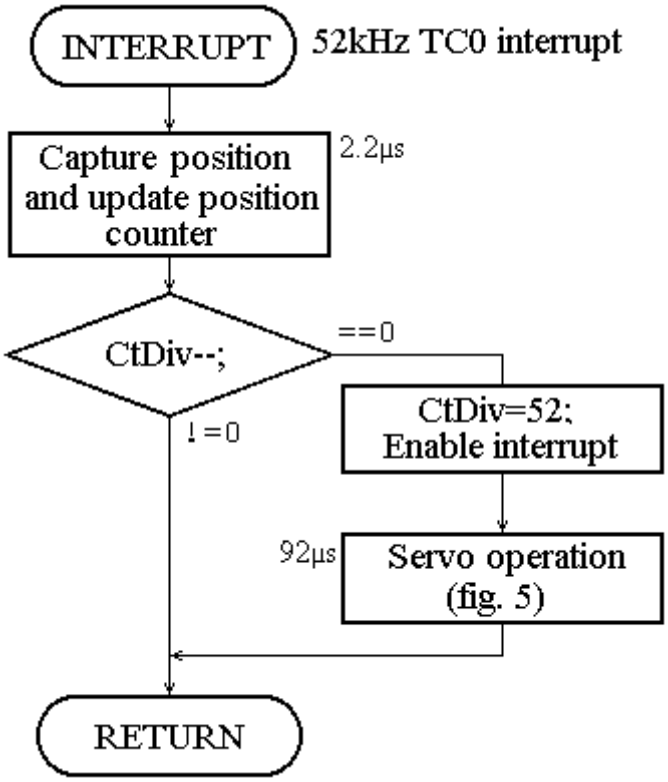


Figure 7b. Background process



SMC has two internal processes, foreground process and background process. The foreground process executes motion commands from host controller. The background process processes position caprureing and real-time servo operation.

Online command

SMC is controlled by online commands form host controller via its serial interface. Ther serial data format is N81 and the data rete is 38.4k bps. It can be connected to a PC's serial port directly. Follows are details for the online commands.

M - Change servo operation mode

Mode#	Servo operation mode
0	Voltage controlled mode (default)
1	Torque controlled mode
2	Velocity controlled mode
3	Position controlled mode

```
M <Mode#>
```

This command changes the servo operation mode. It means to change the data path at servo operation shown in figure 5. Position command register, sub-command register and position counter are also cleard at the same time. The correspondence between mode numbers and operation modes are shown in right table.

L - Show position counter

```
L<CR>
```

The position counter is a 24 bit signed register which retains current motor position. The L command shows the value into console in real-time. Type any key will terminate this command.

E - Change echo back mode

```
E <Switch><CR>
```

This command enables/disables echo back of command characters from the host. Zero disables the echo back, one enables the echo back. Power-on initial value is one.

To prevent receive buffer overflow at tracking operation with continuous J commands, the command echo back should be disabled.

S - Set a value into sub-command register

```
S [<Value>]<CR>
```

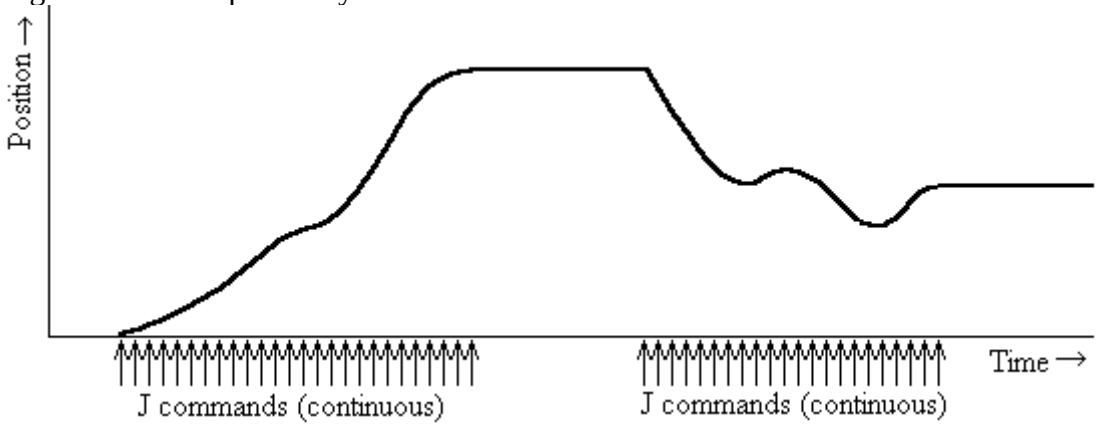
The position counter is a 16 bit signed register which is enabled at except mode 3. When the <Value> is ommited, current value is displayed and prompts to input a new value.

In mode 0, the range from -255 to 255 is effective, and controls output voltage between -Vs to +Vs. In mode 1, the range from -255 to 255 is effective, and controls output voltage between -Vs to +Vs. The output voltage is compensated according to rotation velocity to cancel counter generation. In mode 2, correspondence between the value and controlled velocity is as follows:

```
Velocity [rpm] = Sub-command reg. * 15000 / P1 / Encoder resolution [ppr]
```

J - Set a value into position command register

Figure 8. Motion profile by J command




```
J <Position><CR>
```

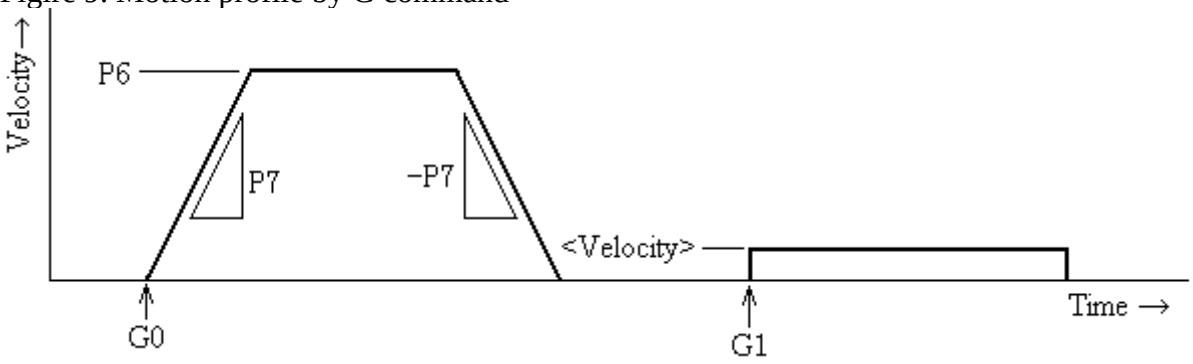
This command sets a value into position command register. The position command register is a signed 24 bit register which is effective in only mode 3. *It can also be changed by step/dir input alternatively or simultaneously.*

When the position command register is changed, motor moves to the commaned position quickly. Long trip by

the J command will cause overshoot, ringing or mechanical noise. This command is suitable for tracking operation shown in figure 8.

G - Generate a motion

Figire 9. Motion profile by G command




```
G0 <Position><CR>
G1 <Position> <Velocity><CR>
```

This command has two different motion profiles shown in Figure 9, each of them are equivalent to G0/G1 commands of NC instruction. The G0 command is used for most quick positioning motion, it goes to the commanded position with trapezoidal motion profile. The accereration and the maxim velocity at the motion is from servo parameter P6 and P7. The G1 command is used for cutting motion to feed a work at constant velocity.

These motions are generated by foreground process, so that next command cannot accepted until the motion is finished and motor stabled. If any character is received during the motion, the motion will be canceled and finished the motion command immediataly.

P - Change servo parameter

```
P <Parameter#> [<Value>]<CR>
```

This command sets the <Value> into the servo parameter register specified by <Parameter#>. The parameters can be saved/loaded to non-volatile data memory with R/W command.

Parameter #0: Velocity limit

This is an unsigned value which regurats moving velocity at positioning mode. For example, a motion at long trip by J command will reach this limitation. As for the SMC3/A, the value should be less than no-load speed, or servo error might occure.

```
P0 = Velocity limit [rpm] * P1 * Encoder resolution [ppr] / 15000
```

Parameter #1: Velocity feedback gain (K_F)

Parameter #2: Velocity error proportional gain (K_P)

Parameter #3: Velocity error integration gain (K_I)

These are unsigned 16 bit fixed point values whoes decimal point is at byte boundary. When you wish to set 1.5, set 1.5*256 = 384.

Parameter #4: Torque limit

This parameter limits output torque (output current). It is a unsigned 16 bit value. As for the SMC3/A, when the torque is limited for a time, a servo error will be occured.

```
P4 = Current limit [A] / Supply voltage [V] * Armature coil resistance [ohm]
```

Parameter #5: Back-EMF compensation gain

Normally, the output torque generated by armature winding current is sensed and fed back to make torque control loop. However, SMC ommits the current sensor and controls the output torque with sensorless current control method. To supply commanded current to the motor, SMC applies compensation to back-EMF (EG). This is inaccuracy compared to conventional current feedbacked control. But I chose it because it is easy to build the circuit board.

This parameter is unsigned 16 bit fixed point values whoes decimal point is at byte boundary. The value can be calcurated as follows:

```
P5 = KG [mV/rpm] / Encoder resolution [ppr] / Supply voltage [V] * 3840
```

Parameter #6: Maximum velocity for G0 command

This parameter specifies maximum velocity at G0 comand. It is a unsigned 16 bit value.

```
P6 = Velocity [rpm] * Encoder resolution [ppr] / 15000
```

Parameter #7: Accereration for G0 command

This parameter specifies acceleration/deceleration at G0 comand. It is a unsigned 16 bit value.

```
P7 = Acceleration [rpm/s] * Encoder resolution [ppr] * 17
```

W - Save servo parameter

```
W<bank#><CR>
```

This command saves current value of the servo parameters into non-volatile data memory. The data memory is divided in eight banks. The <bank#> specifies the bank number to be saved.

R - Load servo parameter

```
R<bank#><CR>
```

This command loads servo parameters from non-volatile data memory. The bank 0 is loaded automatically at power-up initialization.

Result and investigation

Processor usage

Figure 10a. Processor load ratio

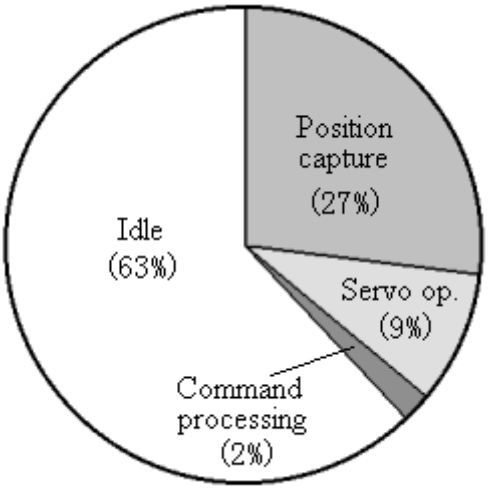


Figure 10b. Program memory usage

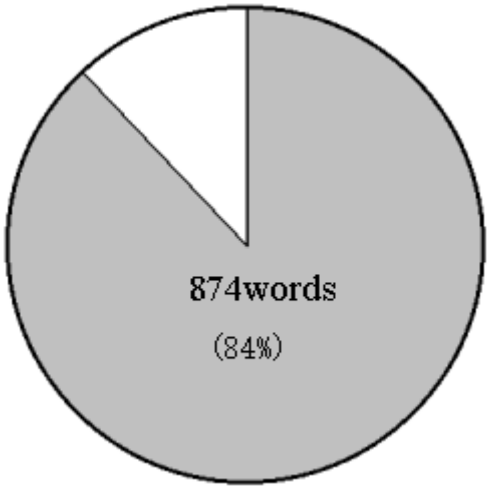


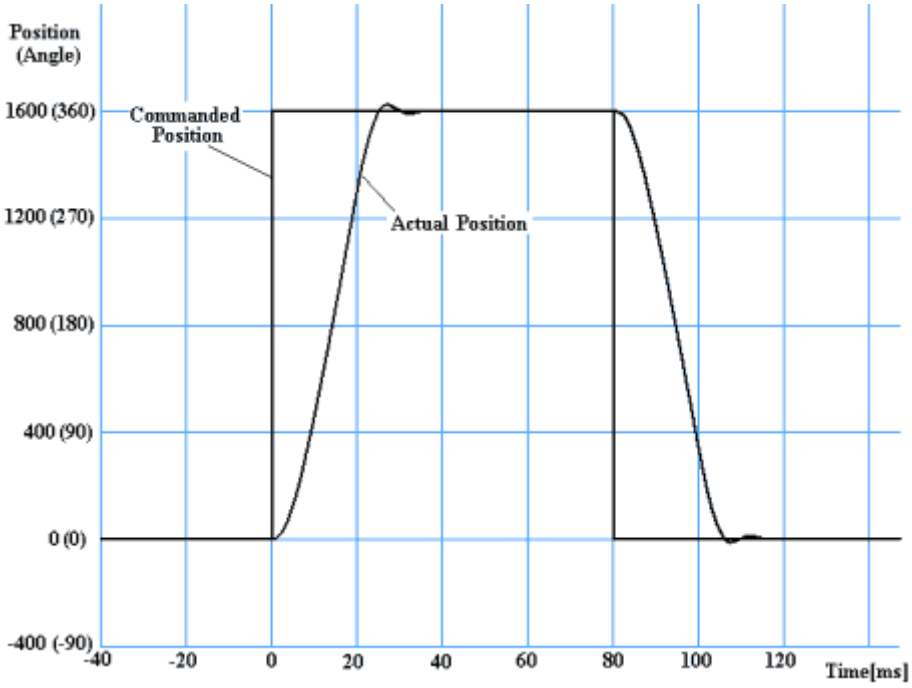
Figure 10a shows the processor load ratio for each process. Because of SMC is capturing encoder input with only software, this process occupies most time among effective processing time. The software encoder captureing can be used at two axis at the best. The servo processing seems to right though it is processing many operation.

Threfore, over 60 percent of the processor power is in idle, the AVR has enough capability for servo control. It will able be used for up to several axis servo control if any external buffer counter for encoder captureing is added. Actually, I have developped a very low cost six axis servo controller for entertainment machine with an AVR (8535) and some CPLDs :-)

The 80 percent of program memory is in use. It might not enough to use as a stand alone motion controller. I reccomend to remove unnessesary functions, such as command processor, to expand application area.

Servo performance

Figure 11. Measured step response



Motor: UGTMEM-A1SA51
(Yasukawa Electric Corp.)
V_S = 14.5V
K_F = 4.6
K_P = 1.5
K_I = 0.13
No load

Figure 11 shows the measured step response at tracking servo operation (mode 3). This is the motion trace of 360 degree hopping motion with J commands. It can be read form the graph, the setting time is less than 40ms, and a little overshoot appears.

To use SMC for any motion control, the servo parameters should be tuned-up for actual load properties. Normally, to analyse servo systems, filters or PLLs mathematically, the transfer function is used. Please refer to any other documents on applying the transfer function to the servo system. You will able to find the documents with the keywords, "transfer function" and "servo".

Technical Data

- Circuit diagram for SMC (Obsolated)
- Firmware for SMC (Obsolated)

- Circuit diagram for SMC3/A | High Power Driver Nov 12, 2005
- Firmware and Changes for SMC3/A Nov 12, 2005