SmartMotor Notes

Disabling limits:

- EIGN(2) 'Disable Left Limit'
- EIGN(3) *'Disable Right Limit'*
- ZS *'Reset errors'*

Initiating Motion:

- ADT=100 *'Set Target Acceleration/Deceleration'*
- VT=1000000 *'Set Target Velocity'*
- PT=300000 *'Set Target Position'*
- BRKRLS
- G *'Go, Starts the move'*
- *S 'stop motion'*
- *BRKENG 'Engage brakes to avoid overheating'*
- RPA '*Current motor position in counts'*

Writing/Running a Program:

- Check test.sms
- compile and download button on toolbar
- Run
- Notes: TWAIT is to wait for move to complete

Tuning PID:

ication.

The main objective in tuning a servo is to get **KP** as high as possible, while maintaining stability. The higher the **KP**, the stiffer the system and the more under control it is. A good start is to simply query what the beginning point is (**RKP**) and then start increasing it 10% to 20% at a time. It is a good idea to start with **KI** equal to zero. Keep in mind that the new settings do not take effect until the **F** command is issued. Each time **KP** is raised, try to physically destabilize the system by bumping or twisting it or have a program loop cycling that invokes abrupt motions. As long as the motor always settles to a quiet rest, keep raising **KP**. Of course if the SMI Tuning Utility is being used, it will employ a step function and show more precisely what the reaction is.

As soon as the SmartMotor starts to find it difficult to maintain stability, find the appropriate derivative compensation. Move **KD** up and down until the value is found that gives the quickest stability. If **KD** is too high, there will be a grinding sound. It is not really grinding, but it is a sign to go the other way. A good tune is not only stable, but reasonably quiet. The level of noise immunity in the **KD** term is controlled by **KS**.

The derivative term **KD** requires estimating the derivative of the position error. While the simplest method is a backward difference, **KS**=0, this is inherently noisy. The choices of **KS**=1, 2 and 3 provide increasing levels of noise immunity at the expense of slightly increasing latency in the estimation. Since higher latency will typically result in lower achievable PID loop gains, choose the best compromise between smoothness and tracking performance. The default set-

74

PID Con

ting is **KS=1**.

After optimizing **KD**, it may be possible to raise **KP** a little more. Keep going back and forth until there's nothing left to improve the stiffness of the system. After that it's time to take a look at **KI**.

KI, in most cases, is used to compensate for friction; without it the SmartMotor will never exactly reach the target. Begin with **KI** equal to zero and **KL** equal to 1000. Move the motor off target and start increasing **KI** and **KL**. Keep **KL** at least ten times **KI** during this phase.

Continue to increase **KI** until the motor always reaches its target, and once that happens add about 30% to **KI** and start bringing down **KL** until it hampers the ability for the **KI** term to close the position precisely to target. Once that point is reached, increase **KL** by about 30% as well. The Integral term needs to be strong enough to overcome friction, but the limit needs to be set so that an unruly amount of power will not be delivered if the mechanism were to jam or simply find itself against one of its ends of travel.

Appendix C page 105 (commands)

What is homing?

Homing is a sequence of predefined motions that are normally required in order to configure the system's absolute position after power up. The homing sequence is carried out by searching for an absolute known sensor along the mechanical travel, and updating the internal position accordingly.

Essentially, when we start the motor, we do not know it's absolute position. Homing means the motor would go to a position known, either update it's position and start from there, or it would record it's encoder counts, update it's absolute position, then add those encoder counts to start from the position we started in.

Things to lookup:

- How to monitor how much current is being pulled by the motor?
  - **RUIA**
  - **Ba** 'get overcurrent status bit'