

MCE/EEC 647/747: Robot Dynamics and Control
Adept Controller Workaround

The PUMA 560 manipulator available in the lab comes with independent joint PI control loops for each axis. These loops are hard-coded and the user is only able to change the control gains. Our objective is to replace the native PI loops with the control algorithms studied and simulated in this course. Ideally, direct access to the servo amplifier analog inputs and joint encoders should be available for maximum flexibility. Modern real-time control interfaces like dSPACE and WinCon are able to receive and send signals to the robot hardware in a Simulink environment. Such access hasn't been prepared yet, so an alternative solution using built-in functions must be found.

One possibility to obtain direct access to the joint amplifiers via software is to “undo” the feedback loop and set the integral gain to zero, as shown in Figure 1. The loop is then closed via the V+ operating system. In theory, any output-based controller can be used (for instance, the observer-based adaptive passivity-based controller). The figure shows a simple P controller with net gain K_1K_p . Incremental variables have been represented in this loop.

Ideally, after a control command du has been computed by the control algorithm, the value should be applied directly to the servo amplifier. Unfortunately, no V+ command is available to perform this operation. The closest command available is `drive`, which is designed to provide a ramp reference input to the fixed-architecture loop shown in the figure. The syntax is

```
DRIVE joint, angle, speed
```

where `joint` is the number of the joint to be moved, `angle` is the requested increment or decrement from the current position and `speed` is a relative percentage from 0 to 100.

The loop cancellation is achieved by adding Δy to the calculated Δu and using this as the reference input to the fixed-architecture loop. Since the `angle` argument is incremental, we should simply use $\Delta u + \Delta y - \Delta y$. Therefore any calculated Δu is used directly in place of `angle`.

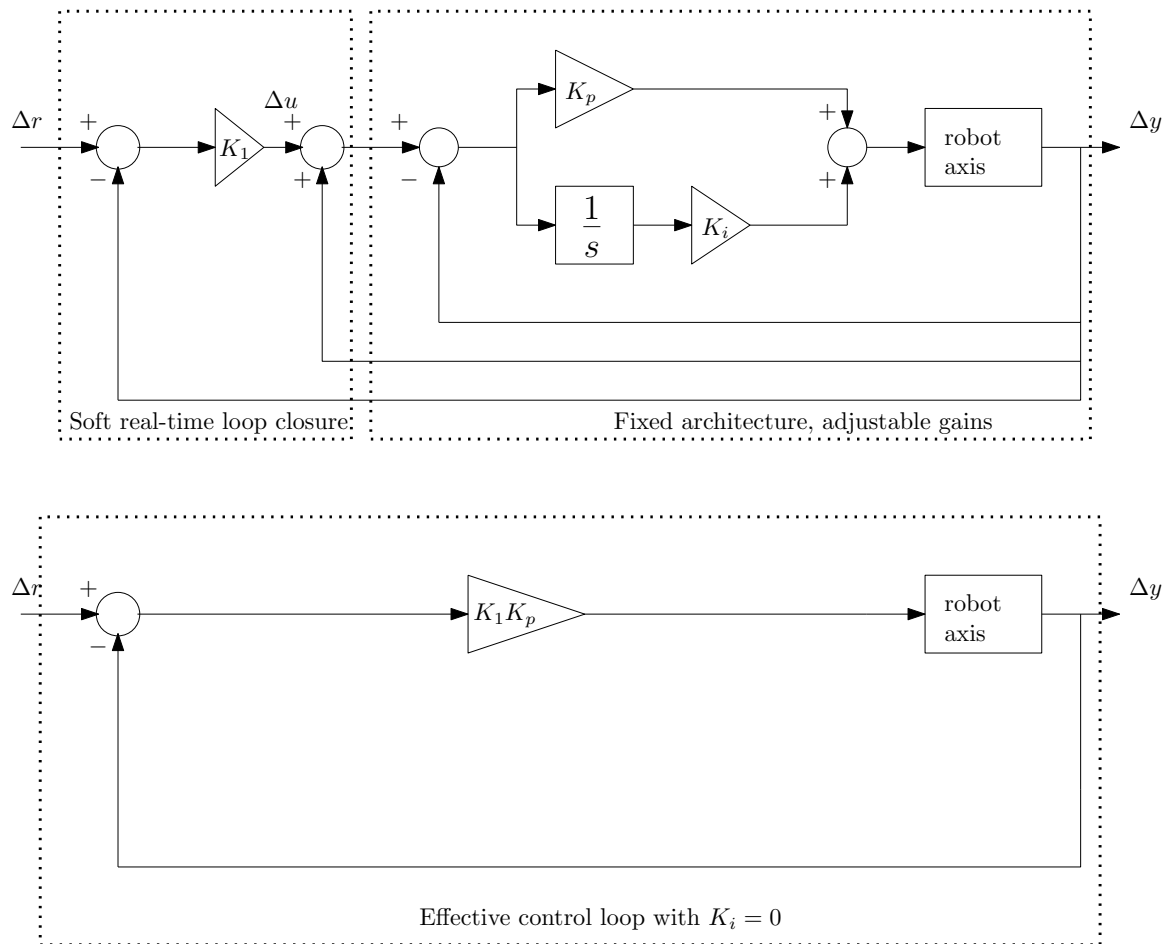


Figure 1: Playing games with the motion control software

Since the implementation is digital, the **drive** commands must be paced at constant intervals separated by the sampling period. In a *soft real-time* implementation, a timer is initialized at the beginning of the cycle. After all sensor readouts, control calculations and drive commands have been carried out, the process must wait until the sampling interval is up (this is called *latency*). Then the cycle is repeated. You can see this in the program listing below. During the initial programming phases, monitoring must be used to ensure that there is positive latency.

Note: Comments in V+ are signaled by a semicolon (;)

```
.PROGRAM pctrl.v2()
sp=50 ;desired angular increment from current location
tol=0.1 ;tolerance for reaching the target and terminating
target=0 ;logical variable, target has not been reached
HERE #a ; assign current joint positions to variable a
DECOMPOSE array[0]=#a ;extracts individual joint positions and fills array
```

```

;starting with index 0
    y0=array[0] ;extract just joint 1 position
    WHILE target <> 1 DO
TIMER(1)=0 ;initialize timer #1 (we need only 1)
HERE #a ;
DECOMPOSE array[0]=#a
dy=array[0]-y0 ; calculate the current increment from the starting position
u=0.5*(sp-dy) ; this is the proportional control law with gain = 0.5
    IF ABS(sp-dy)<=tol THEN
target=1
END
DRIVE 1,u,100
twait=0.05-TIMER(1) ; figure out required latency to meet a sampling interval of 0.05
    ;TYPE twait ;un-comment for monitoring purposes
    DELAY twait ;do nothing until interval is up
    END
TYPE dy ;display ending increment (should be close to sp, within tolerance)

```

There are two limitations with this approach. The more serious one is that the code shown, although pretty simple, consumes about 0.03 seconds per cycle, placing a limit in the sampling interval. A more calculations are added for Δu , available latency decreases, forcing us to use a longer sampling interval. Eventually, the sampling interval is too long for the intended system bandwidth and this approach cannot be used.

The other limitation is that the `drive` command is intended for a complete motion operation ending with zero velocity. When used within the control code, it will produce choppy motion, especially with small gains.

Important Safeguards

1. Use setpoints which will not cause the robot to reach the maximum ranges of motion
2. Set the integral gain in the built-in loop to zero, and set the proportional pole and zero to the same values, so they cancel. You don't need to change the proportional gain in the built-in loop, since the program can override that value.

Activity Run the program with various gains and observe the overshoot resulting from using values of proportional gain significantly higher than nominal.

Using the class discussion on digital PID implementations, modify the above program to include an integral gain. Tune the PI controller for maximum performance.

As part of the final project, you will add a digitized high-gain observer for future use with adaptive control approaches.