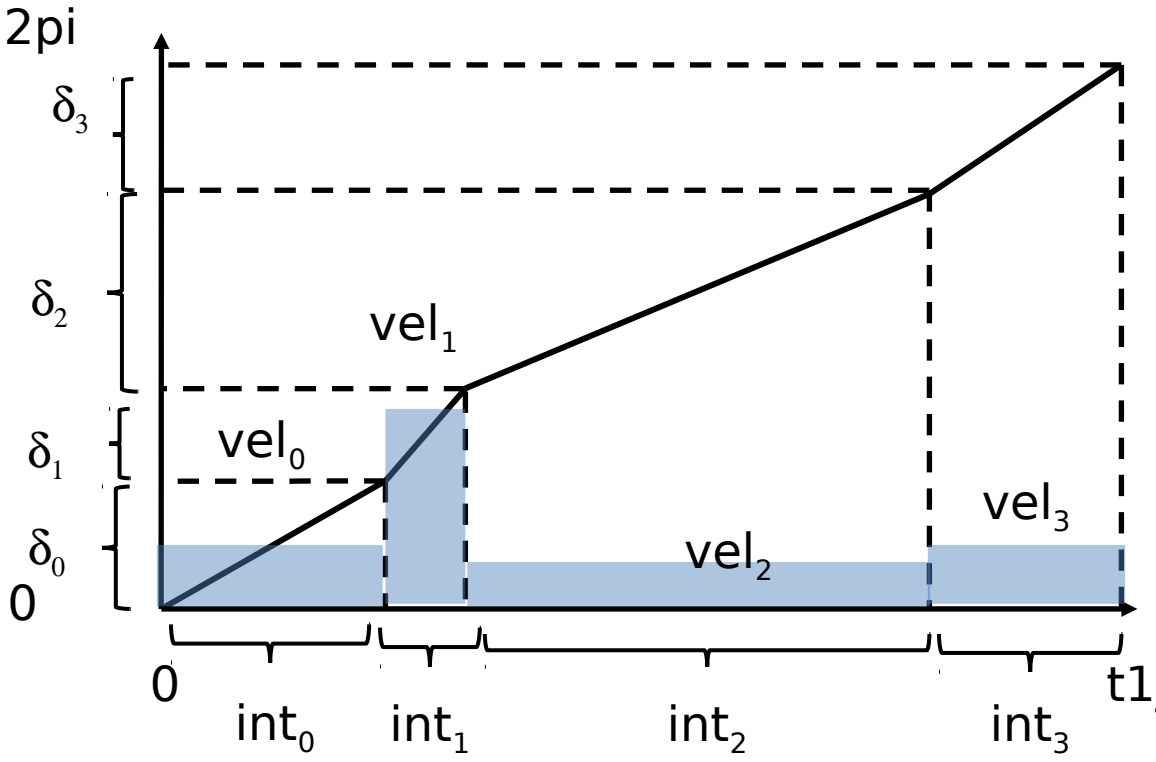



One cycle

at start of interval0:
p_input = p_input + ref position $\delta_0$,
set new $\delta$ after every
interval (long)
interpolate used to make smoother pos ref setpoints
v_input = $vel_0$ (should be calibrated to reach $\delta_0$
in $int_0$ ms)
v_input = ref velocity, set every interval (int)

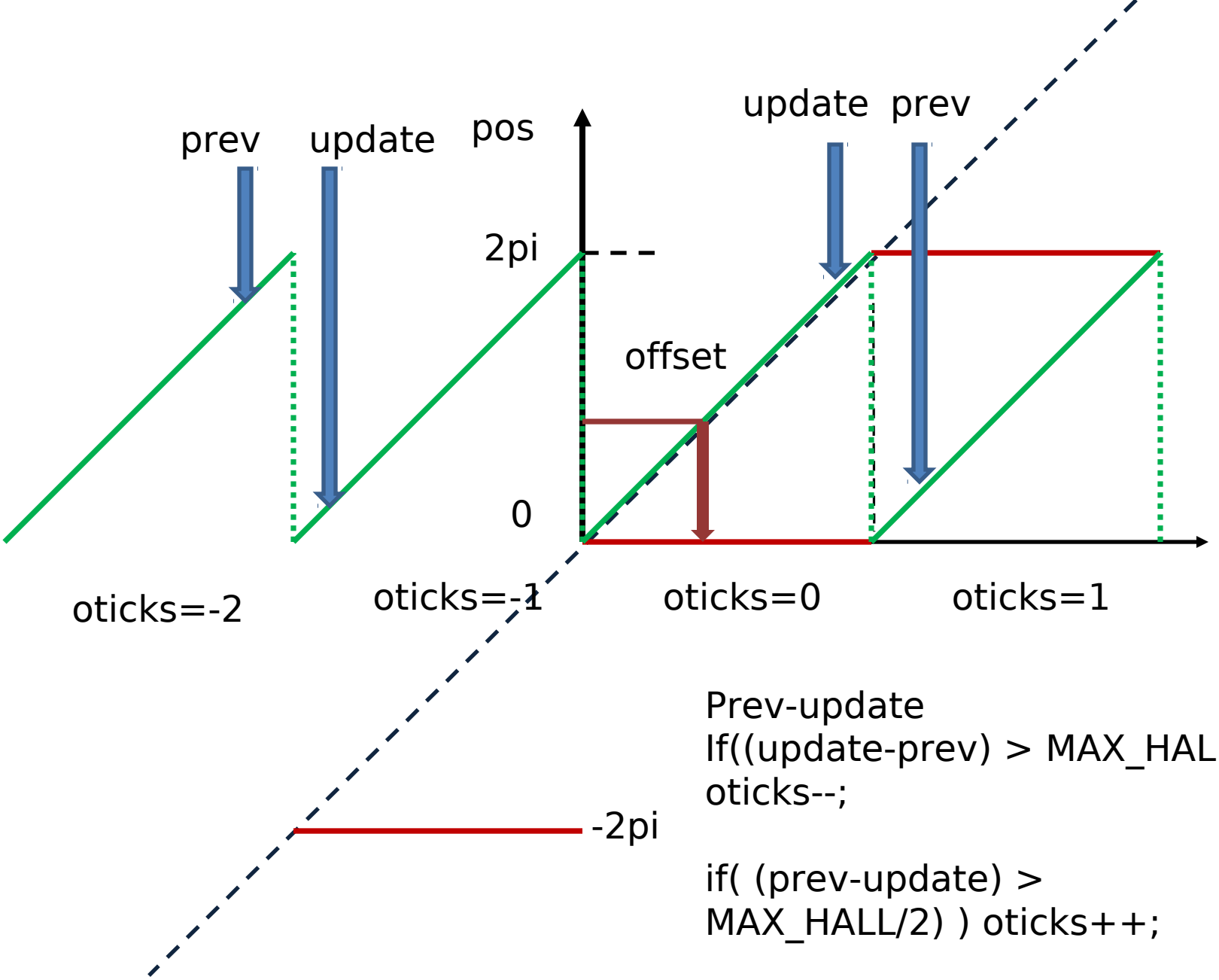
$vel_i = \delta_i / int_i$

$\delta$= unsigned 16 bit int value
int_i in ms
Python:
velocity
Delta (Hall counts)
Interval (ms)

**NOTE: using AustriaMicro Systems AS5048B 14 bit range converted to 16 bit unsigned**

A/D units per rev/sec * 2^16 per ms, scale by >>8
~43/256

$V_{input} = K_{EMF} * vel_i$




Prev-update
If((update-prev) > MAX_HALL/2)
oticks--;

if( (prev-update) > MAX_HALL/2) ) oticks++;

offset is subtracted from total position

# Obsolete PID Code pid-rf4.c

V_input is 0 by default. Set by pidSetInput()

```
pidVel[j].interpolate += pidVel[j].vel[index];
if (t1_ticks >= pidVel[j].expire)  // time to reach previous setpoint has passed
{               pidObjs[j].p_input += pidVel[j].delta[index];         //update to next set point
                pidObjs[j].v_input = (int)( pidVel[j].vel[(index+1) % NUM_VELS] * K_EMF); // update to next velocity

                … }

pidObjs[j].p_error = pidObjs[j].p_input + (pidVel[j].interpolate >> 8)  - motor_count[j];
pidObjs[j].v_error = pidObjs[j].v_input - measurements[j];

pid->p = (long)pid->Kp * pid->p_error;
pid->i = (long)pid->Ki  * pid->i_error;
pid->d=(long)pid->Kd *  (long) pid->v_error;

neglecting saturation:
pid->output = pid->feedforward + pid->p +
                                ((pid->i + pid->d) >> 4); // divide by 16
```

Experiment with VelociRoACH transmission

| Velocity (from hall angle sensor) Rad/sec | Back EMF (A/D units) |
|---|---|
| 50 | ~100 |
| 80 | ~140 |