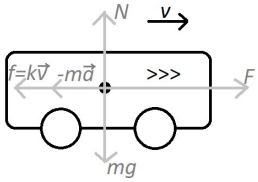
# Report for PID Digital Feedback Control Implementation on Robot Car

## Mathematical Modelling

For a robot car, its mechanical module can be developed from a first order force analysis:




Written in scalar form, $F_{motor} = ma + kv$.

Dividing two sides by $m$, the equation can be written as $\frac{F_{motor}}{m} = a + \frac{k}{m} * v$, where $F_{motor}$ acts as input while $a$ and $v$ act as output. Therefore, the math model can be written as:

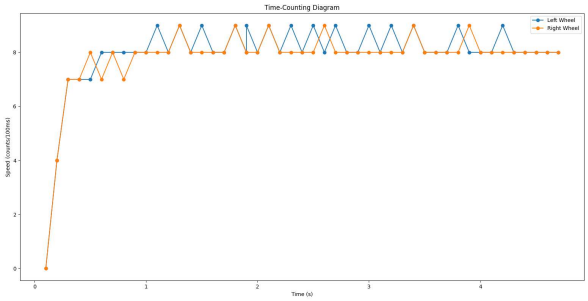
$$\boldsymbol{v(t)' + a * v(t) = b * F(t)} \text{ , or } \boldsymbol{y(t)' + a * y(t) = b * x(t)}.$$

Transferring the above equation to s-domain, we get:

$$\boldsymbol{(s + a)\, C(s) = b\, R(s)} \text{ , or } \boldsymbol{G(s) = \frac{C(s)}{R(s)} = \frac{b}{s + a}},$$

where $a = \frac{2\pi}{\tau}$, b = $\frac{\text{steady state under max PWM}}{\text{maximum PWM}} * a$, with τ being the 1st-order time constant.

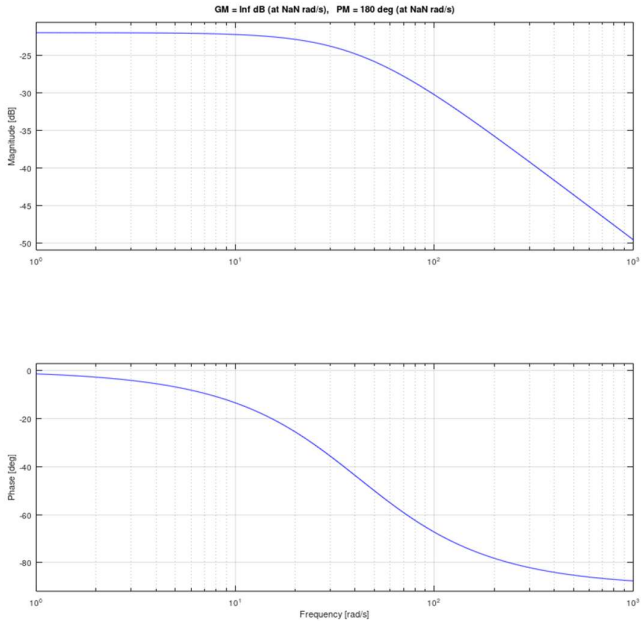
To obtain the time constant of the robot car, testing was conducted by applying full PWM power to the wheel and measure the time taken to achieve 63% response. The result is shown below.




From the graph, we can see that the time constant τ (time used to achieve an actual value of ~63% of the steady state) of the robot car system is around 0.15 seconds under the maximum PWM input. In the testing, the steady state is 10 ticks per 100 ms, and the effective output compare value for PWM duty cycle adjustment ranges from 0 to 100. Based on the above parameters, the robot car transfer function can be approximated on the first order as

$$G(s) = \frac{\text{steady state under max PWM}}{\text{maximum PWM}} * \frac{2\pi\tau^{-1}}{s + 2\pi\tau^{-1}} = \frac{8}{100} * \frac{2\pi * 6.667}{s + 2\pi * 6.667} = \frac{3.351}{s + 41.89}.$$

The bode plot of the system is shown as below.




To obtain a satisfactory transient response, $10*2\pi*\tau^{-1} = 418$ rad/s is chosen to be $\omega_{0dB}$. Currently without PID feedback compensation, $|\omega_{0dB}| \approx -21$ dB.

For the computer PID controlling to the above system, we need to push the chosen $\omega_{0dB}$ point to the gain of 0dB. The design starts from $Gc(s) = K_p * \frac{s + K_i/K_p}{s}$, where $K_p = |G(j\omega_{0dB})| \approx 21$ dB $\approx$ 11.2, and $\frac{K_i}{K_p} = \frac{\tau}{2\pi} = 41.89$ to achieve zero-pole cancellation with *G(s)*, which rises $K_i = 469.2$.

The resultant PID compensator in s-domain is

$$Gc(s) = \frac{11.2\,s + 469.2}{s}$$

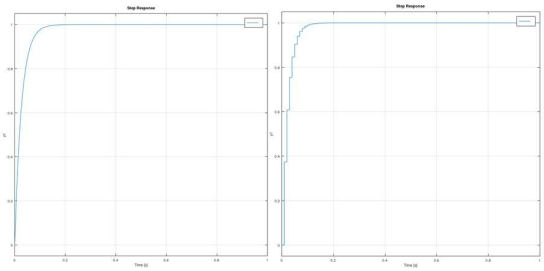
To ensure a satisfactory response in digitalization, the sampling rate should be at least 418 rad/s ($\omega_{0dB}$) or 66.5 Hz. Here a sampling rate of 100Hz is used to Transform *Gc* to the z-domain, gets

$$Gcz(z) = \frac{13.66\,z - 8.986}{z-1}, \text{ corresponding to } Gcz(z) = K_p + K_i\frac{T\,z}{z-1} = \frac{(K_p + K_iT)z - K_p}{z - 1},$$

Which yields $K_p = 8.986$, $K_i$ T = 4.674 in digitalized compensator.

The simulation results of step response to such a system in continuous and discrete time are given as below respectively. It can be seen that the system yields a good response both in

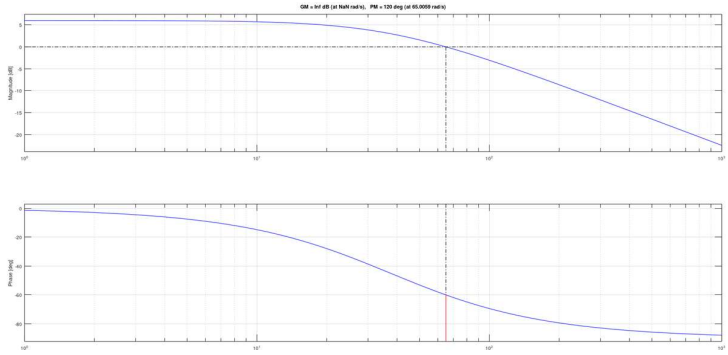
continuous and 100Hz discrete time domain.



In terms of the outer-loop control on the speed difference, the forward control plant is defined as two times of the feedback loop of G(s)*Gc(s), or

$$T(s) = 2 * \frac{G(s)*Gc(s)}{1+G(s)*Gc(s)} = \frac{75.06\,s+314}{s^2+79.42\,s+157}.$$

The bode plot of the defined forward transfer function *T(s)* is shown below.



From the graph, |T(0)| = 6.06 dB. From this we derive the $K_p$ of the outer-loop controller to be -6.06 dB or 0.5, $K_i = K_p = 0.5$. Thus, the PI compensator of the outer-loop is

$$Gc_T = K_p + \frac{s+K_i/K_p}{s} = \frac{0.5\,(s+1)}{s}, \text{ or } Gcz_T = \frac{0.5025\,z - 0.4975}{z-1} \text{ in z-domain (100 Hz sampling)}$$

## Feedback Control System Implementation and Optimization

From the z-domain representation of the PI controller $Gcz(z) = \frac{(K_p+K_iT)z - K_p}{z-1}$, we can derive the differential equation of a PI controller in the time domain as

$$y[k] - y[k-1] = (K_p + K_iT)\,e[k] - K_p\,e[k-1].$$

Given that $y[k] - y[k-1] = \Delta y$ and $e[k] - e[k-1] = \Delta e$, we get

$$\Delta y = K_p\,(e[k] - e[k-1]) + K_iT\,e[k] = K_p\,\Delta e + K_ie[k]$$

Summing up the two side of the equation in respective with k from k=0, we get

$$y[k] = K_p\, e[k] \;+\; K_i \sum_{i=0}^{k} e[i].$$

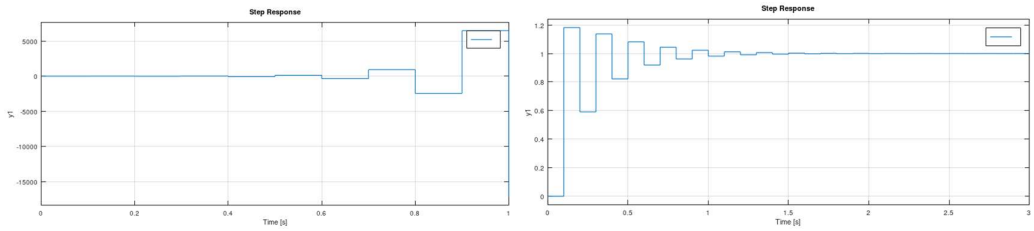
Based on this equation, the c program math engine is implemented as follows. (EmbedFire, n.d.)

```
float pid_run(pid* ctrl, float actual_val){
    ctrl->err_prev = ctrl->err;
    ctrl->actual_val = actual_val;
    ctrl->err = ctrl->target_val - actual_val;
    ctrl->integral += ctrl->err;
    ctrl->control_val = (ctrl->Kp)*(ctrl->err) + (ctrl->Ki)*(ctrl->integral) +
        (ctrl->Kd)*((ctrl->err) - (ctrl->err_prev));
    return ctrl->control_val;
}
```

In the implementation above, `ctrl` is an object-liked struct defined as

```
typedef struct{
    float target_val;   // target reference value
    float actual_val;   // actual value measured from output
    float control_val;  // the PID controlled output value
    float err;          // target_val - actual_val (at current sampling)
    float err_prev;     // target_val - actual_val (at previous sampline)
    float Kp, Ki, Kd;   // controlling parameters. Ki = KiT
    float integral;     // the accumulated error
} pid;
```

It is worth noting that, when implementing the above control system to the physical robot car, the encoder disks of the car are low in grid density (resolution), and therefore increasing the sampling rate will lead to a very low speed value resolution. Empirically a sampling rate of more than 10 Hz is not practically feasible on the given robot car. This means, the control system derived and simulated should be re-assessed under a sampling rate of 10Hz. However, a sampling rate of 10Hz is not acceptable in the theoretical analysis. From the simulation result, the response of the system is very bad. After tuning the controller parameters, it is found that a compensator of $Gcz(z) = \frac{15\,z - 5}{z - 1}$ can obtain an acceptable response under 10Hz sampling rate in the simulation.




Satisfactory results were obtained on the modified $Gcz(z)$. However, due to non-ideal effects of the physical robot car, theoretical-level optimization is no longer feasible at this stage.

- Motors are poor in physical property. An example is the loose and unstable gear-occlusion.
- The surface friction μ is low, leading to encoder-undetectable wheel sliding.
- The math model assumes a linear resistance, which is obviously not the case in reality.

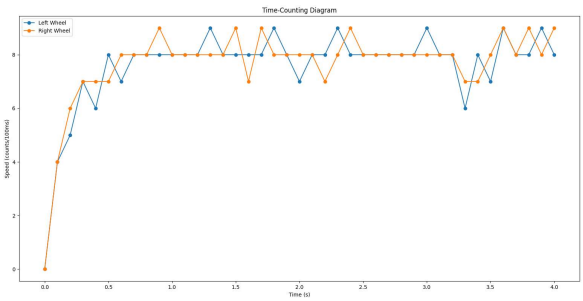
- To travel through the 2.4m road in 3s, the car has to run close to its max speed, giving it less room for feedback adjustment and going overshoot, and overshoot is much less a problem.
- The PWM output is clipped in the range of [0, 100] in the physical car implementation, thus exhibits much less overshoot with contrast to PWM output of [-∞, +∞] in the simulation.

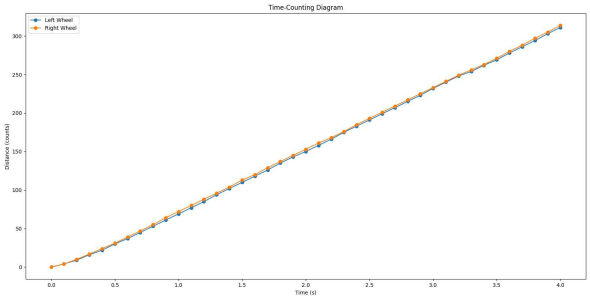
As the compensator is obtained from simulation, tuning has been conducted using empirical method on the physical robot car, by

1. Setting $K_i$ and $K_d$ to be 0, adjusting $K_p$ so that output is close to reference but not overshoot;
2. Adjusting $K_i$ so that the settling time is in an acceptable range ($K_i$ hereby refers to $K_i T$);
3. Adjusting $K_d$ to mitigate the overshoot problem.

After tuning on the physical car, the optimized parameters arrive at $K_p = 10$, $K_i = 5$. As after tuning the PI the robot car can already drive in a straight line at a constant speed, $K_d$ is not tuned in the field testing and remains 0. Outer-loop control against speed deviation is also not required.

As of the stage of successful feedback task implementation, the corresponding PID controller equation is $Gcz(z) = \frac{15z - 10}{z - 1}$. It is interesting to find out that the final PID employed in the physical car with sampling rate of 10Hz is very close to that of simulation at a sampling rate of 100Hz. The diagrams for encoder counter reading with respect to time are given as follows.




**Reference**

EmbedFire. (n.d.). *PID 算法的通俗解说*. Retrieved May 24, 2024, from https://doc.embedfire.com/motor/motor_tutorial/zh/latest/improve_part/PID_detailed.html