PID

Szabályozó

(3. csoport)

Szimulációs technikák

(NGB_IN040_1)

Tartalomjegyzék

Források

http://hu.wikipedia.org/wiki/PID_szab%C3%A1lyoz%C3%B3

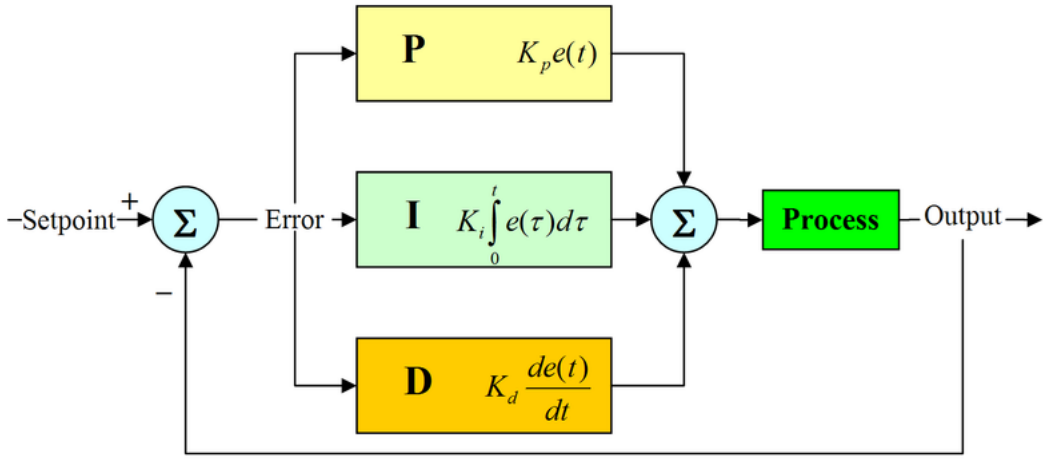
https://www.udacity.com/file?file_key=agpzfnVkYWNpdHl1ckcLEgZDb3Vyc2UiBWNzMzczDAsSCUNvdXJzZVJldiIHZmViMjAxMgwLEgRVbml0GLG0EwwLEgxBdHRhY2hlZEZpbGUY4ZgXDA

http://www.timgittos.com/learning/udacity-ai-for-robotics/week-5/

PID szabályozó

A **PID szabályozó** egy lineáris rendszerek szabályozásánál gyakran alkalmazott, párhuzamos kompenzáción alapuló szabályozótípus. A PID rövidítés a szabályozó elvére utal, a szabályozó által kiadott végrehajtójel,

- a hibajellel (**P**: *proportional*),
- a hibajel integráljával (**I**: *integral*), valamint
- a hibajel változási sebességével, deriváltjával (**D**: *derivative*) arányos tagokból adódik össze.



Matematikai leírása: A PID szabályozó bemenetét $e(t)$-vel, kimenetét $u(t)$-vel jelöjük:

$$u(t) = K_P e(t) + K_I \int e(t)\mathrm{d}t + K_D \frac{\mathrm{d}}{\mathrm{d}t} e(t).$$

Részei:

- Arányos rész

$$P_{\text{out}} = K_p\, e(t)$$

- Integráló rész

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

- Differenciáló tag

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

Error:

A jelenlegi pozíció és a referencia útvonal közötti különbség mértéke. Jele: d

Arányos-tag:

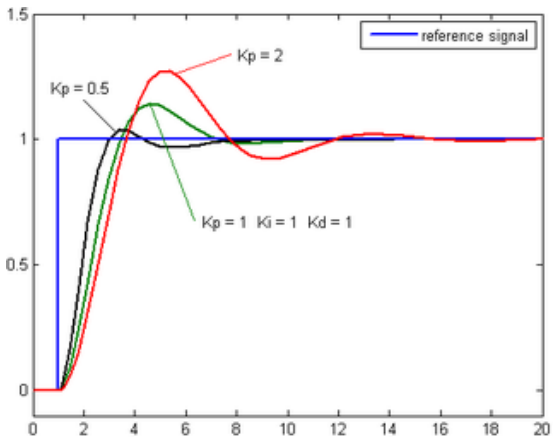
Az arányos tag arányos a hibajellel (d - error). Ha az erősítés túl nagy, instabilitási problémák léphetnek fel, míg ha túl kicsi, akkor kevésbé érzékeny a szabályzó kör.

Integráló-tag:

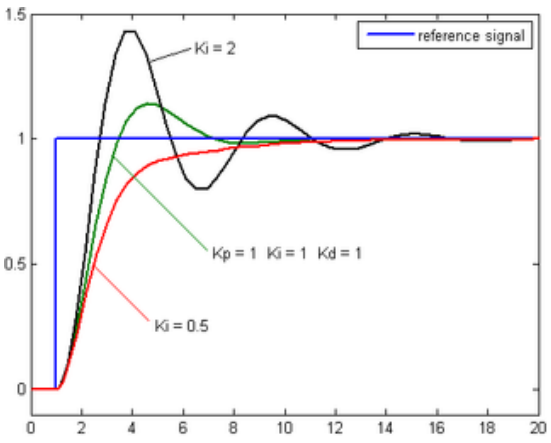
Az integráló tag (amennyiben hozzáadjuk az arányos-taghoz) gyorsítja a folyamat mozgását az alapérték (útvonal - trajektória) felé és kiküszöböli a maradó szabályozási eltérést. Mivel az integráló tag a múltbeli összegzett hibajelekkel áll összefüggésben, a jelenben túllendülést produkálhat a kívánt értékhez képest.

Differenciáló-tag:

A differenciáló tag késlelteti a szabályozó kimenetének változásának a mértékét. Ezen túl a differenciáló-tag szerepe, hogy csökkentse az integráló-tag túllendülését, valamint fokozza a szabályozási folyamat stabilitását.

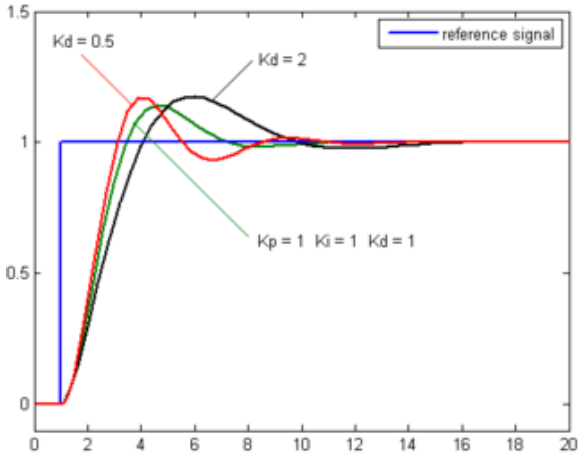


Arányos-tag



Integráló-tag

Differenciáló-tag



A Twiddle algoritmus

A munkánk során a Twiddle algoritmust használtuk fel, hogy az optimális kimenetet érjük el. Ez az algoritmus optimalizálja a három tag (Ki, Kp, Kd) értékét szimulációk segítségével. Kezdetben három véletlenszerű értékkel kerül inicializálásra, majd ezzel szimulálja a robot mozgását és ennek segítségével határozza meg a megfelelő elfordulás mértékét. Ebből tanulva módosítja az erősítési tényezőket. A cél az útvonaltól való eltérés minimálisra történő csökkentése.

Mielőtt a robot tényleges mozgást végezne a Twiddle (kis szimuláció) algoritmussal meghatározzuk a megfelelő erősítési tényezőket, ezután a kapott értékeket használjuk fel a robot irányításához (nagy szimuláció).

A szimuláció pontossága egy adott küszöbértéktől (treshold) függ. A mi esetünkben ez az érték: 0,00001.

A végtelen ciklus elkerülése végett, egy feltétel segítségével leállítjuk a szimulációt egy adott határérték elérésekor. Ettől nagymértékben függ a szimuláció pontossága.

```
while(sumdp(dp) > 0.00001 ) {
  if(j>500){
        elert=true;break;
  }
}
```

treshold (küszöbérték)

futási határérték

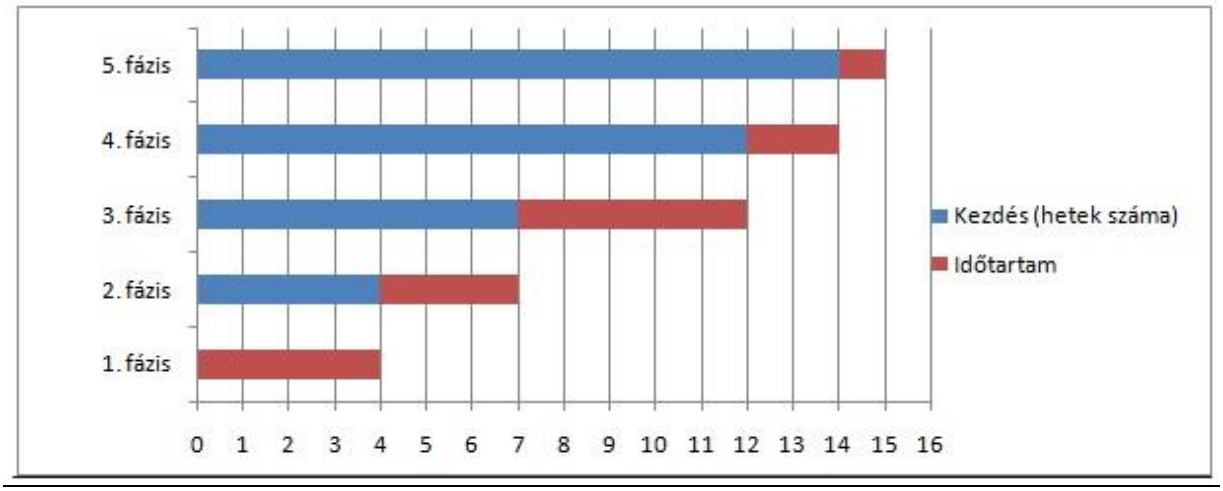
Ezen felül további funkciókkal egészítettük ki, segítve ezzel a szimuláció hatékonyságát.

Már korábban szereplő bemeneti értékek esetén egy csv fájl segítségével kiszűrjük az értékek újra számolását. Ezzel időt és erőforrást megtakarítva.

Valamint a robot kerekeinek elmozdulását maximum 45 fokra korlátoztuk, a nagymértékű elfordulás elkerülése érdekében. Az alábbi kódrészlet szemlélteti:

```
double max_steering_angle = Math.PI/4.0;
if (steering > max_steering_angle){
steering =max_steering_angle; }
if (steering < -max_steering_angle){
steering = -max_steering_angle; }
```

Munkafolyamat bemutatása Gantt-diagram segítségével



1. fázis: Adatgyűjtés, követelmények specifikálása, a csapattagok munkájának meghatározása.

2. fázis: Kutatások, adatgyűjtés eredményének kiértékelése, forráskód elkészítésének kezdete, algoritmusok összegyűjtése.

3. fázis: Algoritmusok implementálása a forráskódba, forráskód optimalizálása.

4. fázis: A program funkcionalitásának bővítése, tesztelés-debuggolás, validálás.

5. fázis: Dokumentáció elkészítése, tapasztalatok megbeszélése.