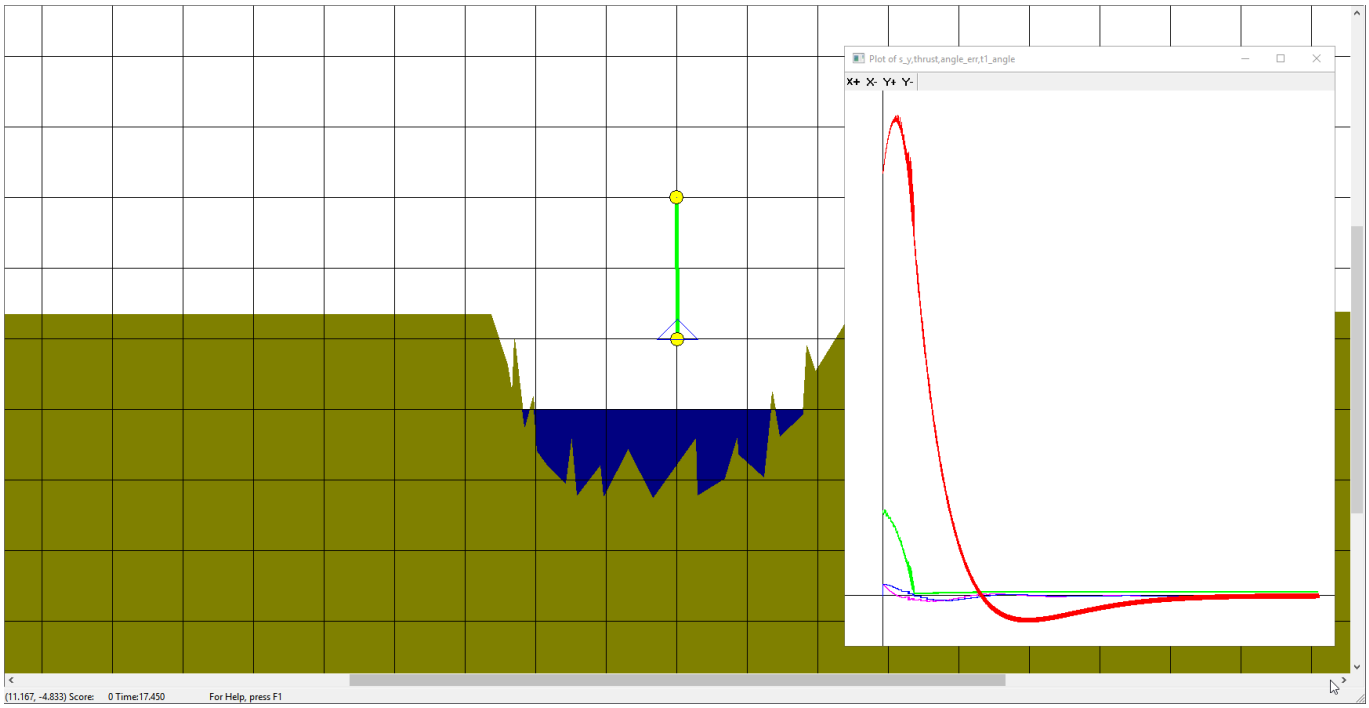
Symulacja: kontrolera PID w programie SimStructure



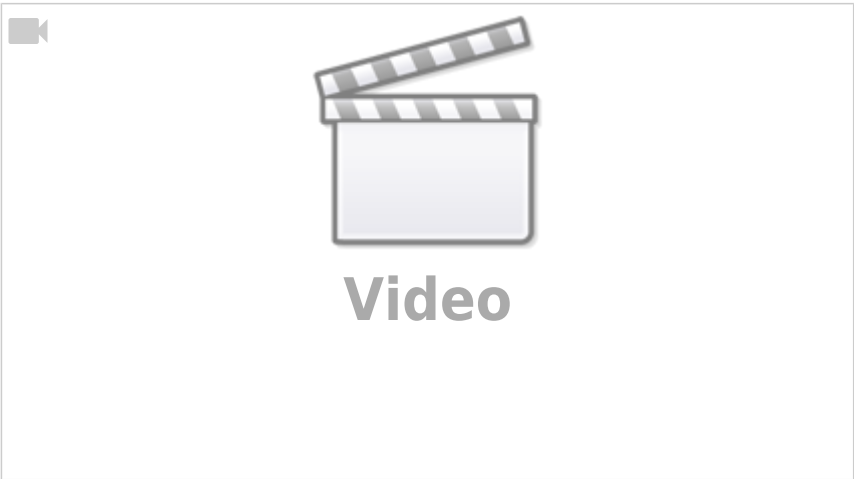
pliki:

- simstrucuture.zip
- ardugeek_rocket1.zip

Fajny symulator do nauki zasad działania kontrolera PID: https://tools.softinery.com/PIDSIM/

Autor programu SimStructure: Terry A. Davis

Źródło:



Założenie

Symulacja ma pokazać zastosowanie kontrolera PID w rakiecie która ma mieć kompensację zmian przyspieszenia oraz ustawienia kątowego. Symulacja jest wykonana w dwóch wymiarach i obliczenia są wykonywane w dwóch wymiarach.

Deklaracja zmiennych

```
 double desired_y = 10, error_y, kd_y = 1, kp_y = 1, ki_y = 0.3; signal s_y; signal thrust; signal t1_angle; integrator i_y;
```

desired_y – zadana wartość wysokości (initialnie 10).

error_y – różnica między wartością zadaną a aktualną.

kp_y, kd_y, ki_y – współczynniki regulatora PID dla osi y (proporcjonalny, różniczkujący, całkujący).

s_y, thrust, t1_angle – sygnały służące do dalszego przetwarzania i wizualizacji.

i_y – wewnętrzny akumulator (całkujący) regulatora.

Regulator PID dla osi _Y

```
 desired_y = 15 + 5 * t; error_y = desired_y - p1.y; s_y = error_y; i_y = error_y; t1.thrust = t1.saturation * ( kp_y * error_y - kd_y * p1.dy + ki_y * i_y ); thrust = t1.thrust / 100000;
```

Linia

```
desired_y = 15 + 5 * t;
```

powoduje zmianę zadanej wysokości liniowo w czasie (t to czas symulacji).

Obliczenie błędu: ;error_y: Różnica między wysokością zadaną a bieżącą (p1.y).

Całkowanie błędu: ;i_y: Proste całkowanie przez przypisanie błędu (może być rozszerzone o sumowanie w pętli).

Wyjście regulatora: ;t1.thrust = faktor_saturacji × (P · error – D · prędkość + I · całka).

Skalowanie sygnału siły ciągu (thrust) dzieleniem przez 100 000, by uzyskać odpowiednie jednostki/zakres.

Regulator PID dla kąta

```
 double a, kp_a = 1.0, kd_a = 0.01, ki_a = 0.001, a_err; []a = pi - pi/180 *
b1.heading + 10 * ( key2("1") - key2("2") ); signal angle_err; angle_err =
50 * a; integrator i_a; i_a = a / 10;

t1.angle = kp_a * a
- kd_a * b1.spin
+ ki_a * i_a;
t1_angle = 50 * t1.angle;
```

a – błąd kąta: różnica między kątem referencyjnym (pi) a aktualnym (b1.heading w radianach), plus korekta z klawiatury (przyciski „1” i „2”).

kp_a, kd_a, ki_a – współczynniki PID dla kąta.

angle_err – skalowany błąd kąta do wizualizacji.

i_a – całka błędu kąta (tu zaledwie a/10, nie sumująca).

Obliczenie wyjścia regulatora kąta: ;t1.angle = P · a – D · obrót + I · i_a

t1_angle – dodatkowe skalowanie sygnału kąta.

Wizualizacja i wykresy

```
 @{ line(p1.x-1000, desired_y, p1.x+1000, desired_y, blue, #3); text(#10,
#50, "Thrust: %12.6f N", t1.thrust); text(#10, #10, "Heading:%12.6f degree",
b1.heading); text(#10, #30, "Angle: %12.6f rad", a); text(#10, #70,
"Integrator: %12.6f", i_y); plot s_y, thrust, angle_err, t1_angle; }
```

Funkcja

```
line()
```

rysuje poziomą linię na wysokości zadanej w odniesieniu do pozycji p1.x.

Kilka wywołań

```
text()
```

wyświetla na ekranie:

wartość siły ciągu (t1.thrust),

aktualny kąt (b1.heading),

błąd kąta (a),

wartość całki błędu osi y (i_y).

```
plot()
```

generuje wykresy kolejnych sygnałów:

s_y – sygnał błędu wysokości,

thrust – siła ciągu,

angle_err – błąd kąta (skala),

t1_angle – sygnał wyjściowy regulatora kąta.

Pełen kod sterowania

Kod:

```
double desired_y=10,error_y,kd_y=1,kp_y=1,ki_y=0.3;
signal s_y;
signal thrust;
signal t1_angle;
integrator i_y;

desired_y = 15+5*t;


error_y = desired_y-p1.y;
s_y = error_y;
i_y = error_y;
t1.thrust = t1.saturation*(kp_y*error_y-kd_y*p1.dy+ki_y*i_y);
thrust = t1.thrust/100000;
double a,kp_a=1.0,kd_a=0.01,ki_a=0.001,a_err;
[]a = pi-pi/180*b1.heading+10*(key2("1")-key2("2"));
signal angle_err;
angle_err = 50*a;
integrator i_a;
i_a = a/10;

t1.angle = kp_a*a-kd_a*b1.spin+ki_a*i_a;
t1_angle = 50*t1.angle;
@{
    line(p1.x-1000,desired_y,p1.x+1000,desired_y,blue,#3);
    text(#10,#50,"Thrust:  %12.6f N",t1.thrust);
    text(#10,#10,"Heading:%12.6f degree",b1.heading);
    text(#10,#30,"Angle:  %12.6f rad",a);
    text(#10,#70,"Integrator: %12.6f",i_y);
    plot s_y, thrust, angle_err, t1_angle;
}
```