SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI PROJEKT

Sprječavanje „efekta zaleta“ pri upravljanju gibanjem svesmjernog mobilnog robota Anda

Ante Pavlić

Katarina Lukašević

Mentor: *Marija Seder*

Zagreb, lipanj, 2022.

1. Uvod

Svesmjerni mobilni robot Anda je platforma s četiri švedska kotača koji mu omogućuju kretanje u svim smjerovima bez potrebe za okretanjem oko vertikalne osi. Svaki kotač neovisno je upravljan PI regulatorom brzine vrtnje.

Prilikom upravljanja robotom dolazi do nepoželjnog efekta zaleta (engl. *windup effect*). Efekt zaleta definiramo kao stanje kumulativne greške integracijskog djelovanja regulatora što dovodi do zasićenja aktuatora radi čega sustav prestaje biti u kontroliranom stanju. Za praktičnu primjenu regulator se mora nadopuniti funkcijom za sprječavanje ove pojave, tj. filtrom za sprječavanje efekta zaleta (engl. *anti-windup filter*).

Opisane funkcionalnosti mogu se implementirati na robota putem koda u C programskom jeziku te se mogu simulirati pomoću Simulink sheme regulatora s anti-windup filtrom.

2. Vrste filtara za sprječavanje efekta zaleta

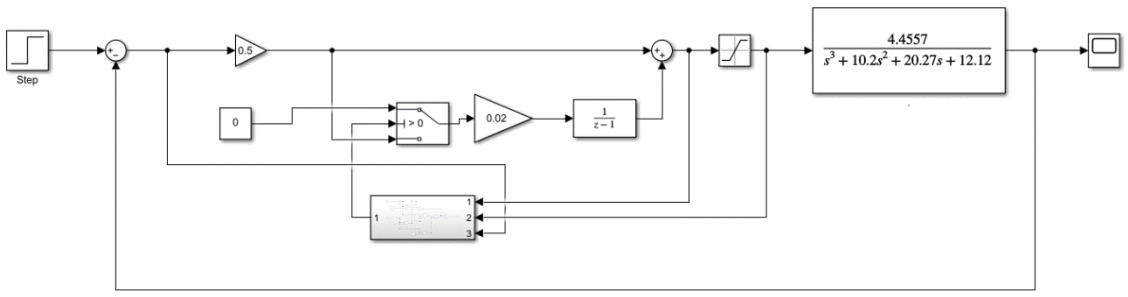
Za sprječavanje efekta zaleta potrebno je ugraditi dodatnu funkciju regulatora. Razmatrane su tri metode za sprječavanje efekta:

1. Postupak uvjetnog integriranja,
2. Postupak povratnog integriranja,
3. Automatsko resetiranje.

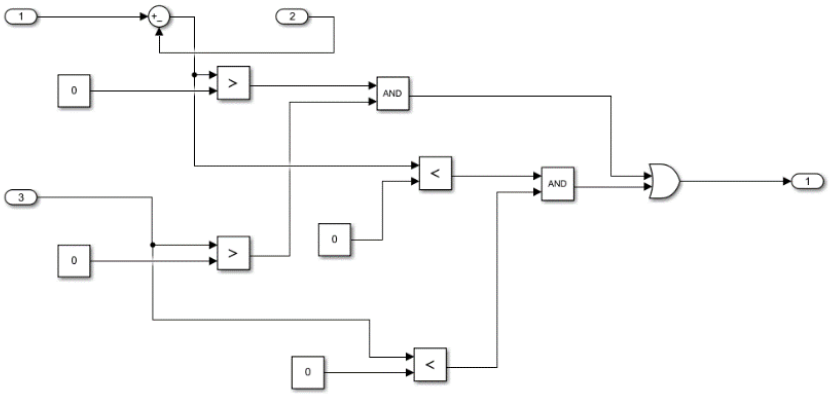
2.1. Postupak uvjetnog integriranja

Moguće je implementirati preklopnik, tj. sklopku kao dio regulatora sustava koji, kad se izvršni član nađe u zasićenju, ulaz integratora stavlja u nulu.

Ovaj postupak efektivno predstavlja „zamrzavanje“ integratora ako i samo ako se ispune određeni uvjeti. (slika 2.2.)

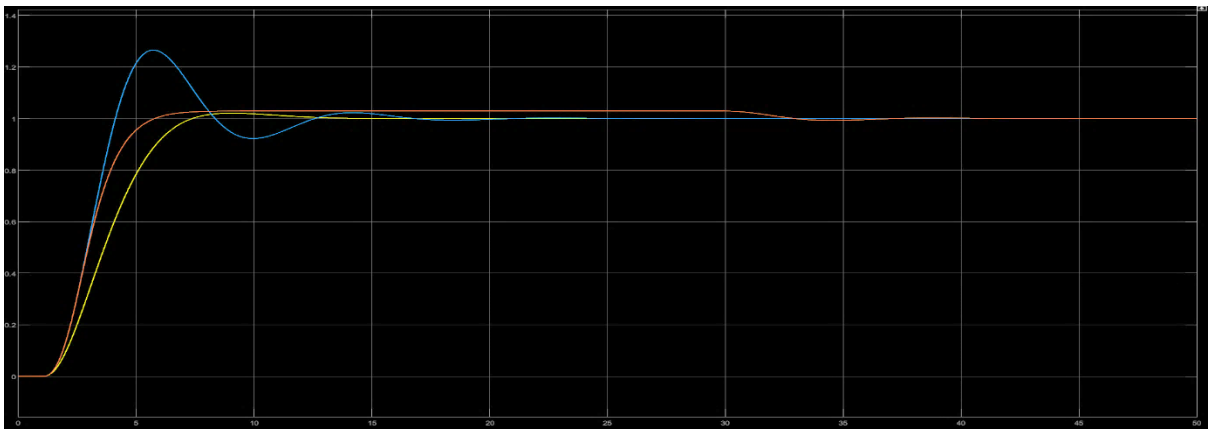


Slika 2.1.. – Simulink shema regulatora s implementiranim anti-windup filtrom s metodom uvjetnog integriranja



Slika 2.2. – uvjet za uključenje sklopke u filtru s uvjetnom integracijom

Simuliranjem priložene sheme dobivamo zadovoljavajuć odziv nakon implementacije filtra (žuta linija na grafu) naspram regulatora bez filtra i saturacije (plava) te regulatora bez filtra, ali s saturacijom (crvena).

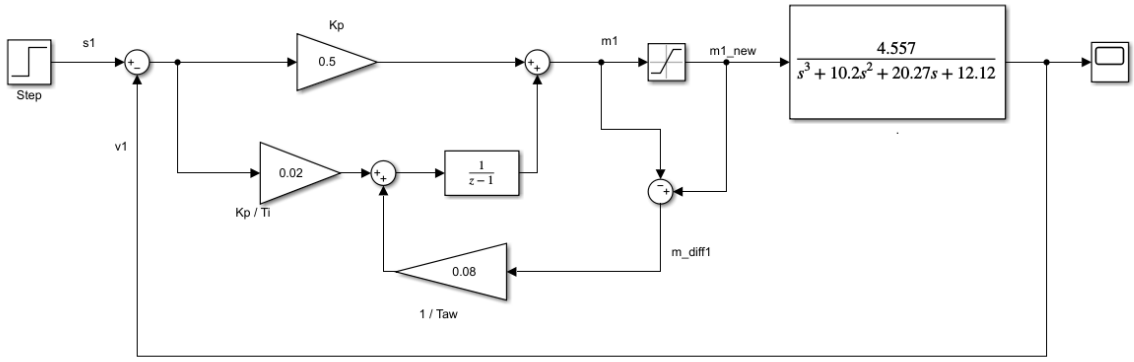


Slika 2.3. – odziv sustava s i bez implentacije filtra

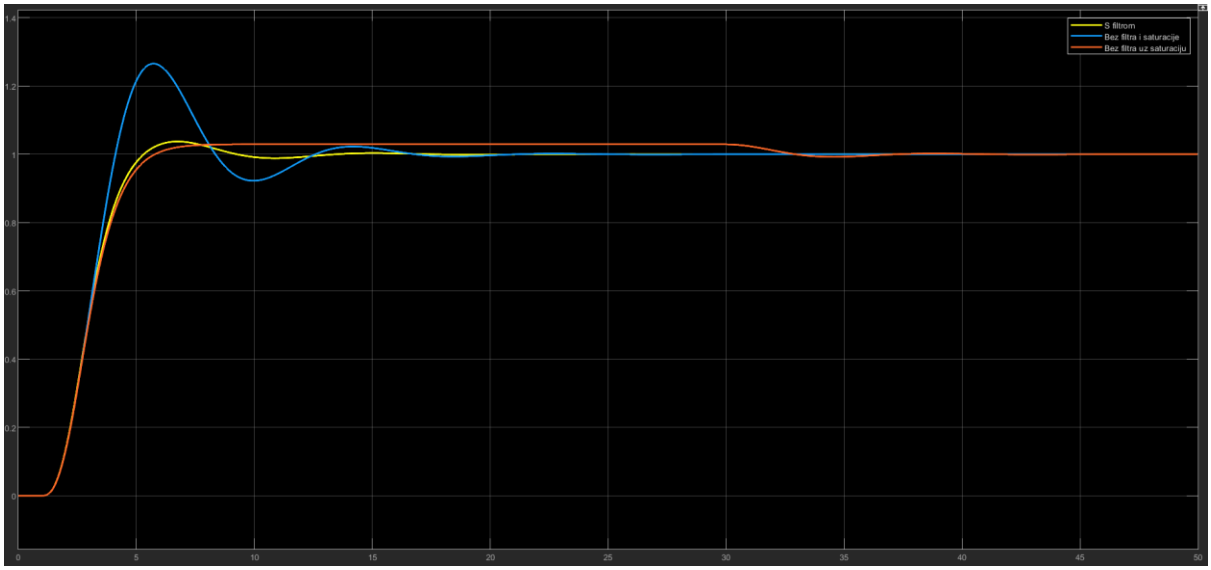
2.2 Postupak povratnog integriranja

U ovisnosti o predznaku regulacijskog odstupanja, s obzirom na dodatni izračun integralnog dijela regulatora, izlaz integratora nastoji se uvećati ili umanjiti.

Izlaz integratora se smanjuje kada je upravljački signal (m1) u zasićenju, no ne mijenja se kada je upravljački signal u linearnom području (m1 = m1_new). (Slika 2.4.)



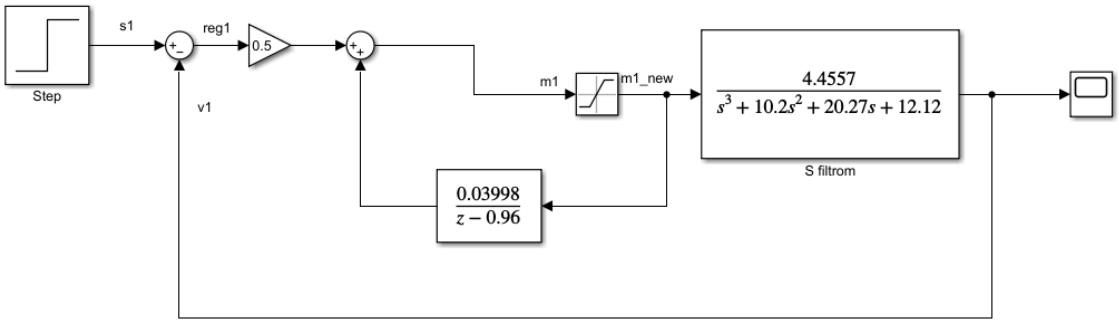
Slika 2.4.. – Simulink shema regulatora s implementiranim anti-windup filtrom s metodom povratnog integriranja



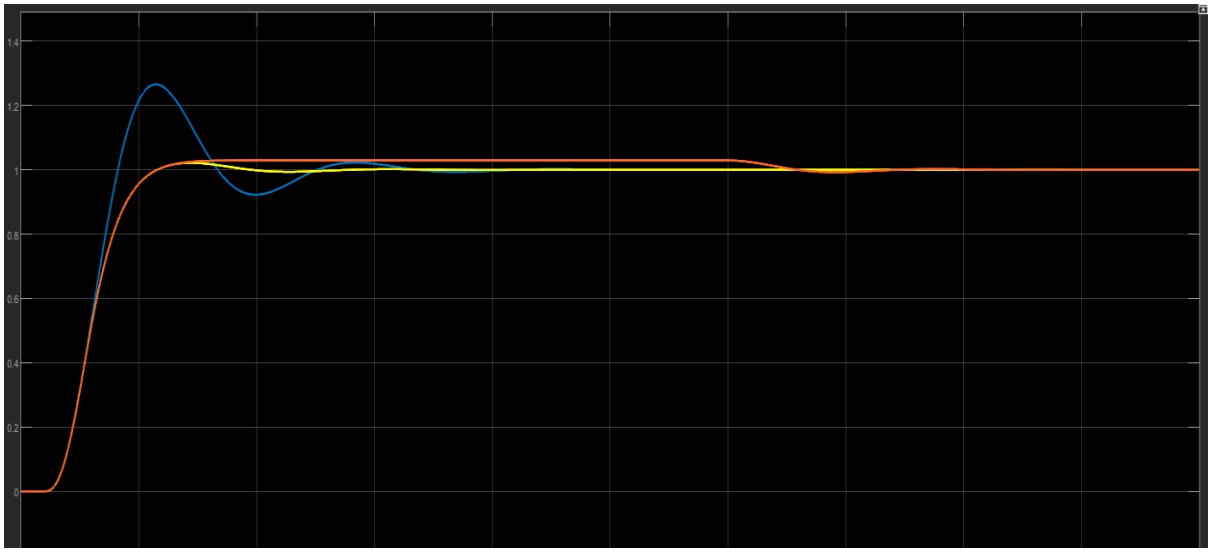
Slika 2.5. – Odziv regulatora s implementiranim filtorm (žuta linija) i bez (plava i crvena)

2.3 Postupak automatskog resetiranja

U postupku automatskog resetiranja PI regulator prikazujemo u serijskom obliku s pozitivnom povratnom vezom (Slika 2.6.). Ako se integralni dio regulatora pronađe izvan postavljenih granica zasićenja dolazi do prividnog prekida povratne veze (integrator se održava u saturiranoj vrijednosti – greška se ne akumulira).



Slika 2.6. - Simulink shema regulatora s implementiranim anti-windup filtrom s metodom automatskog resetiranja



Slika 2.7. - Odziv regulatora s implementiranim filtorm (žuta linija) i bez (plava i crvena)

2.4. Analiza rješenja

Uspoređujući odzive svih triju metoda može se provesti grafička analiza rješenja i u teoriji odrediti najoptimalnija metoda implementacije filtra za sprječavanje efekta zaleta.

Metodom uvjetnog integriranja uvećava se vrijeme rasta odziva sustava naspram vremenu rasta odziva sustava bez filtra i sa saturacijom.

Metodom povratnog integriranja uvelike se smanjuje vrijeme rasta odziva, no na jednom djelu ono prerasta vrijeme rasta odziva sustava bez filtra i sa saturacijom. Iako se radi o maloj vrijednosti preferira se rješenja bez toga.

Povratno integriranje i automatsko resetiranje imaju slična vremena rasta. Vrijeme pada kod metode povratnog integriranja zanemarivo je kraće, no sveukupno gledajući, automatsko resetiranje, u teoriji, smatra se najboljim rješenjem za implementaciju na Andi.

3. Implementacije metoda u C programskom jeziku

Svesmjerni mobilni robot Anda zaustavlja svoje kretanje kada senzor očita predmet ili osobu u nedozvoljenoj blizini robota. U slučaju kada je robot zaustavljen, no prima naredbe za kretanje moguć je nastanak problema da robot zbraja dobivene brzine te kada senzor prestane očitavati prepreku robot se pokreće većom brzinom od željene.

Za rješenje ovog problema uvedena je nova varijabla (bool) zastavica koja služi kao upozorenje robotu da mu je poslana naredba za kretanje, a nije u pokretu. Zastavica je prvotno postavljena u vrijednost false, no u situaciji kada se robot pokreće iz mirovanja postavlja se u true. Ako je zastavica postavljena u true te robot prima naredbe za kretanje, ali se ne kreće radi utjecaja senzora, onda se vrijednost ulaza u integrator resetira.

Sljedeći kod predstavlja upravljačku petlju robotu bez implementiranih metoda.

```
#include "Main.h"

void ControlLoop ( void )
{
    forward = GetJoystickAnalog( 1 , 2 ) ;  //ucitavanje komandi s joystick-a
    right = GetJoystickAnalog( 1 , 1 ) ;    //ucitavanje komandi s joystick-a
    yaw = GetJoystickAnalog( 1 , 4 ) ;      //ucitavanje komandi s joystick-a

    //implementacija mrtve zone na joysticu
        //uvijek postoji poneki mali signal oko nule
    if ( forward>10 ){
        forward=forward-10 ;
    } else if ( forward>-10 ){
        forward=0 ;
    } else {
        forward=forward+10 ;
    }
    if ( right>10 ) {
        right=right-10 ;
    } else if ( right>-10 ){
        right=0 ;
    } else {
        right=right+10 ;
    }
    if ( yaw>10 ) {
        yaw=yaw-10 ;
    } else if ( yaw>-10 ){
        yaw=0 ;
    } else {
        yaw=yaw+10 ;
    }
```

```
//citanje brzina s enkodera
e1 = GetQuadEncoder ( 1 , 2 );  //brzina prvog kotaca
e2 = GetQuadEncoder ( 3 , 4 );  //brzina drugog kotaca
e3 = GetQuadEncoder ( 5 , 6 );  //brzina treeg kotaca
e4 = GetQuadEncoder ( 7 , 8 );  //brzina etvrtog kotaca

//raunanje promjene brzine kotaa
v1=e1-e1_old ;
v2=e2-e2_old ;
v3=e3-e3_old ;
v4=e4-e4_old ;
e1_old=e1 ;
e2_old=e2 ;
e3_old=e3 ;
e4_old=e4 ;

readUART();

//racunanje referentne brzine za svaki kotac
s1=(forward-right-yaw)/4 + r1_uart;
s2=(forward+right+yaw)/4 + r2_uart;
s3=(forward+right-yaw)/4 + r3_uart;
s4=(forward-right+yaw)/4 + r4_uart;


//racunanje regulacijskog odstupanja za svaki kotac
reg1=v1-s1;
reg2=v2-s2;
reg3=v3-s3;
reg4=v4-s4;
```

```
//racunanje sume regulacijskih odstupanja
e1_integral+=reg1;
e2_integral+=reg2;
e3_integral+=reg3;
e4_integral+=reg4;

if (s1!=0 && v1==0 && s2!=0 && v2==0 && s3!=0 && v3==0 && s4!=0 && v4==0 &&
    zastavica == true){
  e1_integral = 0;
  e2_integral = 0;
  e3_integral = 0;
  e4_integral = 0;
}
```

```
if (s1==0 && v1==0){
    e1_integral = 0;
}
if (s2==0 && v2==0){
    e2_integral = 0;
}
if (s3==0 && v3==0){
    e3_integral = 0;
}
if (s4==0 && v4==0){
    e4_integral = 0;
}

if (s1!=0 && v1==0 && s2!=0 && v2==0 && s3!=0 && v3==0 && s4!=0 && v4==0){
  zastavica = true;
}
else {
  zastvica = false;
}

//PI controller + pojaavanje reference
fm1 =-s1*FFGAIN+(v1-s1)*PGAIN+e1_integral*IGAIN;
fm2 =-s2*FFGAIN+(v2-s2)*PGAIN+e2_integral*IGAIN;
fm3 =-s3*FFGAIN+(v3-s3)*PGAIN+e3_integral*IGAIN;
fm4 =-s4*FFGAIN+(v4-s4)*PGAIN+e4_integral*IGAIN;


//realni brojevi u integer, zaokruzivanje
m1=(int)(fm1+0.5);
m2=-(int)(fm2+0.5);
m3=(int)(fm3+0.5);
m4=-(int)(fm4+0.5);
```

```
//svaki motor ima neku zonu neosjetljivosti na male brzine
    //linearizacija oko nule
if ( m1>0 ){
    m1=m1+15 ;
} else if (m1<0) {
    m1=m1-6 ;
}
if ( m2>0 ){
    m2=m2+15 ;
} else if (m2<0) {
    m2=m2-6 ;
}
if ( m3>0 ){
    m3=m3+15 ;
} else if (m3<0) {
    m3=m3-6 ;
}
if ( m4>0 ){
    m4=m4+15 ;
} else if (m4>0) {
    m4=m4-6 ;
}
```

```
// ogranienje brzine motora izmeu -127 i 127 (to je oko 4 m/s)
//saturacija
if ( m1 > 127 ){
    m1 = 127 ;
}
if ( m1 < -127 ){
    m1 = -127 ;
}
if ( m2>127 ){
    m2=127 ;
}
if ( m2<-127 ){
    m2=-127 ;
}
if ( m3>127 ){
    m3=127 ;
}
if ( m3<-127 ){
    m3=-127 ;
}
if ( m4>127 ){
    m4=127 ;
}
if ( m4<-127 ){
    m4=-127 ;
}

//slanje upravljake brzine na motore
SetMotor ( 2 , m1 ) ;
SetMotor ( 3 , m2 ) ;
SetMotor ( 4 , m3 ) ;
SetMotor ( 5 , m4 ) ;
```

3.1. Povratno integriranje

U dijelu koda za saturaciju definirane su nove varijable m*_new umjesto promjene iste varijable m* (u kodu bez implementiranih filtara linije 143-172).

Varijabla m*diff predstavlja razliku signala nakon i prije saturacije koja je negativna ako signal prelazi razinu saturacije, odnosno jednaka je nuli ako je signal unutar granica. (Slika 3.2.)

Takva se varijabla zbraja s varijablom e*_integral (suma regulacijskih odstupanja), odnosno smanjuje se vrijednost integralnog dijela regulatora ako se sustav nalazi u zasićenju. (Slika 3.1.) *(I_Taw = 0.08)

```
//PI controller + pojavljanje reference
fm1 =-s1*FFGAIN+(v1-s1)*PGAIN+(e1_integral + m1diff*I_Taw)*IGAIN;
fm2 =-s2*FFGAIN+(v2-s2)*PGAIN+(e2_integral + m2diff*I_Taw)*IGAIN;
fm3 =-s3*FFGAIN+(v3-s3)*PGAIN+(e3_integral + m3diff*I_Taw)*IGAIN;
fm4 =-s4*FFGAIN+(v4-s4)*PGAIN+(e4_integral + m4diff*I_Taw)*IGAIN;
```

Slika 3.1. – funkcija regulatora s implementacijom povratnog integriranja

```
// ogranienje brzine motora izmeu -127 i 127 (to je oko 4 m/s)
//saturacija
if ( m1 > 127 ){
    m1_new = (int)127 ;
}
if ( m1 < -127 ){
    m1_new = (int)-127 ;
}
if ( m2>127 ){
    m2_new = (int)127 ;
}
if ( m2<-127 ){
    m2_new = (int)-127 ;
}
if ( m3>127 ){
    m3_new = (int)127 ;
}
if ( m3<-127 ){
    m3_new = (int)-127 ;
}
if ( m4>127 ){
    m4_new = (int)127 ;
}
if ( m4<-127 ){
    m4_new = (int)-127 ;
}

m1diff = (int)(m1_new-m1);
m2diff = (int)(m2_new-m2);
m3diff = (int)(m3_new-m3);
m4diff = (int)(m4_new-m4);
//slanje upravljake brzine na motore
SetMotor ( 2 , m1_new ) ;
SetMotor ( 3 , m2_new ) ;
SetMotor ( 4 , m3_new ) ;
SetMotor ( 5 , m4_new ) ;
```

Slika 3.2.

3.2. Automatsko resetiranje

Koristeći varijable m*_new i m*diff definirane kao u poglavlju 3.1. automatsko resetiranje implementirano je na sljedeći način:

Ako je signal m* unutar granica saturacije, tj. m*diff jednaka je nuli, ulazimo u fukciju definiranu pod (1). Odnosno izvršava se normalna funkcija regulatora.

Kad postoji razlika među varijablama m* i m*_new sustav se nalazi u zasićenju te po principu automaskog resetiranja prestajemo dodavati integralni dio u regulator. (Slika 3.3.)

```
    //PI controller + pojaavanje reference
    //(1)
    if(m1diff == 0 || m2diff == 0 || m3diff == 0 || m4diff == 0) {
      fm1 =-s1*FFGAIN+(v1-s1)*PGAIN+e1_integral*IGAIN;
      fm2 =-s2*FFGAIN+(v2-s2)*PGAIN+e2_integral*IGAIN;
      fm3 =-s3*FFGAIN+(v3-s3)*PGAIN+e3_integral*IGAIN;
      fm4 =-s4*FFGAIN+(v4-s4)*PGAIN+e4_integral*IGAIN;
  }

    //(2)
    if(m1 != m1_new || m2 != m2_new || m3 != m3_new || m4 != m4_new){
      fm1 =-s1*FFGAIN+(v1-s1)*PGAIN;
      fm2 =-s2*FFGAIN+(v2-s2)*PGAIN;
      fm3 =-s3*FFGAIN+(v3-s3)*PGAIN;
      fm4 =-s4*FFGAIN+(v4-s4)*PGAIN;
    }
```

Slika 3.3. – funkcija PI regulatora s automatskim restiranjem

4. Zaključak

Efekt zaleta moguće je ukloniti koristeći razne metode, od kojih su se razmatre tri: uvjetna integracija, povratno integriranje i automatsko resetiranje.

Analizom odziva shema filtara u Simulinku najboljom se pokazala metoda automatskog resetiranja.

Na temelju Simulink shema i teorijskog principa pojedine metode, u C programskom jeziku, implementirane su metode povratnog integriranja i automatskog resetiranja.