PI-Regler mit Anti-Windup für eine feldorientierte Motorregelung für ein Elektrofahrzeug

Bachelorthesis - Abschlussprojekt

Autor:	Luis Alfonso Puertollano Ventura
Studienbereich:	Elektro- und Informationstechnik
Matrikelnummer:	23 50 38 2
Erstprüfer:	Prof. Dr.- Ing. Armin Dietz
Zweitprüfer:	Prof. Dr.- Ing. Norbert Graß
Datum:	23/08/2012

Einleitung

Die Arbeit wurde in Rahmen des europeischen StudentenAustauschprogramms ERASMUS am Institut ELSYS durchgeführt.

Es geht um die Programmierung und Steuerung des Motors eines bereits vorhandenen Elektrofahrzeugs (E-Buggy). Für das Fahrzeug braucht man verschiedene Komponenten, wie einen Umrichter der von einem Mikrocontroller gesteuert und geregelt wird, andere Leistungselektronik Elemente, sowie ein kleines Batteriemanagement.

Inhaltsverzeichnis

A. Abkürzungsverzeichnis

µC	Mikrocontroller
FOC	Feldorientierte Regelung (engl. Field Oriented Control)
PWM	Pulsenweitenmodulation
PMSM	Permanenterregte Synchronmaschine (engl. Permanent Magnet Synchronous Motors)

1. Zusammenfassung

Das Projekt E-Buggy hat zum Ziel der Bau eines Elektrofahrzeugs (E-Buggy). Dafür wurden verschiedene Teams gebildet, welche sich verschiedenen Teilprojekten gewidmet haben. Dazu gehört das Design des Fahrzeuges, die Untersuchung der Batterie und das Programmieren der Leistungelektronik. Ich persönlich habe mich besonders mit der Motorregelung beschäftigt.

Für die Steuerung der PMSM, wird die feldorientierte Regelung (Field Oriented Control, FOC) genutzt (Abb. 1). Dafür war es zunächst nötig zu lernen, wie jedes Teilaufgabe funktioniert. Dazu gehört zum Beispiel: die Anwendung der Park- und Clark-Transformationen, die Funktion des Umrichters, das Programmieren eines PI-Reglers in C für den Infineon µC XC2287M.

Für diesen Teil der Arbeit, hat man sich in der PI-Regler beschäftigt. So kann man mit dem Elektromotor und seinem Drehmoment arbeiten (Abb. 1), ohne die maximale Grenze des Motorstromes zu überschreiten. Um das ganzen Regelungsystem gut funktionieren zu machen, muss man auch anpassende Reglerparameter hingefügt werden.

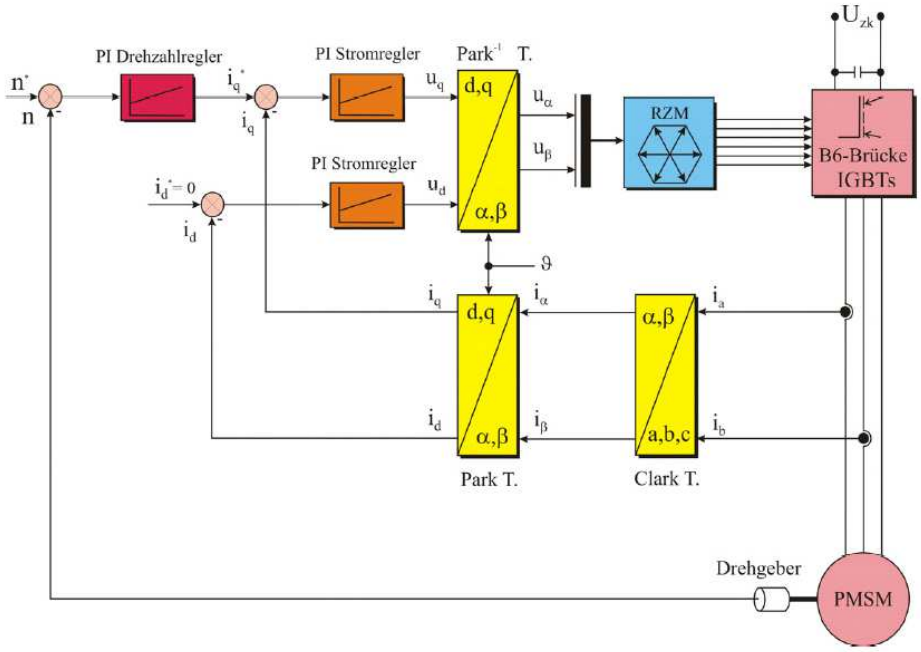


Abbildung 1: Struktur der Feldorientierten Motorregelung [Sahhary, 2008]

Folgende Programme wurden eingesetz:

- DAvE – Auto code generation.
- TASKING VX-toolset for C166 – C/C++ compiler and debugger.
- DAS Control Server – Device Acces Server.
- UDE Desktop 3.0 – Universal Debug Engine, Develop and test software applications.

2. Vergleich der Reglertypen

In der Abbildung 2 (Abb. 2) ist der Vergleich von P-, I-, PI-, PD- und PID-Regler in einem Regelkreis mit PT2-Glied als Regelstrecke dargestellt. Es ist deutlich zu sehen, dass die Regler ohne I-Anteil (P und PD) eine bleibende Regelabweichung aufweisen. Erst die Regler mit I-Anteil können auf den normierter Endwert von 1 ausregeln. Beim reinen I-Regler geht das so langsam, dass es gar nicht mehr auf dem Diagramm zu sehen ist. Der Hauptzweck eines I-Anteils ist also die Vermeidung bleibender Regelabweichungen. Daher ist ein I-Anteil normalerweise nicht nötig, wenn die Strecke schon einen I-Anteil besitzt (ausnahme: es wird ein doppelter I-Anteil zur Vermeidung von Schleppfehlern benötigt).

Die schnellsten Regler sind die mit einem D-Anteil (PD- und PID-Regler). Der D-Anteil kommt deshalb hauptsächlich zum Einsatz, wenn schnelle Dynamik gefragt ist oder die Strecke selbst schon instabil ist. Voraussetzung für die Schnelligkeit ist allerdings, dass keine Begrenzung im Stellglied oder Aktuator auftritt. In der Praxis ist eine Begrenzung meistens nicht zu vermeiden, deshalb gilt die Sprungantwort in der Praxis nur für kleine Sprünge.

Die Regler ohne D-Anteil, aber mit P-Anteil (P- und PI-Regler) sind mittelschnell. Für einfache Regelaufgaben reicht auch oft schon ein reiner P-Regler aus, wenn die bleibende Regelabweichung vernachlässigt werden kann oder wenn die Strecke schon einen I-Anteil besitzt.

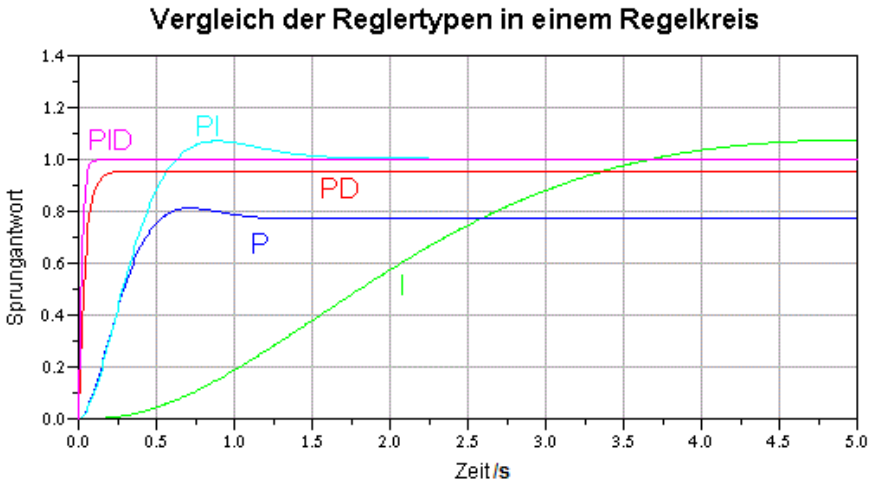


Abbildung 2: Vergleich der Antwort der Reglertypen [www.rn-wissen.de, Regelungstechnik]

3. PI-Regler mit Anti-Windup

Der PI-Regler (Proportional–Integral Regler) besteht aus den Anteilen des P-Gliedes K_P und I-Gliedes mit der Zeitkonstante T_N. Er kann sowohl aus einer Parallelstruktur oder aus einer Reihenstruktur definiert werden. Der Begriff der Nachstellzeit T_N stammt aus der Parallelstruktur des Reglers (Abb. 3).

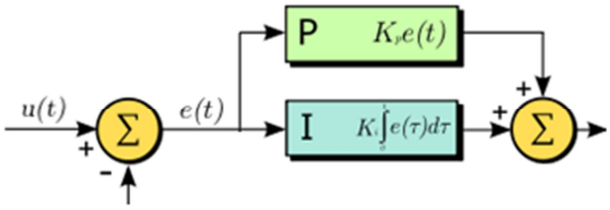


Abbildung 3: Struktur des PI-Reglers [Udenar]

So dass diese Ideale Sprungsantwort sehen kann (Abb. 4):

$$u(t) = Kp * e(t) + Ki \int_0^t e(\tau)\, d\tau \qquad (3.1.)$$

K_p → Proportionale Verstärkung
K_i → Integral Verstärkung

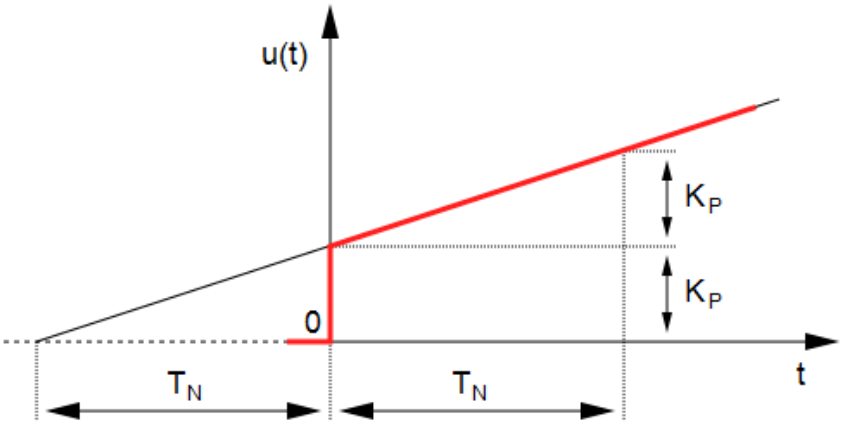


Abbildung 4: Idealer Sprungantwort des PI-Reglers [Wikipedia, PI-Regler]

Signaltechnisch wirkt der PI-Regler gegenüber dem I-Regler so, dass nach einem Eingangssprung dessen Wirkung um die Nachstellzeit T_N vorverlegt ist. Durch den I-Anteil wird die stationäre Genauigkeit gewährleistet, die Regelabweichung wird nach dem Einschwingen der Regelgröße zu Null.

3.1. Eigenschaften des PI-Reglers

- Kompensation eines PT1-Gliedes der Strecke: Er kann mit dem PD-Glied ein PT1-Glied der Strecke kompensieren und damit den offenen Regelkreis vereinfachen.
- Keine Regelabweichung bei konstantem Sollwert: Durch das I-Glied wird im stationären Zustand bei konstantem Sollwert die Regelabweichung zu Null.
- Langsamer Regler: Der durch das I-Glied erworbene Vorteil der Vermeidung einer stationären Regelabweichung hat auch den Nachteil, dass eine zusätzliche Polstelle mit -90 ° Phasenwinkel in den offenen Regelkreis eingefügt wird, was eine Reduzierung der Kreisverstärkung K_{PI} bedeutet. Deshalb ist der PI-Regler kein schneller Regler.
- 2 Einstellparameter: Der Regler enthält nur zwei Einstellparameter, $K_{PI} = K_P / T_N$ und T_N.
- Regelstrecke höherer Ordnung: kann optimal an einer Regelstrecke höherer Ordnung eingesetzt werden, von der nur die Sprungantwort bekannt ist. Durch Ermittlung der Ersatztotzeit T_U (Verzugszeit) und der Ersatzverzögerungs-Zeitkonstante T_G (Ausgleichszeit) kann das PD-Glied des Reglers die Zeitkonstante T_G kompensieren. Für die I-Regler-Einstellung der verbleibenden Regelstrecke mit Ersatztotzeit TU gelten die bekannten Einstellvorschriften.
- Regelstrecke mit 2 dominanten Zeitkonstanten: er kann eine Regelstrecke mit zwei dominanten Zeitkonstanten von PT1-Gliedern regeln, wenn die Kreisverstärkung reduziert wird und die längere Dauer des Einschwingens der Regelgröße auf den Sollwert akzeptiert wird. Dabei kann mit K_{PI} jeder gewünschte Dämpfungsgrad D eingestellt werden, von aperiodisch (D=1) bis schwach gedämpft schwingend (D gegen 0).
- PD-Glied ohne Differenzierung: Das in der Reihenstruktur entstandene PD-Glied des PI-Reglers ist mathematisch ohne Differenzierung entstanden. Deshalb entsteht bei der Realisierung des Reglers in der Parallelstruktur auch keine parasitäre Verzögerung. Wegen eines möglichen Windup-Effektes durch Regelstreckenbegrenzung der Stellgröße u(t) ist die schaltungsmäßige Realisierung des PI-Reglers in Parallelstruktur anzustreben.

Die Grundidee des PI-Reglers (Abb. 5) ist einfach und lässt sich vergleichen mit der Entscheidungsfindung von Menschen, welche häufig auf der Kenntniss von Vergangenheit und Gegenwart basiert. Der PI-Regler macht das Gleiche mit dem Regler. Das Steuersignal besteht aus zwei Teilen:

- Ein proportionaler Teil zum Auffinden des Fehlers (Regelverhalten aufgrund der gegenwärtigen Information).
- Ein proportionaler Teil zum Integral-Fehler der Vergangenheit (Regelverhalten aufgrund der vergangenen Information).

So dass:

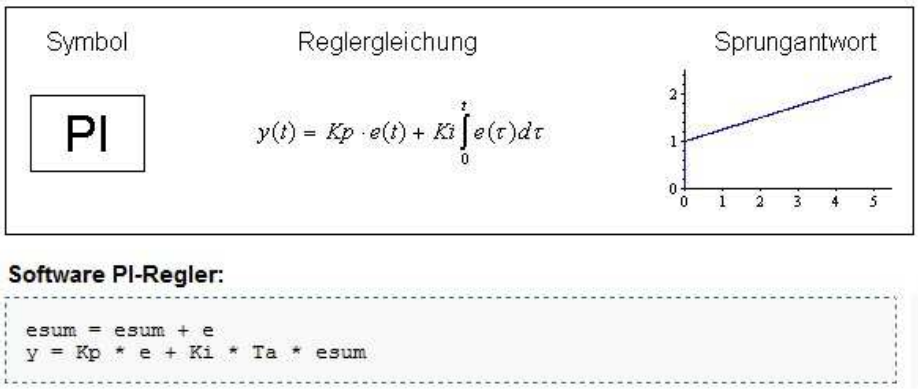


Abbildung 5: PI-Regler [www.rn-wissen.de, Regelungstechnik]

Übertragungsfunktion der Parallelstruktur. Solche Werten werden so normalerweise berechnet:

$$\frac{U(S)}{E(S)} = K_p \frac{1 + T_N * S}{T_N * S}$$

$$K_{PI} = \frac{K_p}{T_N}$$

TN und KP sind berechnet so dass es keine Oberschingungen erscheinen. Wenn man es falsch macht, kann folgendes passieren (Abb. 6):

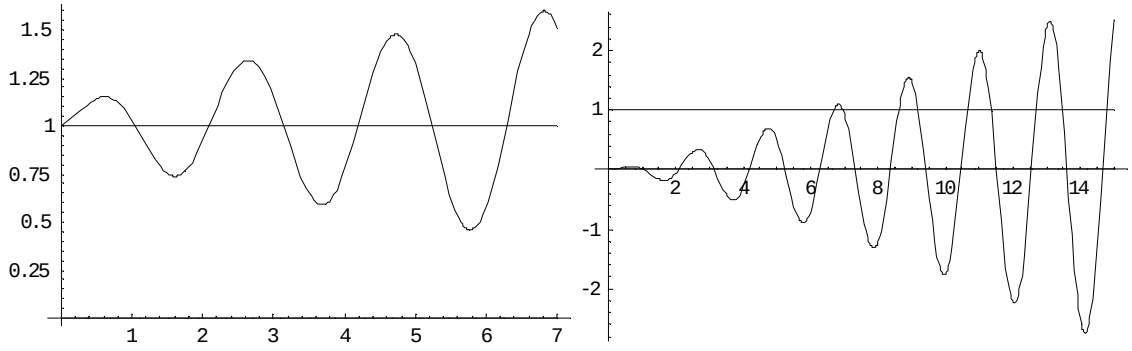


Abbildung 6: Sprungsnatworte mit Oberschwingungen [Luis Puertollano, 2012]

Deswegen, werden die kritische Werte berechnet, um an der Grenze den Oberschwingungen zu bleiben, und werden danach wie folgendes benutzt:

$$K_p = 0.45 * K_{PKrit}$$

$$T_N = 0.83 * T_{NKrit}$$

Obwohl viele Aspekte eines Steuersystems von der linearen Regelungstechnik verstanden werden können, müssen einige nicht-lineare Effekte berücksichtigt werden. Bei der Implementierung eines Reglers sind dies zum Beispiel:

- Ein motor hat eine nominale Geschwindigkeit (wie in unserem Fall).
- Ein Ventil hat eine maximale und eine minimale Öffnung.
- Eine Stromversorgung eines elektrischen Gerätes ist begrenzt.

3.2. Anti-Windup Funktion

Für eine Steuerung mit einer Vielzahl von Betriebsbedingungen kann es vorkommen, dass die Regelgröße die vorgegebenen Grenzen des Aktuators erreicht. Wenn dies passiert, bleibt die Rückkopplungsschleife in ihrer Grenze unabhängig von dem Prozeß-Ausgang. Bei Verwendung eines integralen Reglers wird ein möglicher Fehler immer weiter seinen Wert steigern. Der Integral-Anteil wird ganz groß und es wird die Wirkung "Windup" produziert (Abb. 7).

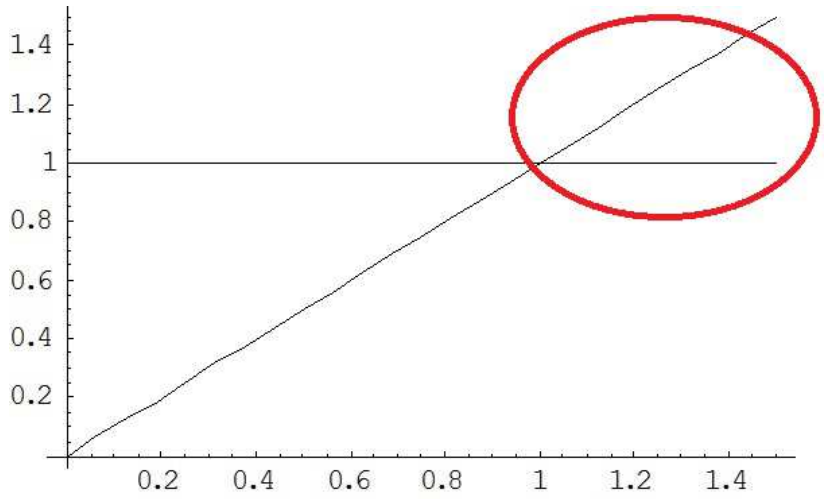


Abbildung 7: Beispiel von Windup [Luis Puertollano, 2012]

Dies kann so vermieden werden:

- Der Begrenzung der Reglerabweichung bewirkt, dass der Ausgang des Reglers nicht die Begrenzung des Aktuators erreichen kann. Dies produziert häufig Begrenzungen in der Wirkung des Reglers, aber kann nichts gegen den Windup-Effekt, welcher von Störungen produziert wird, ausrichten.
- Ein anderer Weg ist die Neuberechnung des Integralanteils: Wenn der Ausgang gesättigt ist, wird das Integral neu berechnet, so dass der neue Wert ein Ausgangssignal an der Sättigungsgrenze darstellt.

Eine andere Methode ist die konditionale Integration zur Begrenzung des Integralüberlaufs. Man benutzt ein Schalter, wenn das Signal sehr weit weg von dem stationären Status liegt. In diesem Fall, wird der I-Anteil nur unter solchen Bedingungen benutzt. In anderen Fällen, bleibt der I-Anteil gleich. Es wird auch „Festbinden des Integrators“ genannt.

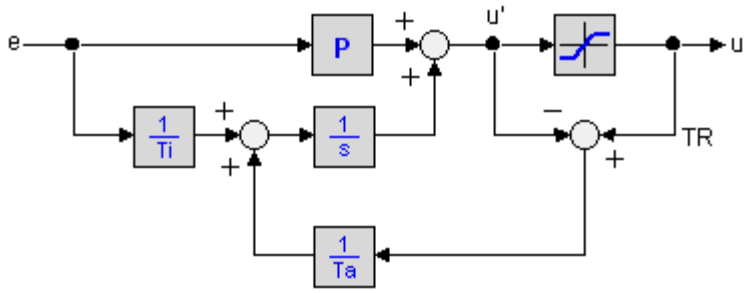


Abbildung 8: PI-Regler Parallelform und Begrenzung [www.20sim.com]

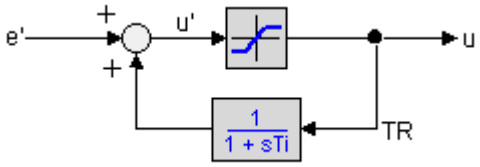


Abbildung 9: PI-Regler Seriesform und Begrenzung [www.20sim.com]

Der Unterschied zwischen Eingang und Ausgang (TR) ist rückgekoppelt in den I-Anteil durch den $1/T_a$ Verstärkung (Abbs. 8 und 9). Eine gute Überwachung ist die Rückkopplung mit Anti-Windup. Der Aktuator ist von einem Signal-Begrenzer vertreten worden.

Sobald der Begrenzer sättigt, bekommt das Signal einen unterschiedlichen Wert von null aufwärts und vermeidet, dass der I-Anteil in Windup geht (Abb. 10). Reset-Zeitkonstanten bestimmen dabei, wie schnell auf Windup des Integrators reagiert wird.

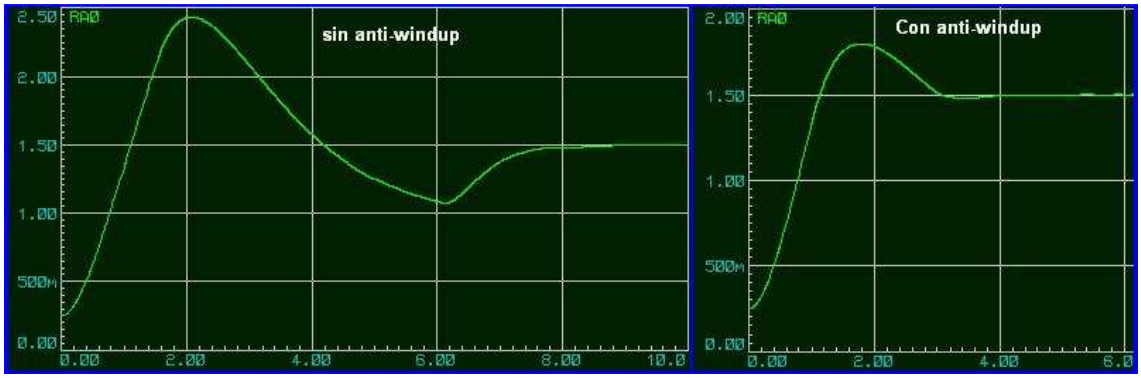


Abbildung 10: Antwort ohne und mit Anti-Windup [micros-designs, Suky, CreativeCommons]

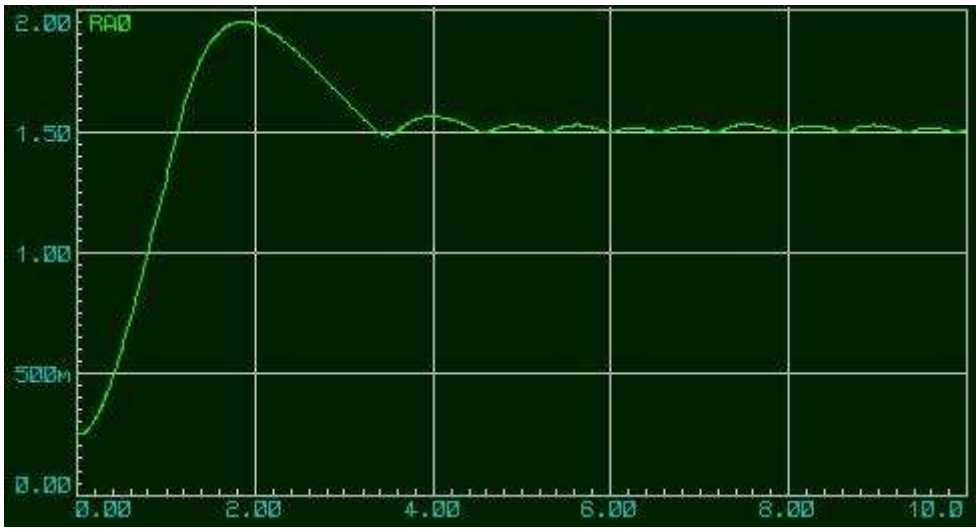


Abbildung 11: Antwort mit Switch On-Off Methode [micros-designs, Suky, CreativeCommons]

Der Ziel des Anti-Windups ist der Integration zu begrenzen (Abb. 12).

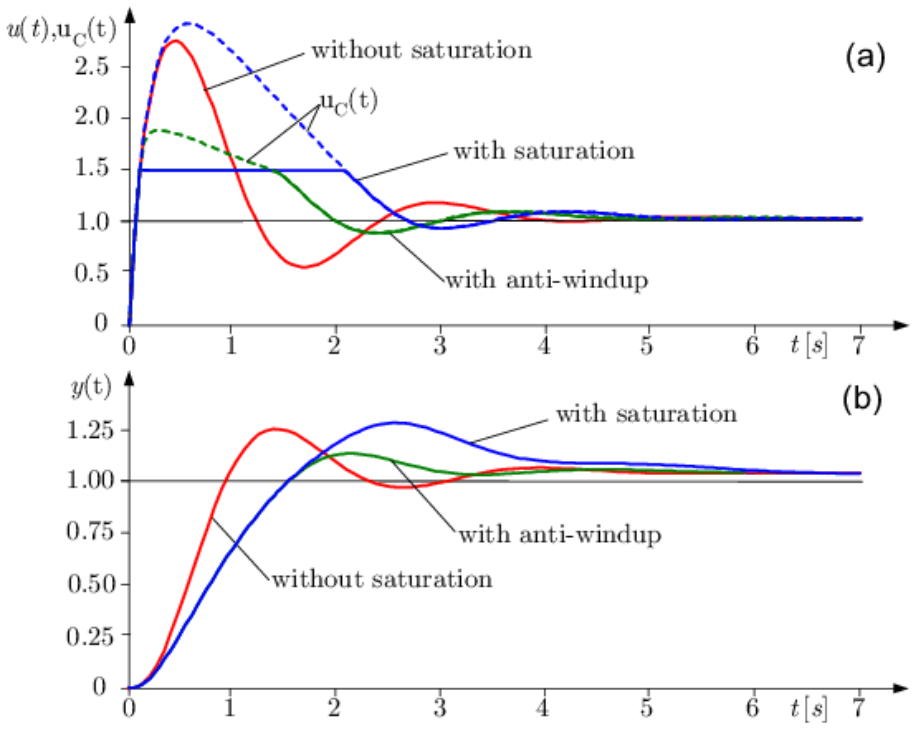


Abbildung 12: Antwort des Systems mit und ohne Anti-Windup
[Universität Bochum, Christian Schmid, 2005]

Diese PI-Regler mit Anti-Windup müssen angewendet werden, so dass ihre Funktion von einem µC ausgeübt werden kann. Der Aktuator wird begrenzt und gibt ein Signal (Abb. 13), welches auf der folgenden Grundidee begrenzt wird:

$$\boldsymbol{output = minimum};\ (\boldsymbol{input < minimum})$$
$$\boldsymbol{output = input};\ (\boldsymbol{minimum <= input <= maximum})$$
$$output = maximum;\ (input > maximum)$$

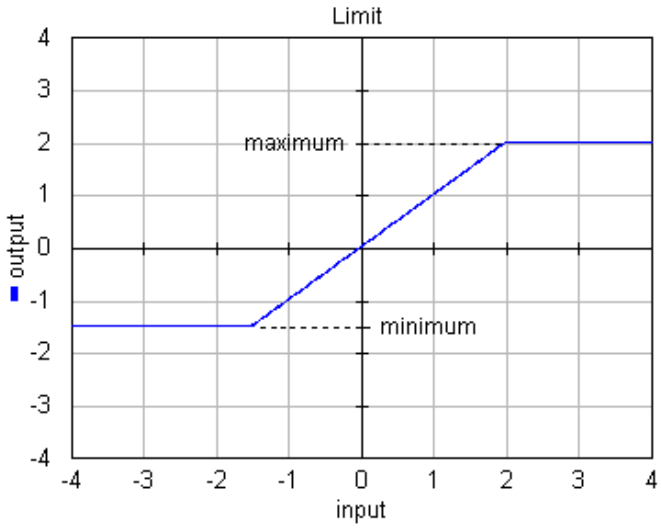


Abbildung 13: Grundfunktionsweise des Reglers mit Anti-Windup stationares Vorhalten [http://www.20sim.com]

Ein Beispiel des Grundprogrammes wäre folgendes:

Programm-Code in C:

$e = w - x$; // Vergleich
$esum = esum + e$; // Integration I-Anteil
if $(esum < -400)\{esum = -400;\}$ // Begrenzung I-Anteil
if $(esum < 400)\{esum = 400;\}$
$y = \mathrm{Kp} * \mathrm{e} + \mathrm{Ki} * \mathrm{Ta} *$ esum;//Reglergleichung
if $(y < 0)\{y = 0;\}$ // Begrenzung Stellgröße
if $(y < 1023)\{y = 1023;\}$
$PWM = y$; // Übergabe Stellgröße

4. Realisierung des PI-Reglers

Für die Codeentwicklung wird die Software „TASKING VX-toolset for C166“ eingesetzt (Abb. 14).

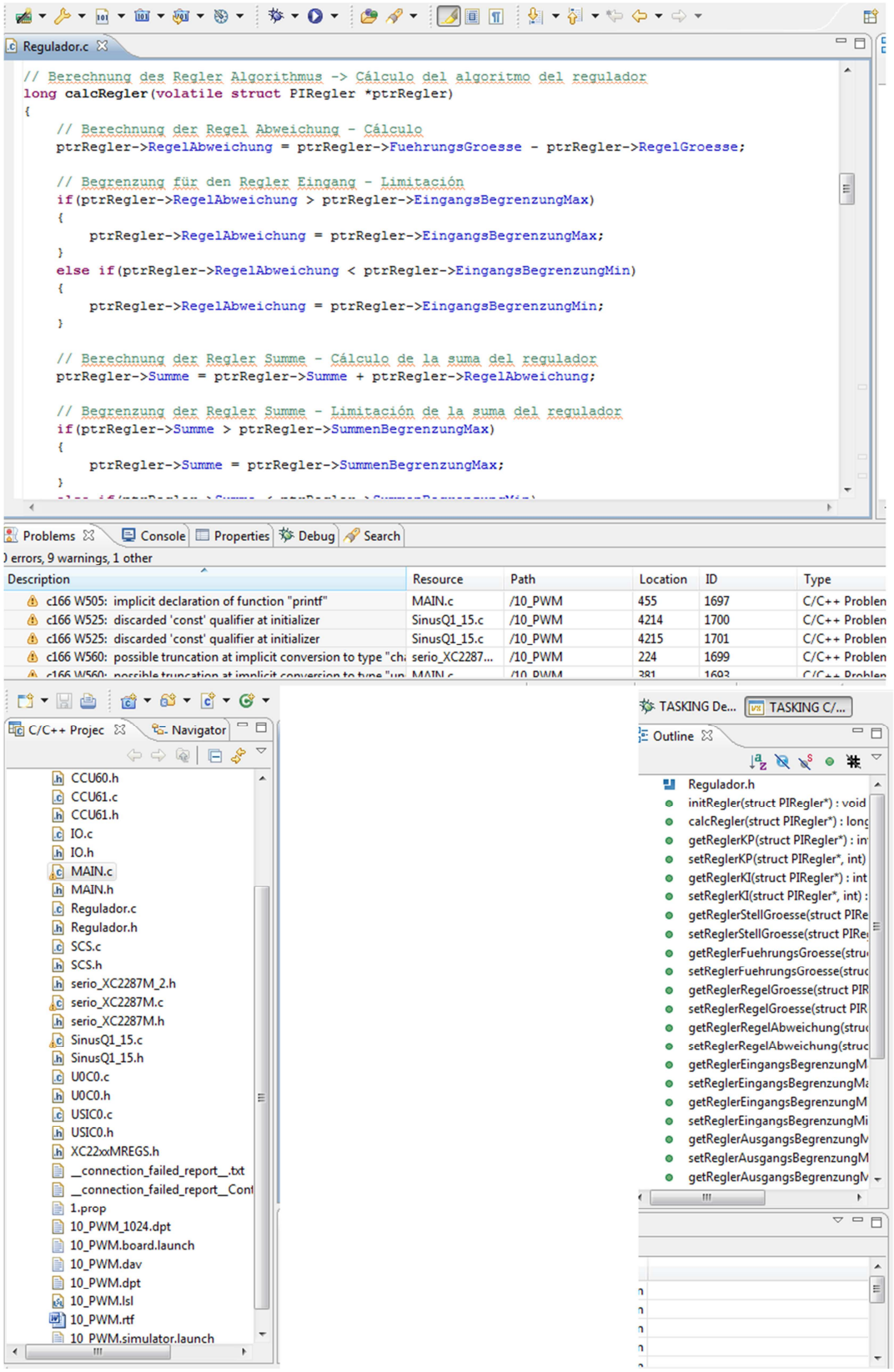


Abbildung 14: Ansicht von TASKING VX-toolset for C166 [Luis Puertollano, 2012]

4.1. PI-Regler mit Anti-Windup-Funktionalität

4.1.1. Initialisierung

Hier wird der Regler initialisiert, und wird folgende Werte für die verschiedenen Glieden benutzen:

```
// Initialisierung des Reglers – Werten die man für die verschiedenen Glieden will.

void initRegler(volatile struct PIRegler *ptrRegler)

{

        ptrRegler->KP                   =       0;// Proportionalkonstant - Constante proporcional

        ptrRegler->KI                   =       0;// Integrationskonstant - Constante de integración


        ptrRegler->StellGroesse         =       0;// Stellgrösse - Control de la variable

        ptrRegler->FuehrungsGroesse     =       0;// Führungsgrösse - Comando de la variable

        ptrRegler->RegelGroesse         =       0;// Regelgrösse - Control de la variable


        ptrRegler->RegelAbweichung      =       0;// Abweichung (Unterschied) - Desviación


        ptrRegler->EingangsBegrenzungMax =      0;// Maximal Eingangs Begrenzung -
                                                        Limitación máxima de entrada

        ptrRegler->EingangsBegrenzungMin =      0;// Minimal Eingangs Begrenzung -
                                                        Limitación mínima de entrada

        ptrRegler->AusgangsBegrenzungMax =      0;// Maximal Ausgangs Begrenzung -
                                                        Limitación máxima de salida

        ptrRegler->AusgangsBegrenzungMin =      0;// Minimal Ausgangs Begrenzung -
                                                        Limitación mínima de salida


        ptrRegler->Abtastzeit           =       0;// Abtastzeit - Tiempo de muestreo


        ptrRegler->Summe                =       0;// Summe - Suma

        ptrRegler->SummenBegrenzungMax  =       0;// Maximal Summe Begrenzung -
                                                        Limitación máxima de la suma

        ptrRegler->SummenBegrenzungMin  =       0;// Minimal Summe Begrenzung -
                                                        Limitación mínima de la suma

}
```

4.1.2. Berechnung und Anti-Windup-Algorithmus

Hier wird es beschrieben wie man die Berechnung des PI-Reglers programmiert ist:

```
// Berechnung des Regler Algorithmus -> Cálculo del algoritmo del regulador

long calcRegler(volatile struct PIRegler *ptrRegler)

{

	// Berechnung der Regel Abweichung - Cálculo

	ptrRegler->RegelAbweichung = ptrRegler->FuehrungsGroesse - ptrRegler->RegelGroesse;


	// Begrenzung für den Regler Eingang - Limitación

	if(ptrRegler->RegelAbweichung > ptrRegler->EingangsBegrenzungMax)

	{

		ptrRegler->RegelAbweichung = ptrRegler->EingangsBegrenzungMax;

	}

	else if(ptrRegler->RegelAbweichung < ptrRegler->EingangsBegrenzungMin)

	{

		ptrRegler->RegelAbweichung = ptrRegler->EingangsBegrenzungMin;

	}


	// Berechnung der Regler Summe - Cálculo de la suma del regulador

	ptrRegler->Summe = ptrRegler->Summe + ptrRegler->RegelAbweichung;


	// Begrenzung der Regler Summe - Limitación de la suma del regulador

	if(ptrRegler->Summe > ptrRegler->SummenBegrenzungMax)

	{

		ptrRegler->Summe = ptrRegler->SummenBegrenzungMax;

	}

	else if(ptrRegler->Summe < ptrRegler->SummenBegrenzungMin)

	{

		ptrRegler->Summe = ptrRegler->SummenBegrenzungMin;

	}
```

```
	// Berechnung der Stellgröße - Cálculo de la variable de control
	ptrRegler->StellGroesse = ptrRegler->KP * ptrRegler->RegelAbweichung +
					ptrRegler->KI * ptrRegler->Abtastzeit * ptrRegler->Summe;

	// Begrenzung für den Regler Ausgang - Limitación de la salida del regulador
	if(ptrRegler->StellGroesse > ptrRegler->AusgangsBegrenzungMax)
	{
		ptrRegler->StellGroesse = ptrRegler->AusgangsBegrenzungMax;
	}
	else if(ptrRegler->StellGroesse < ptrRegler->AusgangsBegrenzungMin)
	{
		ptrRegler->StellGroesse = ptrRegler->AusgangsBegrenzungMin;
	}

	return ptrRegler->StellGroesse;
}
/*****************************************************************/
```

4.1.3. Ergebnisse

Hier wird es beschrieben wie man die Ergebnisse des PI-Reglers bekommen kann:

```
// Get Methode für KP
int getReglerKP(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->KP;
}

// Set Methode für KP
void setReglerKP(volatile struct PIRegler *ptrRegler, int kp)
{
	ptrRegler->KP = kp;
```

```
}
/*********************************************************************/
// Get Methode für KI
int getReglerKI(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->KI;
}

// Set Methode für KI
void setReglerKI(volatile struct PIRegler *ptrRegler, int ki)
{
	ptrRegler->KP = ki;
}
/*********************************************************************/
// Get Methode für StellGroesse
long getReglerStellGroesse(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->StellGroesse;
}

// Set Methode für StellGroesse
void setReglerStellGroesse(volatile struct PIRegler *ptrRegler, long stellgroesse)
{
	ptrRegler->StellGroesse = stellgroesse;
}
/*********************************************************************/
// Get Methode für FuehrungsGroesse
long getReglerFuehrungsGroesse(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->FuehrungsGroesse;
```

```
}

// Set Methode für FuehrungsGroesse
void setReglerFuehrungsGroesse(volatile struct PIRegler *ptrRegler, long fuehrungsgroesse)
{
	ptrRegler->FuehrungsGroesse = fuehrungsgroesse;
}
/*****************************************************************/
// Get Methode für RegelGroesse
long getReglerRegelGroesse(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->RegelGroesse;
}

// Set Methode für RegelGroesse
void setReglerRegelGroesse(volatile struct PIRegler *ptrRegler, long regelgroesse)
{
	ptrRegler->RegelGroesse = regelgroesse;
}
/*****************************************************************/
// Get Methode für RegelAbweichung
long getReglerRegelAbweichung(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->RegelAbweichung;
}

// Set Methode für RegelAbweichung
void setReglerRegelAbweichung(volatile struct PIRegler *ptrRegler, long regelabweichung)
{
	ptrRegler->RegelAbweichung = regelabweichung;
```

```
}
/*********************************************************************/
// Get Methode für EingangsBegrenzungMax
long getReglerEingangsBegrenzungMax(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->EingangsBegrenzungMax;
}

// Set Methode für EingangsBegrenzungMax
void setReglerEingangsBegrenzungMax(volatile struct PIRegler *ptrRegler, long eingangsBegrenzungMax)
{
	ptrRegler->EingangsBegrenzungMax = eingangsBegrenzungMax;
}
/*********************************************************************/
// Get Methode für EingangsBegrenzungMin
long getReglerEingangsBegrenzungMin(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->EingangsBegrenzungMin;
}

// Set Methode für EingangsBegrenzungMin
void setReglerEingangsBegrenzungMin(volatile struct PIRegler *ptrRegler, long eingangsBegrenzungMin)
{
	ptrRegler->EingangsBegrenzungMin = eingangsBegrenzungMin;
}
/*********************************************************************/
// Get Methode für AusgangsBegrenzungMax
long getReglerAusgangsBegrenzungMax(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->AusgangsBegrenzungMax;
```

```
}

// Set Methode für AusgangsBegrenzungMax
void setReglerAusgangsBegrenzungMax(volatile struct PIRegler *ptrRegler, long ausgangsBegrenzungMax)
{
	ptrRegler->AusgangsBegrenzungMax = ausgangsBegrenzungMax;
}
/*********************************************************************/
// Get Methode für AusgangsBegrenzungMin
long getReglerAusgangsBegrenzungMin(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->AusgangsBegrenzungMin;
}

// Set Methode für AusgangsBegrenzungMin
void setReglerAusgangsBegrenzungMin(volatile struct PIRegler *ptrRegler, long ausgangsBegrenzungMin)
{
	ptrRegler->AusgangsBegrenzungMin = ausgangsBegrenzungMin;
}
/*********************************************************************/
// Get Methode für Abtastzeit
int getReglerAbtastzeit(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->Abtastzeit;
}

// Set Methode für Abtastzeit
void setReglerAbtastzeit(volatile struct PIRegler *ptrRegler, int abtastzeit)
{
	ptrRegler->Abtastzeit = abtastzeit;
```

```
}
/*********************************************************************/
// Get Methode für SummenBegrenzungMax
long getReglerSummenBegrenzungMax(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->SummenBegrenzungMax;
}

// Set Methode für SummenBegrenzungMax
void setReglerSummenBegrenzungMax(volatile struct PIRegler *ptrRegler, long summenBegrenzungMax)
{
	ptrRegler->SummenBegrenzungMax = summenBegrenzungMax;
}
/*********************************************************************/
// Get Methode für SummenBegrenzungMin
long getReglerSummenBegrenzungMin(volatile struct PIRegler *ptrRegler)
{
	return ptrRegler->SummenBegrenzungMin;
}

// Set Methode für SummenBegrenzungMin
void setReglerSummenBegrenzungMin(volatile struct PIRegler *ptrRegler, long summenBegrenzungMin)
{
	ptrRegler->SummenBegrenzungMin = summenBegrenzungMin;
}

/*********************************************************************/
```

4.1.4. Einstellungen des Reglers

Hier wird es geschrieben wie man die Parametereinstellungen des PI-Reglers:

```
// Reglereinstellungen setzen für Id-Regler - Establecer los parámetros del Regulador (Potencia reactiva)


void mainSetPIReglerIdParams(struct PIRegler *ptrPIReglerId)

{

        setReglerKP(ptrPIReglerId, 0);

        setReglerKI(ptrPIReglerId, 0);

        setReglerEingangsBegrenzungMax(ptrPIReglerId, 350);

        setReglerEingangsBegrenzungMin(ptrPIReglerId, -350);

        setReglerAusgangsBegrenzungMax(ptrPIReglerId, 52213);

        setReglerAusgangsBegrenzungMin(ptrPIReglerId, -52213);

        setReglerAbtastzeit(ptrPIReglerId, 1);

        setReglerSummenBegrenzungMax(ptrPIReglerId, 100000);

        setReglerSummenBegrenzungMin(ptrPIReglerId, -100000);

}


// Reglereinstellungen setzen für Iq-Regler  - Establecer los parámetros del Regulador (Potencia activa)

void mainSetPIReglerIqParams(struct PIRegler *ptrPIReglerIq)

{

        setReglerKP(ptrPIReglerIq, 0);

        setReglerKI(ptrPIReglerIq, 0);

        setReglerEingangsBegrenzungMax(ptrPIReglerIq, 350);

        setReglerEingangsBegrenzungMin(ptrPIReglerIq, -350);

        setReglerAusgangsBegrenzungMax(ptrPIReglerIq, 52213);

        setReglerAusgangsBegrenzungMin(ptrPIReglerIq, -52213);

        setReglerAbtastzeit(ptrPIReglerIq, 1);

        setReglerSummenBegrenzungMax(ptrPIReglerIq, 100000);

        setReglerSummenBegrenzungMin(ptrPIReglerIq, -100000);

}
```

Mit diesem Code wird der Regler definiert, d.h. die Ausgangsbegrenzungen und die Parameter des Reglers. Und man muss auch den Regler initialisieren. Damit wird folgender Code in der Main.c File geschrieben:

4.2. Hauptprogramm

Hier wird es geschrieben, wie man die anderen Teilen des Programms anrufen muss:

```
// Initialisierung des Reglers

void initRegler(volatile struct PIRegler *ptrReglerId);

void initRegler(volatile struct PIRegler *ptrReglerIq);

void mainSetPIReglerIdParams(struct PIRegler *ptrPIReglerId);

void mainSetPIReglerIqParams(struct PIRegler *ptrPIReglerIq);
```

B. Literaturverzeichnis

[1] Laboratorio de procesos industriales:
http://www.elai.upm.es:8009/spain/Asignaturas/ControlProcesos/archivos/Practicas/Practica_1.pdf

[2] Regelungstechnik:
http://www.rn-wissen.de/index.php/Regelungstechnik

[3] Anti-Windup:
http://www.20sim.com/webhelp/library/signal/control/pid_control/antiwindup.htm

[4] Control PID con Anti-Windup:
http://www.micros-designs.com.ar/control-pid-con-anti-windup-en-pic/

[5] Regelungstechnik:
http://de.wikipedia.org/wiki/Regler#PI-Regler

[6] Infineon: [AP16084] FOC of a PMSM Application Note, V1.0, May 2004

[7] Sahhary, Bassel; Elektrische Antreibe mit dauermagneterregten Maschinen im dynamischen sensorlosen Betrieb. Hamburg, 2008.

[8] Nickl, Felix; Feoldorientierte Motorregelung für ein Elektrofahrzeug; Nürnberg, 2011.

[9] Control system with anti-windup measure:
http://www.atp.ruhr-uni-bochum.de/rt1/syscontrol/node92.html

[10] Universidad de Zaragoza;
Regelungstechnik-, Leistungs-, und Digitalelektronikunterlagen.

[11] [Sahhary 2008] Sahhary, Bassel: Elektrische Antriebe mit dauermagneterregten Maschinen im dynamischen sensorlosen Betrieb, Helmut-Schmidt-Universität Hamburg, Diss., 2008. http://opus.unibw-hamburg.de/opus/volltexte/2009/1904/pdf/2009_Sahhary.pdf, Abruf: 01.02.2011

[12]Datenblätter:
http://www.infineon.com/dgdl/XC2287M-PB.pdf?folderId=db3a3043132679fb01133eb909a307c3&fileId=db3a30431c69a49d011c94d56aa7058d

[13] [Udenar] Universidad de Colombia, Kolumbien Universität

C. Anhang

C.1. Mikrocontrollers Infineon XC2287M

XC22xxM-Series

⊞ Alles ausklappen

Unterkategorie auswählen

⊞ XC2000 Development Tools, Software and Kits ⊞ Service, Support and Training

The XC223xM (LQFP-64), XC226xM (LQFP-100) and XC228xM (LQFP-144) microcontroller series is based on Infineon's popular and well-established C166 architecture. With a Flash size of up to 832 KByte and a 80MHz performance, the microcontroller is well suited for automotive body applications.

Key features:

- 80 MHz frequency = 80 MIPS performance
- Up to 832 KB of flash memory and 50 KB of RAM
- Parallel Flash Programming
- Up to 4 PWM units (CCU6) to drive 3-phase motor
- Two very fast 10-bit A/D converters
- Up to 6 serial interfaces (USIC channels)
- Up to 6 CAN
- Low PIN count package LQFP-64
- Low power consumption
- Low power modes
- DAP - Device Access Port (2 wire JTAG, replaces 5 wire JTAG)

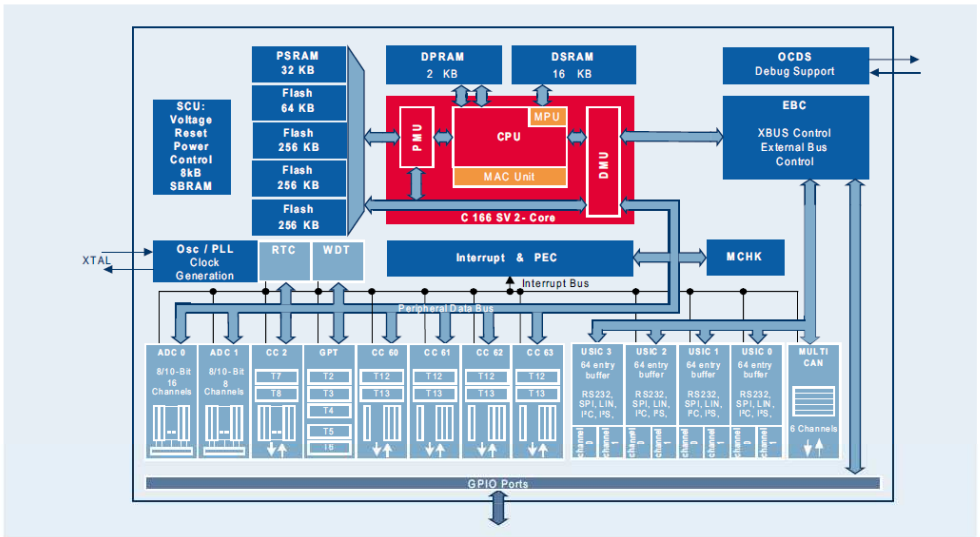
XC22xxM Series		
Subfamily		XC2287M
Core	Core	C166SV2
	Frequency (MHz)	40-80
Package		QFP 144
Flash	Program Flash (kB)	384-768
Flash	Data Flash (kB)	64
SRAM	∑ SRAM (kB)	26-50
CAN	Channels (message objects)	6 (256)
ADC	Channels	24
Universal Serial Interface [USIC] Channels*		8
Capture Compare Units [CCU]**	CCU 1	0
	CCU 2	1
	CCU 6	4
Temperatur (Tambient)		-40°C to +125°C

Abbildung 15: Dattenblatt des XC2287M Mikrocontroller [Infineon]

XC2287M

Next Generation Microcontroller with 32 - Bit Performance

Block Diagram XC2287M



Further Features

- Two synchronizable A/D converters with 24 channels, optional data pre-processing, and a conversion time down to 1.2µs
- 16-Channel general purpose capture/compare unit
- Four capture/compare units for flexible PWM signal generation (3 capture/compare channels and 1 compare channel)
- Multi-functional general purpose timer unit with 5 timers
- Eight serial interface channels to be used as UART, LIN, buffered SPI, IIC Bus Interface, IIS Interface
- On-Chip MultiCAN Interface (Rev. 2.0B active) with 256 message objects on 6 CAN nodes and gateway functionality
- On-chip real time clock
- Enhanced power saving modes with flexible power management
- Programmable watchdog timer and oscillator watchdog
- Up to 119 general purpose I/O lines
- On-chip bootstrap loader
- Supported by a large range of development tools
- On-chip debug support via JTAG interface
- 144-pin green LQFP package, 0.5 mm (19.7 mil) pitch
- Temperature range: -40° to +125°C
- Single Power Supply from 3.0V to 5.5V
- Hardware CRC-Checker with Programmable Polynomial to supervise On-Chip Memory Areas

Abbildung 16: Diagram Block XC2287M [Infineon]