# Quadrocopter Regelung

# Vorraussetzungen

Zum Verständnis der Erweiterung ist es empfiehlt es sich die Dokumentation des „Projektes Quadrocoper“ zu lesen.

# Übersichtsbild

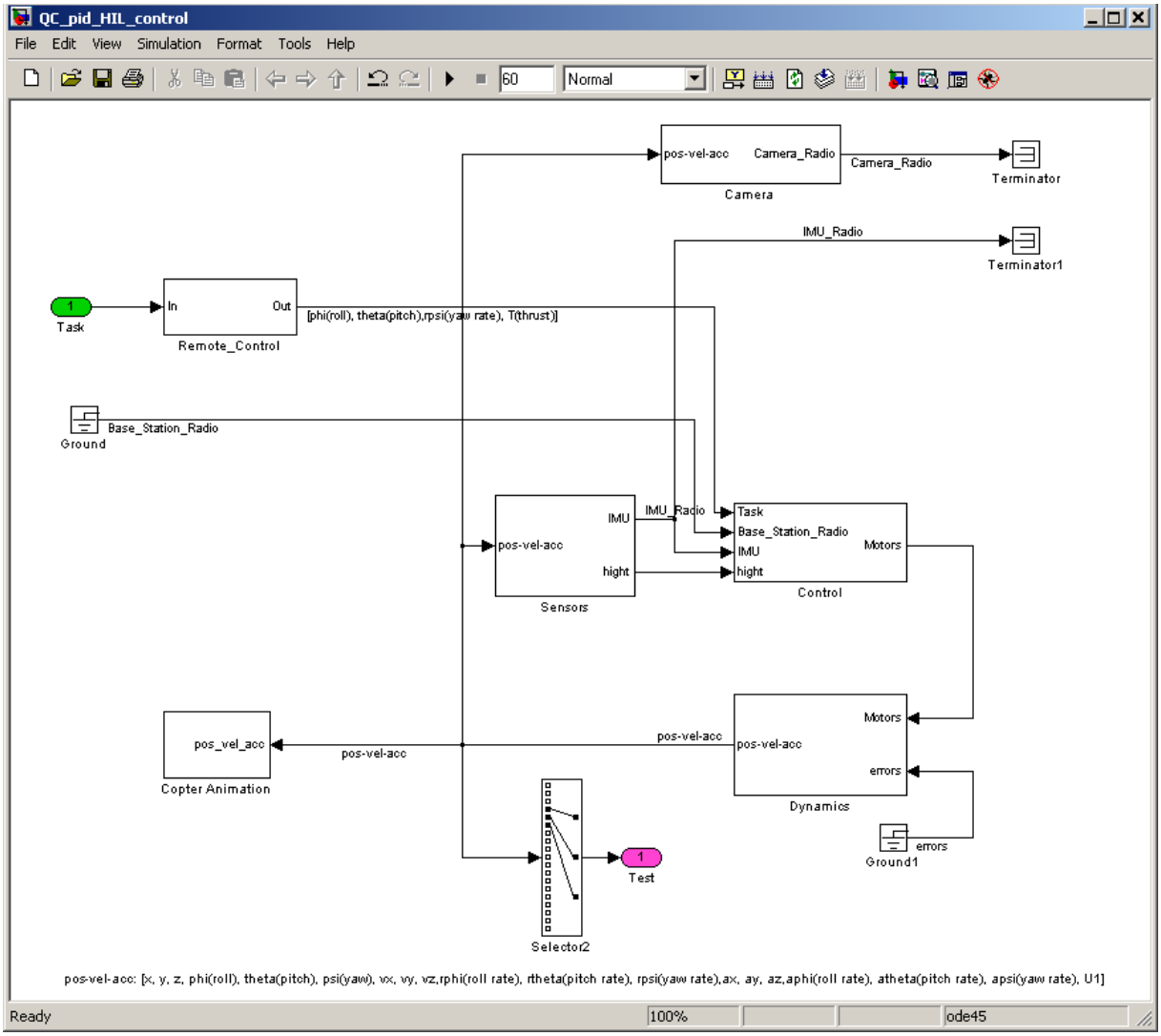


**Abbildung 1: Gesamtsystem**

## Beschreibung der Komponenten

**Camera:** Dieser Block dient einer späteren Erweiterung und ist in dieser Version nur eine Verbindung von Ein- zu Ausgang.

**Camera Radio:** Dieses Signal dient einer späteren Verwendung um ein Kamerabild zu einem Host zu übertragen.

**IMU-Radio:** Dieses Signal dient einer späteren Erweiterung um gemessene Zustände des Quadrocopters zu einem Host zu übertragen.

**Remote Control:** Dieser Block simuliert die Sollwerte von Fernbedienung. Auf diese Werte wird geregelt.

**Base Station Radio:** Dieses Signal dient einer späteren Erweiterung um Daten/Sollwerte vom Host zu erhalten und diese in die Regelung einfließen zu lassen.

**Sensors:** Dieser Block generiert aus den Zuständen des Quadrocopters die Informationen der Sensoren unter anderem durch hinzufügen von Offsets, Rauschen, etc.)

**Control:** Dieser Block beinhaltet die Regelung hier werden die Sensordaten, Sollwerte der Fernbedienung oder des Hosts ausgewertet und die Stellgrößen für die 4 Motoren berechnet und als Ausgänge zur Verfügung gestellt.

**Copter Animation:** Dieser Block beinhaltet eine S-Function für eine „D-Animation des Quadrocopters.

**Dynamics:** Dieser Block beinhaltet die Regelstrecke. Hier wird berechnet wie der Quadrocopter auf die Stellgrößen und andere Einflüsse reagiert. Somit werden hier alle Zust ände des Quadrocopter berechnet.

**Pos-vel-acc:** Dieses Signal beinhaltet alle Zustände des Quadrocopters.

**Selector2:** Dieser Block ist ein Multiplexer mit dem Zustände auf den Ausgang geschaltet werden. Wird über die Matlab-Konsole simuliert kann hier festgelegt werden welche Zustände betrachtet bzw. auf den Workspace exportiert werden sollen**.**

Um das Modell zu benutzen müssen zuerst die Parameter in den Workspace geladen werden. Dies geschieht durch ausführen des M-File „quadrocopter_param.m“. Dort werden Massen, Trägheitsmomente und Ähnliches angegeben.

# Regelstrecke

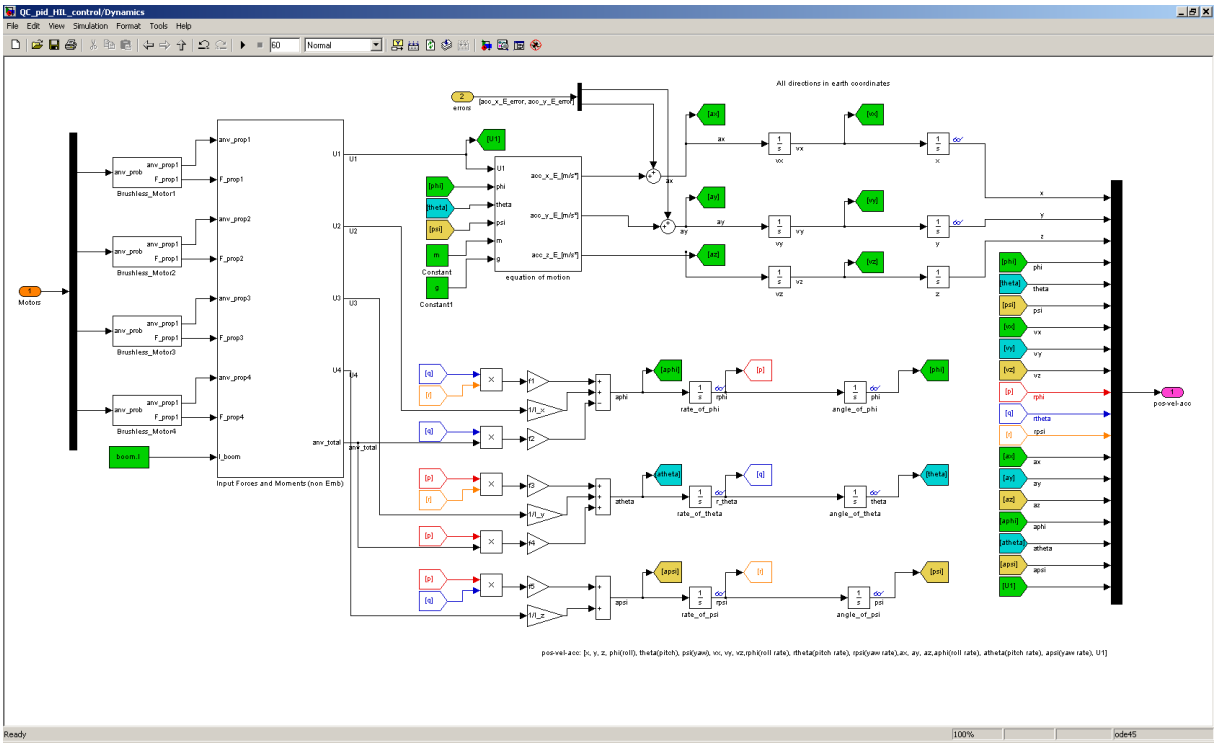


**Abbildung 2: Blockschaltbild der Regelstrecke**

Die Regelstrecke besteht aus den 4 Motoren, welche jeweils mit einer unterschiedlichen Verstärkung parametriert werden können. Diese unterschiedliche Verstärkung simuliert Asymmetrien des erzeugten Schubs auf Grund von Tolleranzen der Motoren und der Brushless-Regler, und der unterschiede auf Grund der asymmetrischen Masseverteilung.

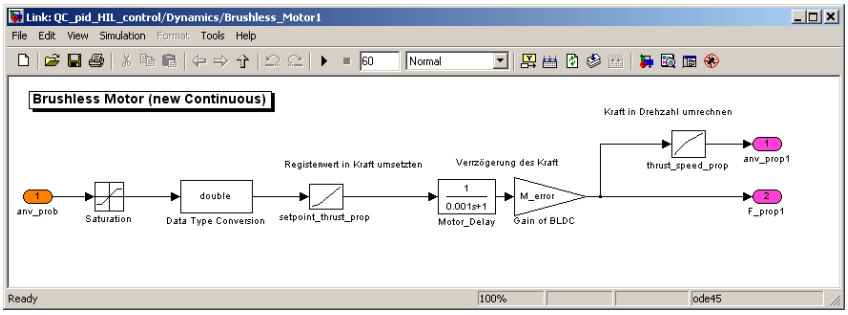


**Abbildung 3: Blockschaltbild eines Brushless-Motors**

Wie in Abbildung 3 ersichtlich, wird die Stellgröße über eine Sättigung auf den Wertebereich von 0 … 255 begrenzt. Die Datentyp Konvertierung auf Double passt den Integerwert wieder an die zeitkontinuierliche Regelstrecke an. Über ein Kennfeld wird die durch den Mikrokontroller vorgegebene Stellgröße in einer Kraft gewandelt. Anschließend verursacht der Block Motor_Delay mit einem PT1-Verhalten eine Verzögerung mit $\tau = 0.1$ ms was einem erreichen des Sollwertes nach ungefähr 0.5ms entspricht. Mit der Verstärkung M_error kann wie bereits erwähnt die Asymmetrie des Schubs der einzelnen Motoren simuliert werden. Ein Wert von „1.0“ entspricht den „normalen“ Wert, ein Wert von „1.1“ einem um 10% größeren Wert als „normal“. Aus dem Schub wird durch ein weiteres Kennfeld die Drehzahl der Rotoren errechnet.

Über den Block „Input Forces and Moments(not Emb)" werden aus dem Schub der einzelnen Motoren die Momente auf „Roll", „Pitch" und die Summenkraft berechnet. Für „Yaw" werden die Verhältnise der Rotordrehenzahlen berechnet und auf Grund dessen ein Moment für diesen Winkel errechnet.

**Formel 1: Bewegungsgleichungen**

$$\dot{p} = \frac{I_y - I_z}{I_x} q\, r - \frac{J_{rotz}}{I_x} q\, \Omega + \frac{U_2}{I_x}$$

$$\dot{q} = \frac{I_z - I_x}{I_y} p\, r + \frac{J_{rotz}}{I_y} p\, \Omega + \frac{U_3}{I_y}$$

$$\dot{r} = \frac{I_x - I_y}{I_z} p\, q + \frac{U_4}{I_z}$$

Die Regelstrecke basiert auf der Umsetzung der in Formel 1 vorgegebenen Bewegungsgleichungen als Simulinkblockschaltbild. Wobei p für die Winkelgeschwindigkeit von Phi(Roll), q für die Winkelgeschwindigkeit von Theta(Pitch) und r für die Winkelgeschwindigkeit von Psi(Yaw) steht.

Im oberen Zweig wird aus der Summenkraft(U1) und den aktuellen Winkeln des Quadrocopters die Beschleunigungen und durch integrieren die Geschwindigkeit und der Ort des Quadrocopters in Erdkoordinaten berechnet. Da sich die Regelung in dieser Version ausschließlich auf die Winkel bezieht ist er für diese Version nur zur Anschauung.

Im unteren Zweig werden die Momente(U2 … U3) und die Drehzahl in Winkelbeschleunigungen umgerechnet und durch integrieren in Winkelgeschwindigkeiten und Winkel umgerechnet. Diese Zustände sind genau die Zustände, welche in dieser Version geregelt werden.

Der hintere Teil des Blockschaltbildes fasst alle Zustände zum Signal pos-vel-acc zusammen und stellt diese somit den anderen Blöcken im Gesamtsystem zur Verfügung.

# Sensoren

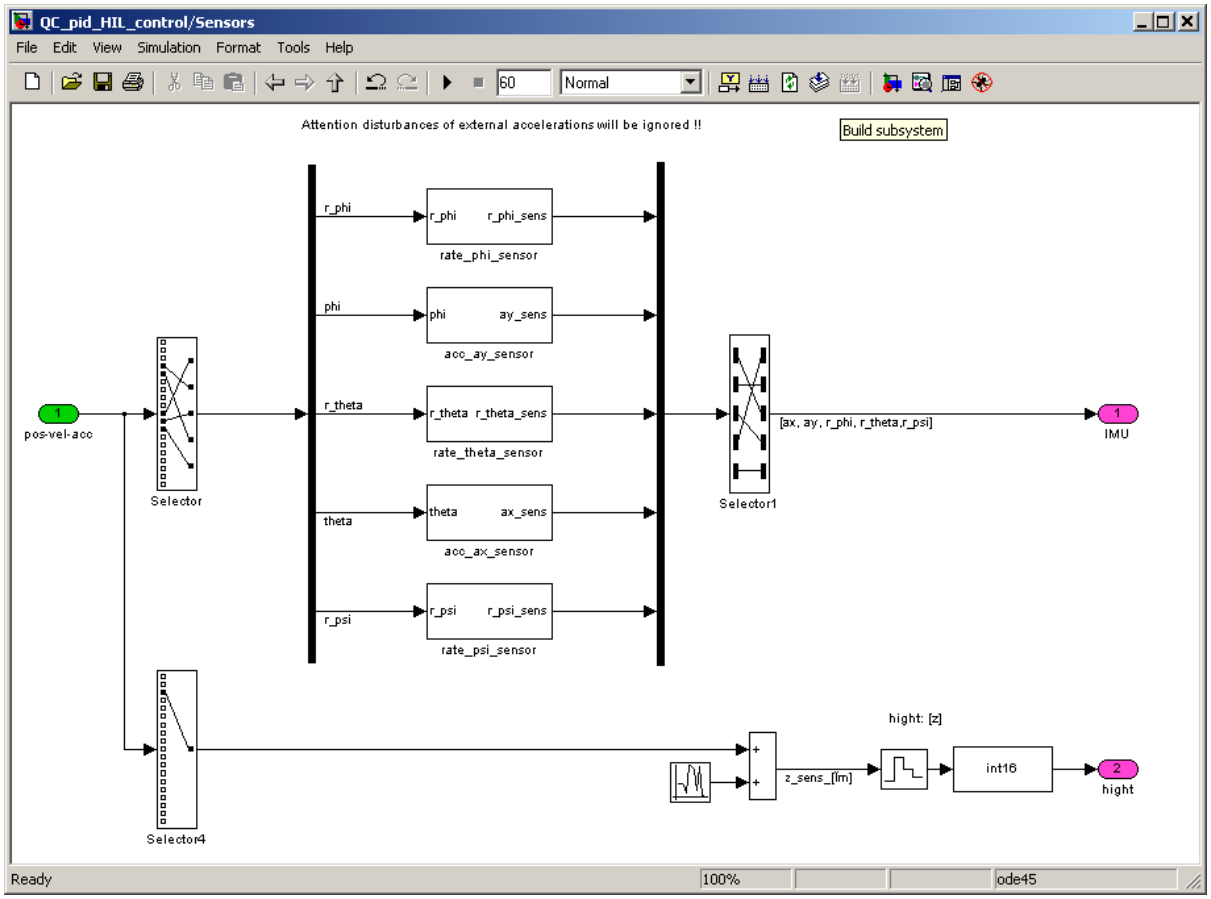


**Abbildung 4: Blockschaltbild des Sensorenblocks**

In diesem Block werden aus den Zuständen des Signals pos-vel-acc die Sensorsignale gebildet. Dies geschieht für die Beschleunigungssensoren und die Gyroskope verschieden.

## *Gyroskop*

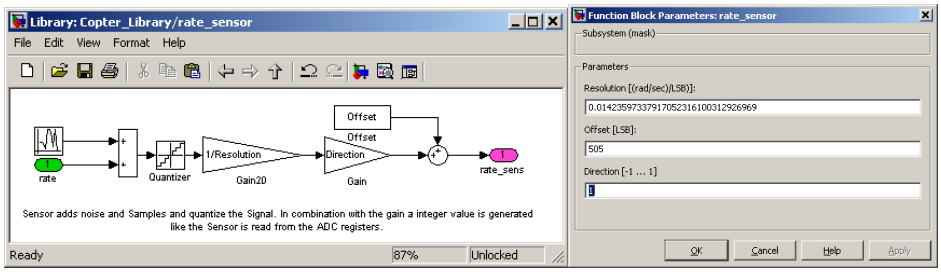


**Abbildung 5: Blockschaltbild eines Gyroskops**

Bei den Gyroskopen wird die jeweilige Winkelgeschwindigkeit aus dem Signal pos-vel-acc mit dem Sensorrauschen summiert. Anschließend wird das Signal quantifiziert in genau der Auflösung mit der über den AD-Wandler die Winkelgeschwindigkeit erfasst werden kann. Mit dem folgenden Verstärkungsblock wird dadurch genau diese Auflösung dividiert um einen Integerwert 0 … 1024 für den Wert des AD-Wandlers zu erhalten. Das Gyroskop besitzt eine Spannungsschnittelle. Um auch negative Werte auszugeben wird der Nullpunkt mit dem halben Spannungspegel realisiert. Um dieses Gegebenheit zu simulieren wird abschließend der Offset hinzuaddiert.

Der Faktor für die Quantifizierung und der Division(Multiplikation mit dem Kehrwert) berechnet sich aus der Auflösung des ADXRS610(Gyroskop)[1] und des 10Bit-AD-Wandlers, des verwendeten HC9S12 Mikrokontroller. Das Gyroskop liefert 3.96 (mV/(°/sec)). Die Umrechnung von Grad auf Rad erfolgt durch die Multiplikation mit 180/π und führt zu einer dem in der Dokumentation „Project Quadrocopter ASM2-SB“ im Kapitel 3.4 Angegebenen Auflösung von 226,89 (mV/(rad/sec)). Der AD-Wandler löst das Spannungsignal mit 10Bit auf was einer Auflösung von 3.3V/($2^{10}$-1)LSB, d.h. 3,23 mV/LSB, enstpricht. Ein LSB des AD-Wandlers entspricht daher 3,23(mV/LSB), 226,89(mV/(rad/sec)), d.h. 0,0142((rad/sec)/LSB).

***Um die Simulation realer darzustellen könnte zum konstanten Offset ein zeitabhängiger drift hinzugefügt werden (Simulation von Temperaturdrifts).***

[1] Diese Angabe wurde der Dokumentation des Projekts „Quadrocopter ASM2-SB, Kapitel 3.4“ entnommen und beschreibt die Auflösung der Winkelgeschwindigkeit bei einer Versorgungsspannung des Sensor mit 3.3V.

## *Beschleunigungssensor*

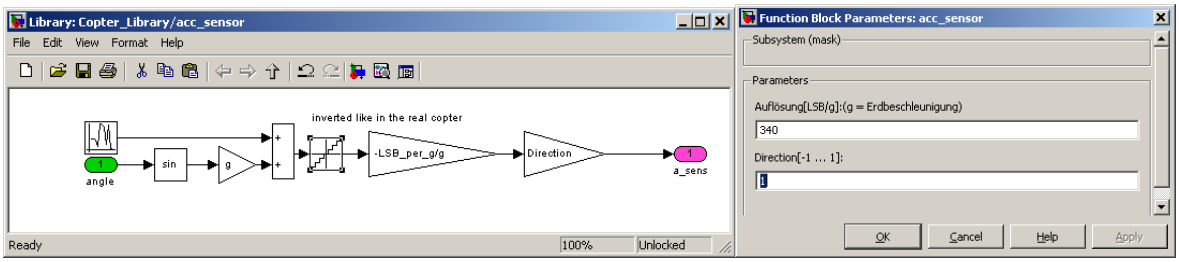


**Abbildung 6: Blockschaltbild für einen Beschleunigungssensor**

Beim Beschleunigungssensor wird aus den Zuständen der Winkel, Roll und Pitch, die jeweils anteilige Erdbeschleunigung berechnet, quantifiziert und, um die Integerwerte der SPI Daten zu erhalten, um genau diese Quantifizierung dividiert.

Der Faktor für die Quantifizierung und der Division(Multiplikation mit dem Kehrwert) berechnet sich aus dem im Datenblatt, des LIS3LV02DQ(Beschleunigungssensor), angegebenen typischen Wert für die Erdbeschleunigung von 340 LSB/Erdbeschleunigung (least significant bits). Das negative Vorzeichen ergibt sich durch die Lage des Sensors im Quadrocopters und gibt an in welcher Richtung die jeweilige Achse einen Positiven Wert ausgibt.

***Achtung! Die Daten enthalten weder Beschleunigungen in eine Richtung noch Beschleunigungen auf Grund von Störgrößen (durch zum Beispiel Seitenwind).***

# Animation

Die Animation basiert auf der des „Open Balancers“ siehe Studienarbeit von Herrn Czech „Modellierung und Regelung eines mobilen, inversen Pendels“. Da die Drehungen des Quadrocoper entgegen denen des Pendels nicht unabhängig voneinander sind muss eine andere Drehmatrix verwendet werden.

$$\begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}$$

Quelle: http://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel

Die Animation befindet sich in der Datei „Copter_Animation.m“ und ist in der Bibliothek

# Remote Control

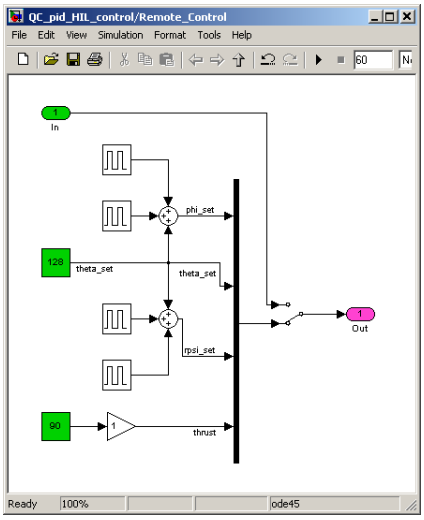


**Abbildung 7: Blockschaltbild für die Fernsteuerung**

Im Block „Remote_Control“ werden die Signale zum Steuern des Quadrocopters erzeugt. Über den Schalter kann hier festgelegt werden, ob genau diese Signale/Signalabfolgen, oder dass die über den Eingang 1 bereitgestellten Signale/Signalabfolgen zur Simulation verwendet werden sollen (letzteres wird benötig bei Simulation des Modells aus der Konsole mit dem Befehl ‚sim’). Egal welche quelle schließlich gewählt wird, befindet sich am Ausgang 1 der Vektor mit den Sollgrößen der Fernsteuerung.

Die „echte“ Fernbedienung sendet für die Winkel Roll, Pitch und die Winkelgeschwindigkeit für Yaw für die Nullstellung den Wert 128. Diese Konstante gibt somit den Offset für die Nullstellung an. Diese Signale können von 0 … 255 variiert werden. Die Konstante 90 gibt den Schub an, welcher von 0 … 255 variiert werden kann.

Über die Pulsgeneratorblöcke wird ein Sollwertsprung(Steuern des Quadrocopters in einer Richtung für eine Kurze Zeit vorgegeben, das mit nur einem Block nur ein Impuls in eine Richtung erzeugt werden kann, wird durch einen Weiteren Block die Aussteuerung in die entgegen gesetzte Richtung realisiert um zu verhindern, dass der Quadrocopter für die restliche Simulation in die gesetzte Richtung „weg schwebt“.

# Regelung

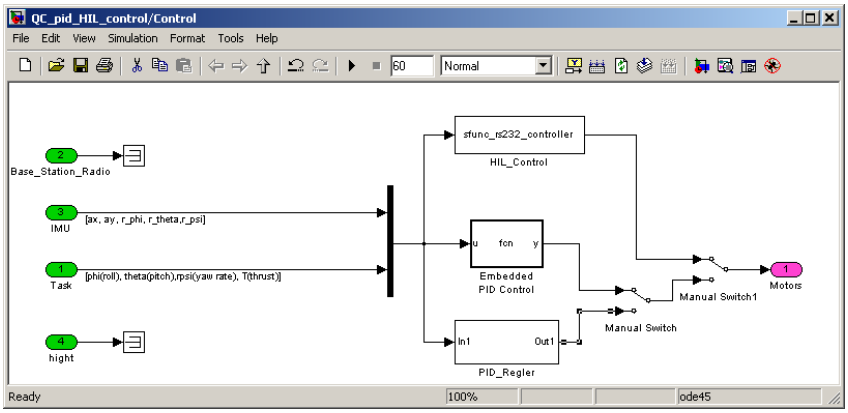


**Abbildung 8: Blockschaltbild der Regelung**

Im Block „Control“ befindet sich die Regelung der Winkel Pitch und Roll, sowie die Regelung der Winkelgeschwindigkeit des Winkels Yaw. Für diese Regelung werden als Eingangsinformationen die Sensordaten, In 3 „IMU“ und die Sollwerte der Fernsteuerung, In 1 „Task“, zu einem Vektorzusammen gefasst und in den drei Varianten der Regelung ausgewertet und die Stellgrößen der Motoren bestimmt. Über die Schalter kann ausgewählt werden, welche Variante das Modell wirklich regelt.

## *Regelung eines Winkels*

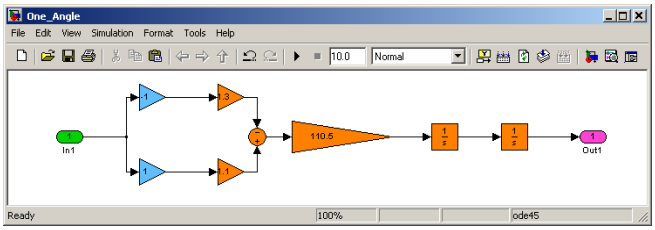


**Abbildung 9: Regelstrecke eines Winkels**

Ein Winkel kann wie in Abbildung 9 als Blockschaltbild dargestellt werden. Die Eingangsgröße wird als Stellgröße auf die beiden Motoren gegeben die resultierende Differenz, das Moment wird über den folgenden Verstärkungsfaktor in eine Winkelbeschleunigung gewandelt und anschließend über die Integratoren in den die Winkelgeschwindigkeit und in den Winkel gewandelt. Aus diesem Blockschaltbild ergibt sich das Bodediagram aus Abbildung 10.

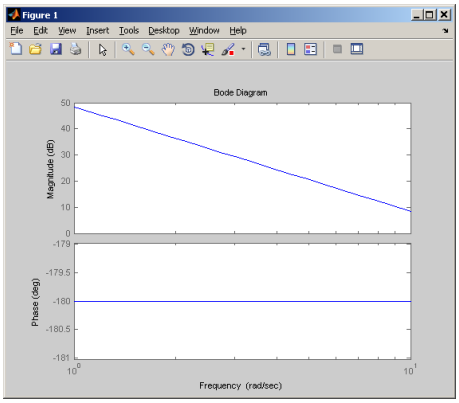


**Abbildung 10: Bodediagram der Regelstrecke eines Winkels**

Wie aus dem Bodediagram ersichtlicht ist erzeugen die beiden Integratoren eine Phasenverschiebung von -180°. Diese verletzt das Stabilitätskriterium von Nyquist. Wird der Ausgang zurückgekoppelt, so ist dieses System auch bei niedrigen Frequenzen instabil. Die Konsequenz ist, dass dieses System mit einer Kaskade geregelt werden muss. Das heißt bereits die Winkelgeschwindigkeit muss auf den Eingang zurückgekoppelt werden, sowie der Winkel selbst. Für einen Idealen Quadrocopter würde es reichen beide Regler der Kaskade mit einem P-Regler zu regeln. Da bei einem realen Quadrocopter jedoch alle Motoren einen unterschiedlichen Schub bei gleicher Stellgrößen, auf Grund von Tolleranzen der Motoren und unterschiedlicher Lastverteilung, liefern, muss der innere Regler als PI-Regler ausgelegt werden. Dieser gleicht die genannten Asymmetrien vollständig aus. Der in Abbildung 11 realisierte Regler nutzt zwar den Block „PID“, verwendet aber für den äußeren Regler nur den P-Anteil und für den inneren nur den P- und I-Anteil.

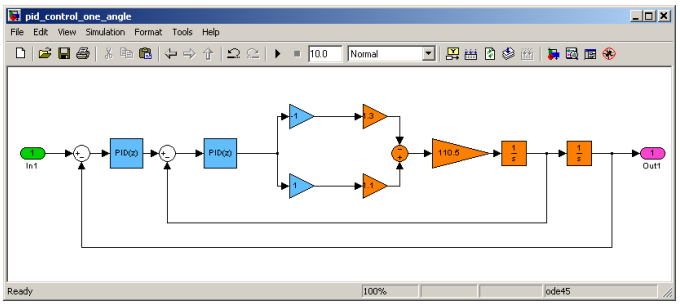


**Abbildung 11: Regelung eines Winkels**

Für die Regelung der Winkelgeschwindigkeit des Winkels Yaw wird nur der innere Regelkreis benötigt.

## *Regelung aller Winkel*

Durch Überlagerung der Stellgrößenänderungen der Regler für jeweils einen Winkel und das Hinzufügen eines Offset(thrust_set) für den Gesamtschub ergibt die Stellgrößen für die 4 Motoren(siehe Abbildung 12). Die Sättigung direkt am Ausgang des Blockschaltbildes begrenzt die geforderte Stellgröße auf die minimal bzw. maximal mögliche.

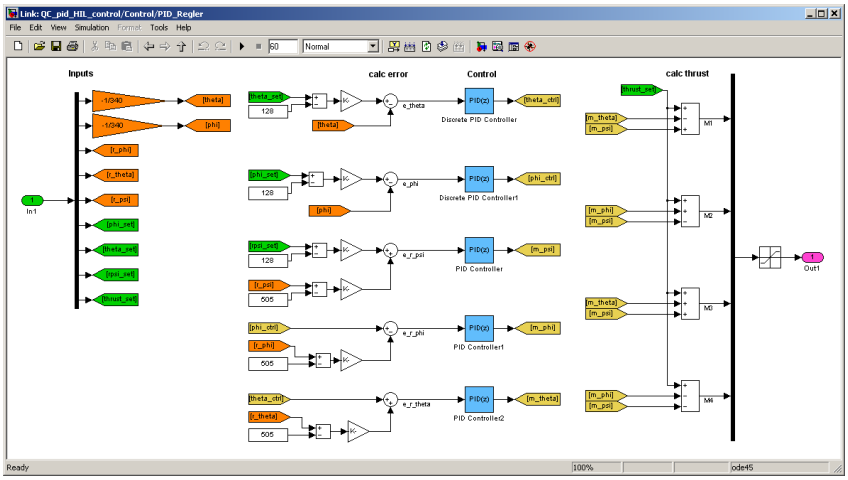


**Abbildung 12: Regelung als Blockschaltbild**

Vor den jeweiligen Reglern werden die Sollwerte der Fernsteuerung und die Signale der Sensoren in Winkel bzw. Winkelgeschwindigkeiten umgerechnet.

## *Umsetzung Regelung in Embedded MATLAB*

Die erstellte Regelung muss im Mikrocontroller des Quadrocopters als Quellcode implementiert werden. Da sich dieser, selbst beim „Modell in the Loop“ (MIL) test, nur zeitaufwendig debuggen lässt. Macht es Sinn die Regelung zuerst in Embedded MATLAB Code zu realisieren.

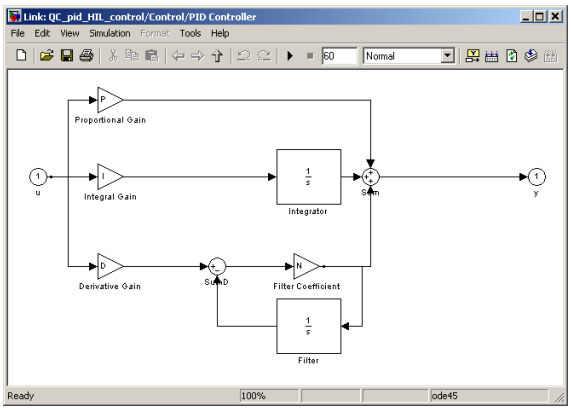


**Abbildung 13: Inhalt des PID Blocks**

Ein PID-Regler wurde im Embedded MATLAB Code so implementiert, dass einer Funktion die Anteile P/I/D, der Filterkoeffizient N, der aktuelle und der vorherige Abtastwert, der letzte Zustand der Integratoren für den I-Anteil und den Filter, sowie die Abtastzeit übergeben wird. Innerhalb der Funktion wurde der Inhalt des Blockschaltbildes aus Abbildung 13 realisiert.

```
function [ret, I_State, Filt_State] = PID (P, I, D, N, ek, ek_1, I_State, Filt_State, Ts)

if P ~= 0
    if I ~= 0
        % calculate new itegral value
        I_State = ek * Ts * I + I_State;
    else
        I_State = 0;
    end

    if D ~= 0
        % calculate new derivative Value
        yd = (ek - ek_1)/Ts * D;

        % Filter the derivative  Value
        Filt_State = Ts * (yd -N*Filt_State)+ Filt_State ;
    else
        Filt_State = 0;
    end

    % Add all values and multiply P gain to all
    ret = (P * ek + I_State + Filt_State);

else
    ret = 0;
end
```

In der Hauptfunktion werden die statischen Variablen mit dem prefix „persistent“ deklariert und unter der Bedingung „if isempty(Variablenname)“ definiert. Die Eingänge (u(1) bis u(9)) werden einer Struktur „copterState“ zugewiesen, welche auch in der Mikrocontrollersoftware des Quadrocopters vorhanden ist.

```
% Accelerometer
%%%%%%%%%%%%%%%%%%
copterState.accY = int16(((int32(copterState.accYRaw) * G) / sensorSensitivityY) / 10);
copterState.accX = -int16(((int32(copterState.accXRaw) * G) / sensorSensitivityX) / 10);
copterState.accZ = -int16(((int32(copterState.accZRaw) * G) / sensorSensitivityZ) / 10);
%// [ 10000*rad ]
copterState.angP = -int16(int32(copterState.accXRaw)*1000/340);
copterState.angR = -int16(int32(copterState.accYRaw)*1000/340);

% Gyroscopes [100 * rad/sec]
%%%%%%%%%%%%%%%
copterState.angVelP = -int16(double(copterState.angVelPRaw-copterState.gyroPOffset)*0.938349);
copterState.angVelR =  int16(double(copterState.angVelRRaw-copterState.gyroROffset)*0.938349);
copterState.angVelY = -int16(double(copterState.angVelYRaw-copterState.gyroYOffset)*0.938349);

% Remote
%%%%%%%%%%%%%%
copterState.remotePitch    =  int16(maxPitchAngle*int32(copterState.remotePitchRaw-128)/maxRemotePitch);
copterState.remoteRoll     = -int16(maxRollAngle*int32(copterState.remoteRollRaw-128)/maxRemoteRoll);
copterState.remoteForce    =  int16(copterState.remoteForceRaw);
copterState.remoteYaw      =  int16(maxYawRate*int32(copterState.remoteYawRaw-128)/maxRemoteYaw);
```

Ferner wird in der Hauptfunktion aus den Eingangssignalen die Winkel und Winkelgeschwindigkeiten gebildet dieser Teil wird in der Mikrocontrollersoftware von der HAL übernommen.

```
% Set outputs to Values calculated last sample time
y = double([M1_limited;M2_limited;M3_limited;M4_limited]);
```

Bei der Diskretisierung wurde der Euler-Vorwärts verwendet. Daher wird zum Beginn der Regelung die Stellgrößen für die Motoren ausgegeben.

```
% calculate error values for the controlled angles
   % Pitch error
   e_Pitch = double(copterState.remotePitch) - angP;
   % Roll error
   e_Roll = double(copterState.remoteRoll) - angR;
   % calculate controller output for the angles (I State & FilterState are not necessary cause only p is
used
   [m_Roll, IRoll, FiltRoll] = PID(150,0,0,0,e_Roll,e_Roll_1,IRoll, FiltRoll, Ts);
   [m_Pitch, IPitch, FiltPitch] = PID(150,0,0,0,e_Pitch,e_Pitch_1,IPitch,FiltPitch,Ts);
```

Für die Regelung wird zuerst die Regeldifferenz von Soll- und Ist-Winkeln gebildet anschließend werden über die Funktion für den PID-Regler(mit dem Wert P=150)die Stellgrößendifferenz berechnet.

```
% Yaw Rate error
   e_r_Yaw = double(copterState.remoteYaw) - double(copterState.angVelY);
   % Pitch Rate error
   e_r_Pitch = m_Pitch - double(copterState.angVelP);
   % Roll Rate error
   e_r_Roll = m_Roll - double(copterState.angVelR);
   % calculate  controller output for rates
   [m_r_Yaw,r_Yaw_I_State, r_Yaw_Filter_State] = PID (70, ...
       0.5, 0,0,e_r_Yaw, e_r_Yaw_1, r_Yaw_I_State, r_Yaw_Filter_State,Ts);
   [m_r_Roll,r_Roll_I_State, r_Roll_Filter_State] = PID (70, ...
       0.5, 0,0,e_r_Roll, e_r_Roll_1, r_Roll_I_State, r_Roll_Filter_State,Ts);
   [m_r_Pitch,r_Pitch_I_State, r_Pitch_Filter_State] = PID (70, ...
       0.5, 0,0,e_r_Pitch, e_r_Pitch_1, r_Pitch_I_State, r_Pitch_Filter_State,Ts);
```

Die Winkelgeschwindigkeiten werden bis auf die des Yaw auf 0 Ausgeregelt. Daher bestimmt sich die Regeldifferenz für die Winkel Roll und Pitch aus der vorher berechneten Stellgrößendifferenz des äußeren Reglers und der gemessenen Winkelgeschwindigkeit. Bei der Winkelgeschwindigkeit des Yaw bildet sich die Regeldifferenz aus dem Sollwert und der gemessenen Winkelgeschwindigkeit. Über die Funktion für den PID-Regler(mit den Werten P=70 und I=0.5) werden die Stellgrößendifferenzen berechnet.

```
%//Calculate new setpoints of all motors
    M1 = (copterState.remoteForce-m_r_Pitch+m_r_Yaw);
    M2 = (copterState.remoteForce+m_r_Roll-m_r_Yaw);
    M3 = (copterState.remoteForce+m_r_Pitch+m_r_Yaw);
    M4 = (copterState.remoteForce-m_r_Roll-m_r_Yaw);
```

Die berechneten Stellgrößendifferenzen und der Gesamtschub werden überlagert und ergeben die benötigten Stellgrößen.

```
if M1 < 0
      M1_limited = uint8(0);
    else
        if M1 > 255
            M1_limited = uint8(255);
        else
            M1_limited=uint8(M1);
        end
    end
```

Bevor die Stellgrößen, zur Ausgabe bei der nächsten Berechnung der Regelung, gehalten werden, werden diese auf Werte von 0 … 255 gesättigt.

Der Embedded MATLAB Block mit der Regelung befindet sich in der Bibliothek „Control_lib“.

## *Model in the Loop(MIL)*

Um den in MATLAB verwendeten Code zu verwenden, muss dieser auf den im Quadrocopter verwendeten Mikrocontroller portiert werden. Um mögliche numerische oder systematische Fehler und Probleme vor dem ersten „realen“ Flug abzufangen, wurde ein Model in Loop Test realisiert. Hierfür wurde eine Level 1 S-Function verwendet. Mit dieser wurde eine serielle Verbindung über ZigBee mit dem Quadrocopter geöffnet. Nach dem Master Slave Prinzip wurde eine Anfrage mit den in der Simulation vorhandenen Eingangssignalen(IMU, Task) zum Quadrocopter gesendet, dort die Stellgrößen berechnet und diese wieder zurück an die S-Function gesendet. Auf diese Weise wurde das Simulink Model über die Mikrocontrollersoftware geregelt. Diese Variante hat jedoch den Nachteil, dass die ZigBee Kommunikation im Quadrocopter zyklisch nur alle 100ms verarbeitet wird und nur eine Baudrate von 9600 verwendet wird.

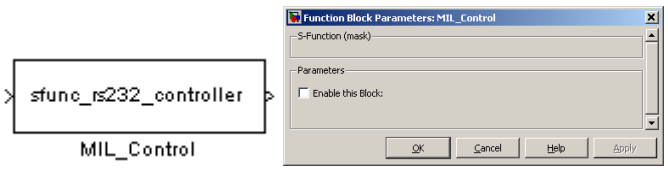


**Abbildung 14: Model in the Loop Block**

Da auf Grund dessen die Simulationsgeschwindigkeit massiv abgebremst wird, wurde die S-Function um einen Parameter erweitert um die Kommunikation abzuschalten wenn diese nicht explizit benötigt wird.

```
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
      if Config.Enabled
         s = serial('COM2','BaudRate', 9600,'Timeout', 1);
         fopen(s);
      end
     [sys,x0,str,ts]=mdlInitializeSizes(Config);
```

In der S-Function wird im Initialisierungsschritt die serielle Schnittstelle geöffnet. Während der Erstellung war es nicht möglich den COM-Port als Variable herauszuführen, deshalb ist dieser an dieser Stelle fest codiert. Die serielle Schnittstelle ist in der S-Function als statische Variable über den prefix persistent gehalten.

```
function [sys,x0,str,ts]=mdlInitializeSizes(Config)
% Set Sizes Matrix
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 9;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

% initialise the initial conditions
x0  = [];

% str is always an empty matrix
str = [];

% initialise the array of sample times
ts  = [0.005 0]; % variable sample time
```

Über die Funktion mdlInitializeSizes wird der Block initialisiert, so dass der Block einen Eingangsvektor der Breite 9 und einen Ausgangsvektor der Breite 4 besitzt. Zusätzlich wird mit „DirFeedthrough = 1“ angegeben, dass Eingänge direkt am Ausgang(ohne zwischenspeichern in Zuständen) ausgegeben wird. Schließlich wird in dieser Funktion die Sample Zeit mit 5 Millisekunden Simulationszeit vorgegeben.

```
function sys = mdlOutputs(t,x,u,s, Config)
if Config.Enabled
    i          = 0;
    ID         = uint8(48); % 0x30
    Length     = uint8(14); % 0x12
    ax         = uint8(double2uint8(u(1)));
    ay         = uint8(double2uint8(u(2)));
    r_phi      = uint8(double2uint8(u(3)));
    r_theta    = uint8(double2uint8(u(4)));
    r_psi      = uint8(double2uint8(u(5)));
    phi_set    = uint8(u(6));
    theta_set  = uint8(u(7));
    r_psi_set  = uint8(u(8));
    thrust     = uint8(u(9));
    sendvect = [ID; Length;ax;ay;r_phi;r_theta;r_psi;phi_set; theta_set; r_psi_set;thrust];
    sendvectint8 = uint8(crc16(double(sendvect),16));
```

In der Funktion mdlOutputs werden die Sensorinformationen zu einem Vektor zusammengefasst, und die im Quadrocopter Protokoll notwendige CRC16 Prüfsummer berechnet. Auch hier findet die Kommunikation nur statt, wenn der entsprechende Parameter gesetzt ist.

Im weiteren Teil werden die Daten gesendet. Da es vorkommen kann, dass ein Datenpaket verloren geht wird es im Timeout fall(Der Quadrocopter antwortet nicht) bis zu 2 weitere Male wiederholt. Für eine Stabilere Kommunikation hat sich herausgestellt, das jeweils vor einer Datenübertragung eine Pausenzeit von +100ms sinnvoll ist.

```
    pause(0.1);
    while i < 3
       fwrite(s, sendvectint8, 'uint8');
       pause(0.1);
       Ret = fread(s, 8, 'uint8');
       if size(Ret) ~= (2+4+2);
          display('Timeout once')
       else
          sys = double(Ret(3:6));
          break;
       end
    i= i+1;
    end
    if size(Ret) ~= (2+4+2)
          display(' !!!!!! --- Total timeout --- !!!!!!')
          sys = [0;0;0;0];
    end
else
    sys = [0;0;0;0];
end
```

Nach dem die Stellgrößen vom Quadrocopter empfangen wurden, werden diese am Ausgangsvektor ausgegeben und so dem Modell zugeführt. Sollte die Kommunikation Fehlschlagen wird dies an der Console und 0 als Stellgröße für alle Motoren ausgegeben.

```
  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
      if Config.Enabled
         fclose(s);
         clear s
      end
     sys=[];
```

Beim beenden des Blocks wird die serielle Schnittstelle geschlossen.

Innerhalb der Mikrocontrollersoftware des Quadrocopters wird durch die Daten der S-Function in der Funktion „basestation“ im gleichnamigen Modul der Case-Fall für den Model in the Loop test aufgerufen.

```
      case 0x30:
      {
        txMsg.id = 0x31;
        txMsg.dataLen = 4;

        //DisableInterrupts;
        copterState = mycopterGetStatePtr();
        // accelerometer
        copterState->accXRaw = rxMsg.data[0]*256 + rxMsg.data[1];        // -2048 .. 2047
        copterState->accYRaw = rxMsg.data[2]*256 + rxMsg.data[3];        // -2048 .. 2047

        // ATD; gyroscopes, temperature, air pressure and battery voltage
        copterState->angVelRRaw = rxMsg.data[4]*256 + rxMsg.data[5];     // 0 .. 1024
        copterState->angVelPRaw = rxMsg.data[6]*256 + rxMsg.data[7];     // 0 .. 1024
        copterState->angVelYRaw = rxMsg.data[8]*256 + rxMsg.data[9];     // 0 .. 1024

        copterState->remoteRollRaw   = rxMsg.data[10];  // 0 .. 255
        copterState->remotePitchRaw  = rxMsg.data[11]; // 0 .. 255
        copterState->remoteYawRaw    = rxMsg.data[12];   // 0 .. 255
        copterState->remoteForceRaw  = rxMsg.data[13]; // 0 .. 255

        (void) accelCalcPhysicalValues(copterState);
        (void) atdCalculatePhysicalValues(copterState);
        (void) calcPhysicalValuesFromRemote(copterState);

        // sensorFilter(copterGetConfigPtr(),copterState);
        // compute control algorithm
        (void) Control(copterState, copterConfig);

        txMsg.data[0] = copterState->setpointFront;
        txMsg.data[1] = copterState->setpointLeft;
        txMsg.data[2] = copterState->setpointRear;
        txMsg.data[3] = copterState->setpointRight;
        //EnableInterrupts;

        (void) xbeeSendMsg(&txMsg);
        break;
      }
```

In diesem Programmteil werden die empfangenen Daten in die Struktur copterState kopiert, welche zwar vom Datentyp der echten Struktur entspricht, aber nur während des MIL-Test unabhängig von der eigentlich verwendeten Struktur gelesen und geschrieben wird. Somit wird verhindert, dass von den „echten“ Sensordaten Informationen die der S-Function überschrieben werden.

In der HAL für die Sensoren wurde die Programmteile zum umrechnen der Eingangsgrößen in Winkelgeschwindigkeiten, Winkel und Beschleunigungen als separate Funktionen realisiert. Durch die eigene Struktur für den MIL-Test können diese Funktionen zum konvertieren der S-Function Informationen sowie auch für die realen Sensordaten verwendet werden. So können über den MIL-Test sogar Teile der HAL mit getestet werden. Nach dem die S-Function Werte für die Gyroskope, Beschleunigungssensoren und die Fernsteuerung so konvertiert wurden wird die Funktion Control, welche die Regelung beinhaltet ebenfalls mit der Daten der S-Function aufgerufen und dort die Stellgrößen berechnet. Die Funktion Control enthält die Umsetzung des Embedded MATLAB Codes in C.

## *Filtern der Gyroskopsignale*

Während der Entwicklung hat sich herausgestellt, dass es nicht ausreicht den Nullpunkt Gyroskope zu beginn der Applikation festzuhalten(in statischer Variablen zu speichern). Es hat sich gezeigt, dass der Nullpunkt während des Betriebs auf Grund von Temperaturänderung „weg läuft“. Abhilfe wurde durch einen Digitalen Hochpass geschaffen:

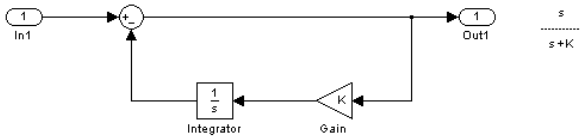


**Abbildung 15: Hochpass 1. Ordnung (zeitkontinuierlich)**

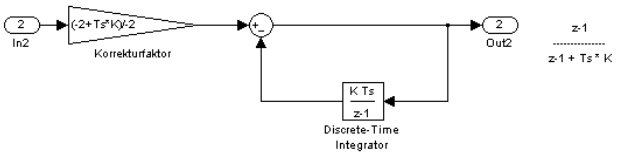


**Abbildung 16: Hochpass 1.Ordnung (zeitdiskret)**

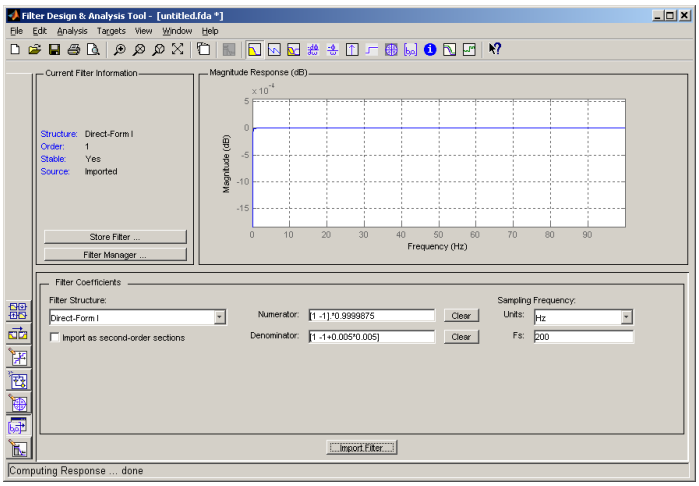


**Abbildung 17: Berechnung des HP-Filters mit MATLAB fdatool**

In Abbildung 15 zeigt das Blockschaltbild des Zeitkontinuierlichen Hochpassfilters dieser wurde wie in Abbildung 16 diskretisiert. Abbildung 17 zeigt den Frequenzgang des Filters mit dem MATLAB-Tool „fdatool“. Über den Faktor K kann die Grenzfrequenz eingestellt werden. Dies ist erforderlich, da man zwar das Offset Filtern möchte, jedoch auch den geringen Drift auf Grund Temperaturänderungen unterdrücken möchte. Andererseits möchte man jedoch auch geringe Bewegungen mit den Gyroskopen wahrnehmen, daher muss K im Quadrocopter so appliziert werden, dass in der Ruhelage gerade kein oder ein sehr geringer Drift ersichtlich ist.

## *Berechnung des Winkels aus den Sensordaten*

Um den Winkel zu erhalten reicht es in bei einem idealen Quadrocopter die Winkelgeschwindigkeit zu integrieren. In der Realität gibt es jedoch Fehler auf Grund der Numerik beim integrieren und Fehler durch den Drift (der HP-Filter kann nie so ideal eingestellt werden, so dass der Drift vollständig gefiltert und dennoch langsame Bewegungen vollständig erkannt werden können). Aus diesem Grund kann durch integrieren nur kurzzeitig aus der Winkelgeschwindigkeit sinnvollen Daten für den Winkel erlangt werden.

Eine weitere Möglichkeit bietet die Berechnung des Winkels aus den vorherrschenden Beschleunigungen. Dies ist möglich, da die Endbeschleunigung ständig auf den Quadrocopter wirkt. Neigt sich dieser, verteilt sich die Erdbeschleunigung auf die X- und Y-Achse:

**Formel 2: Berechnung des Neigungswinkels an der Y-Achse**

$$\varphi = \arcsin\left(\frac{a_x}{g}\right)$$

Diese Berechnung liefert jedoch nur sinnvolle Werte, wenn sich der Quadrocopter in ruhe befindet, da Beschleunigungen in eine Richtung(Flug in X-/Y-Richtung, Kurvenflug, …) den bestimmten Winkel verfälschen. Durch einen Tiefpass-Filter können diese schnellen Änderungen gefiltert werden. Für kleine Winkeländerungen kann auf die Arkusfunktion theoretisch verzichtet werden, der Quadrocopter wird jedoch auch bei Winkeln >10° geflogen. Deshalb wird die Arkussinus-Funktion benötigt und im Mikrocontroller des Quadrocopters über eine interpolierte Kennlinie realisiert.

Die Probleme der beiden Varianten können gegeneinander kompensiert werden, da diese genau in entgegen gesetzten Frequenzbereichen auftreten. Aus diesem Grund wird ein Komplementärfilter(siehe „Projekt Quadrocopter“) eingesetzt.

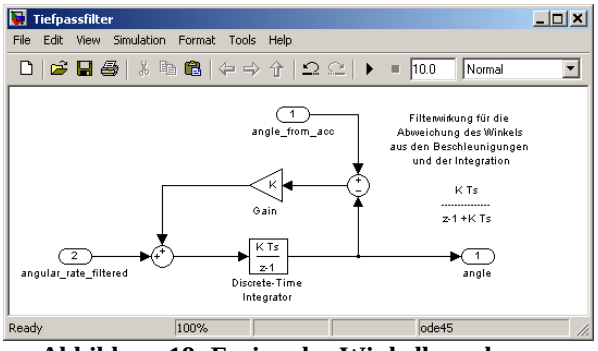


**Abbildung 18: Fusion der Winkelberechnung**

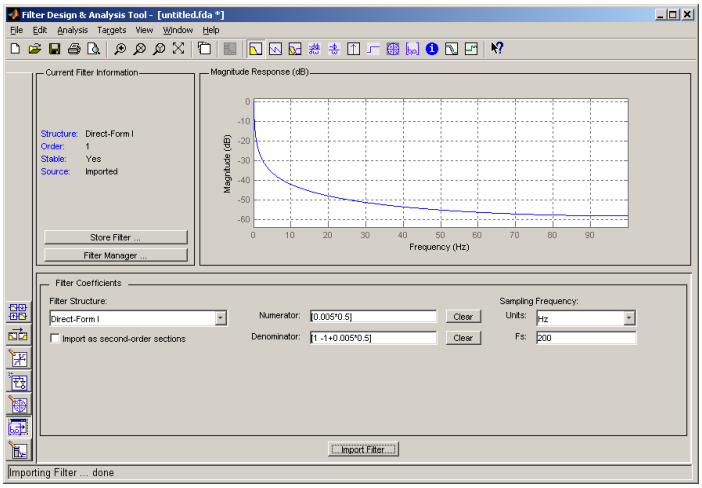


**Abbildung 19: Berechnung des TP-Filters mit MATLAB fdatool**

In Abbildung 18 wird gezeigt wie die Winkelbestimmung fusioniert wird. Die bereits durch einen Hochpass gefilterte Winkelgeschwindigkeit, wird integriert, mit dem aus den Beschleunigungen bestimmten Winkel verglichen und die resultierende Differenz über einen Faktor K zurückgekoppelt. Die Filterwirkung der Korrektur ist in Abbildung 19 ersichtlich. Durch Variation von K kann die Grenzfrequenz des Tiefpassverhaltens variiert werden. Somit kann vorgegeben werden wie „stark“ der Winkel korrigiert werden soll.

# Quick-Start (Matlab-Simulation)

## *Starten*

Die Parameter des Modells werden vom Workspace bezogen. Über das M-File *quadrocopter_param.m* sollten diese dort geladen werden.

Wird der MIL-Test verwendet kann der Fehlerauftreten, dass der COM-Port bereits geöffnet ist. Dieser kann über folgenden Befehl in der MATLAB-Konsole geschlossen werden:

*fclose(instrfind())*

Für die Simulation kann es erforderlich werden den Pfad mit den Bibliotheken Control_lib.mdl, Copter_Library.mdl, Copter_Animation.m, crc16.mexw32 und hinzuzufügen

Was kann man einstellen und wie?

## *Speicherort*

Die MATLAB/Simulink-Dateien befinden sich im SVN unter:
*https://svn.hs-esslingen.de/~dsatikid/svn/MasterThesis/MASTER_THESIS_PRJ/trunk/scratch/RobertCzech/Quadrocoptercontrol*
Innerhalb der einzelnen SVN Verzeichnisse befindet sich Textdateien mit dem Namen Content.txt welche Auskunft über den Inhalt liefern.

Die Mikrocontrollersoftware befindet sich im SVN unter:
*https://svn.hs-esslingen.de/~dsatikid/svn/MasterThesis/MASTER_THESIS_PRJ/trunk/scratch/RobertCzech/Copter_Software_modified*