Project Quadrocopter

Automotive Systems Master

Software Based Automotive Systems

ASM2-SB / SS 2010

QC Control & Sensor Filtering



*Prof. Dr. rer. nat. (Purdue Univ.) Dipl.-Ing. Jörg Friedrich*

*Tutor: Dionysios Satikidis*

Joachim Schwab

Sreekanth Sundaresh

Nina Vetlugina

Marc Weber

Contents

[1 QC Control Algorithm 2](#_Toc267508523)

[1.1 Control modelling 3](#_Toc267508524)

[1.2 PID techniques 6](#_Toc267508525)

[1.2.1 Roll Control 8](#_Toc267508526)

[1.2.2 Pitch Control 8](#_Toc267508527)

[1.2.3 Yaw Rate Control 9](#_Toc267508528)

[2 Sensor Filtering 10](#_Toc267508529)

[2.1 Complementary Filter 10](#_Toc267508530)

[2.1.1 To calculate the angle from x-axis Accelerometer Value 11](#_Toc267508531)

[3 Model Transformation into Embedded version Realization 14](#_Toc267508532)

[3.1 Brief Info about S-Functions 14](#_Toc267508533)

[3.1.1 Simulation Stages and S-Function Routines 15](#_Toc267508534)

[3.2 Implemented S-Function blocks of Quadrocopter 16](#_Toc267508535)

[3.2.1 S-Function block Quadrocopter Controller 17](#_Toc267508536)

[3.2.2 S-Function block Sensor Filter 18](#_Toc267508537)

[3.3 *Visual Studio* settings for compiling S-Functions 19](#_Toc267508538)

[4 Test cases & Simulation results 26](#_Toc267508539)

[5 Improvements & Conclusions 27](#_Toc267508540)

[Appendix A: Subsystem Interfaces 28](#_Toc267508541)

[A 1 Subsystem Quadrocopter Controller 28](#_Toc267508542)

[A 1.1 Inputs 28](#_Toc267508543)

[A 1.2 Outputs 29](#_Toc267508544)

[A 1.3 Parameters 29](#_Toc267508545)

[A 1.4 Constants 30](#_Toc267508546)

[A 2 Subsystem Sensor Filter 30](#_Toc267508547)

[A 2.1 Inputs 31](#_Toc267508548)

[A 2.2 Outputs 31](#_Toc267508549)

[A 2.3 Parameters 31](#_Toc267508550)

[A 2.4 Constants 32](#_Toc267508551)

[References 33](#_Toc267508552)

[Figure 1: Control Block Diagram 5](#_Toc267508512)

[Figure 2: PID Structure 7](#_Toc267508513)

[Figure 3: Block diagram of the Roll Control 8](#_Toc267508514)

[Figure 4: Block diagram of the Pitch Control 8](#_Toc267508515)

[Figure 5: Block diagram of the Yaw Rate Control 9](#_Toc267508516)

[Figure 6: Quadrocopter lying on the reference platform 11](#_Toc267508517)

[Figure 7: Quadrocopter tilted w.r.t the reference platform 11](#_Toc267508518)

[Figure 8: Fusion of Accelerometer & Gyro to get the filtered angle estimate 12](#_Toc267508519)

[Figure 9: How Simulink® Performs Simulation 15](#_Toc267508520)

[Figure 10: S-Function Quadrocopter controller 17](#_Toc267508521)

[Figure 11: S-Function Sensor Filter 18](#_Toc267508522)

# QC Control Algorithm

In this chapter, the basic control of the quadrocopter (Flight control) is presented. In the first step, the control algorithm is developed by Matlab/Simulink® basic blocks & tested with some probabilistic & “near-real-life/real-time” test cases. The control algorithm(“Controller”) is interfaced with the quadrocopter model (“Plant”) (which is also developed by Matlab/Simulink® basic blocks). Together with this arrangement, a filtering algorithm (“Sensor Filter”) is later developed in the feedback loop to filter the sensors values for its “drifts”.

For the simulations, the optimized values for control parameters were “auto-tuned” with the Matlab/Simulink’s® in-built tuning functions.

Later in the second step, tests were carried out on the quadrocopter platform to evaluate the behavior of the real system.

As this chapter is closely coupled with the “QC Modelling” chapter, we try to analyze the QC Model itself & attempt to “invert” some of the equations to control the quadrocopter.

The first section (1.1: Control modelling) shows the basic quadrocopter model simplifications. These must be done to be able to use an easier controller and to lower the algorithm complexity.

The second section (1.2: PID techniques) introduces the PID concepts and its strengths. After that it shows and explains in detail the 3 inner control diagrams. Their goal is to determine the basic movement signals from accelerometer and gyros data (sensors) and from task references (remote controller).

## Control modelling

Since the quadrocopter model is described in the previous chapter in sufficient detail, here we summarize most important equations.

The first equation shows, how the quadrocopter accelerates according to the basic movement commands given.

(1.1)

The second system of equations explains how the basic movements are related to the forces to the individual propellers.

(1.2)

The propellers’ forces are transformed into the speed of the individual rotors using a transformation curve characteristics (from the manufacturer), since the BLDC motors expect the rotor speed as an input to control the speed of the individual rotors.

From these set of equations, it is possible to determine the quadrocopter position by integrating the accelerations (linear & angular) twice. To do this operation, just the internal state of the quadrocopter and the speed of the four motors must be managed. This process is also known as direct kinematics and direct dynamics.

Controlling of the quadrocopter is done by finding the right values of the speed of the motors which maintains the quadrocopter in a stable position commanded by the task references. To do this, one should invert the model characteristics & equations. This process is also known as inverse kinematics and inverse dynamics. Since the inverse operations are bit complicated to perform, we present here some simplified inverse equations from equation (1.1)

Since we can only command the Roll, Pitch & Yaw angles from the remote controller, we try to control the angular accelerations of Roll, Pitch & Yaw values. For these, 3 out of 6 equations described in the equation (1.1) is sufficient to simplify & invert. Hence we get the following set of equations.

(1.3)

Or

(1.4)

The control algorithm receives, as inputs, the data from the sensors and from the remote controller. The output of the control algorithm is the speed for the four motors.

The control algorithm itself contains 2 blocks – “Inner Control Algorithm” & “Inverted Movement Matrix”

Figure : Control Block Diagram

* “Inner Control Algorithm” represents the core PID controllers. It processes the command received from the *remote controller* & *sensor data* to form the error signal to each of the PID controller. Output of each PID controller gives a signal for each of the basic movements which balances the position error
* “Inverted Movement Matrix” represents a block which calculates the forces for the individual propellers from the 4 basic movement signals. This is obtained by inverting the equation (1.2)

(1.5)

## PID techniques

In most of the general industrial applications, the most widely used controllers are the PID controllers because of the following reasons,

* Simple & robust structure
* Good performance for several processes
* Adjustable even without a specific model of the controlled system

Hence we have also chosen the PID controllers in our project to control the quadrocopter.

So the quadrocopter is constantly controlled by a fast digital feedback loop. The sensors are read and then the error is calculated.

Since the control loop is the PID controller, the output is controlled by 3 components as shown in the figure.

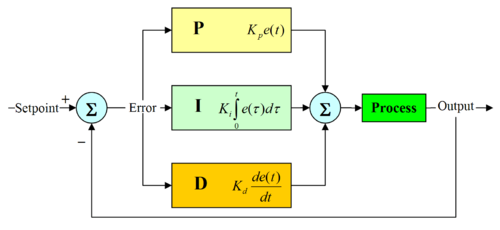


Figure : PID Structure

* Proportional: This is the error in output multiplied by a number. This defines the proportional bandwidth.
* Integral: This is the integral value of the output. In other words, by constantly adding all errors together it is possible to create a steady state error of zero, even though this component increases the overshoot & the settling time.
* Differential: This is the differential value of the output. In other words, taking the change in error, fast changes (wind gusts) in the error are corrected by this value. This component helps to decrease the overshoot & the settling time.

The equation governing the PID controller is as below:

(1.6)

Where is a generic controlled variable, is is the error between the “Setpoint” and the “Process output”, is the proportional coefficient, is the integral coefficient and is the derivative coefficient.

The description of the 3 inner control algorithms needed to control the Roll, Pitch & the Yaw are described in the following section

### Roll Control

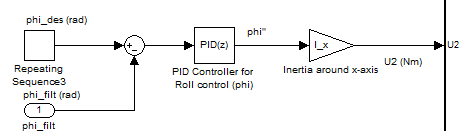


Figure : Block diagram of the Roll Control

*phi\_des [rad]* represents the desired roll angle commanded from the remote controller; *phi\_filt [rad]* represents the filtered roll angle derived out of the “Sensor Filter” block. The difference between these 2 signals gives the error signal for the roll angle which is then given to the PID controller. The output of the controller is a roll angular acceleration () which is then multiplied by the body moment of inertia around the x-axis *[Nms2]* to get the required roll torque *[Nm]*. This inertia comes from the equation (1.4) & is needed to relate the roll control to.

*Kp\_phi [*1/s2*], Ki\_phi [1/s3] & Kd\_phi [1/s]* represents the control parameters for the roll control.

### Pitch Control

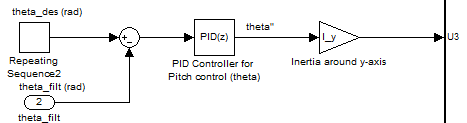


Figure : Block diagram of the Pitch Control

*theta\_des [rad]* represents the desired pitch angle commanded from the remote controller; *theta\_filt [rad]* represents the filtered pitch angle derived out of the “Sensor Filter” block. The difference between these 2 signals gives the error signal for the pitch angle which is then given to the PID controller. The output of the controller is a pitch angular acceleration () which is then multiplied by the body moment of inertia around the y-axis *[Nms2]* to get the required pitch torque *[Nm]*. This inertia comes from the equation (1.4) & is needed to relate the pitch control to.

*Kp\_theta [*1/s2*], Ki\_theta [1/s3] & Kd\_theta [1/s]* represents the control parameters for the pitch control.

### Yaw Rate Control

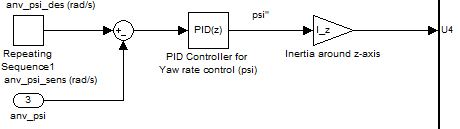


Figure : Block diagram of the Yaw Rate Control

*anv\_psi\_des [rad/s]* represents the desired yaw rate commanded from the remote controller; *anv\_psi\_sens [rad/s]* represents the actual yaw rate measured by the gyroscope. The difference between these 2 signals gives the error signal for the yaw rate which is then given to the PID controller. The output of the controller is a yaw angular acceleration () which is then multiplied by the body moment of inertia around the z-axis *[Nms2]* to get the required yaw torque *[Nm]*. This inertia comes from the equation (1.4) & is needed to relate the yaw rate control to.

*Kp\_anv\_psi [*1/s*], Ki\_anv\_psi [1/s2] & Kd\_anv\_psi [-]* represents the control parameters for the yaw rate control.

# Sensor Filtering

Before explaining what “Sensor Filtering” means, let us look into sensors we use & what values we get from these sensors. Currently in our project we use 1 Accelerometer (LIS3LV02DQ) which is capable of measuring the acceleration in all the axes (X-, Y- & Z- axis) & 3 Gyros (ADXRS610) which is capable of measuring the angular rates (angular velocities Roll, Pitch & Yaw).

One Accelerometer & three Gyros, together form what is known as “Inertial Measuring Unit (IMU)”. The purpose of the IMU is to constantly keep track of the position and angle of the quadrocopter.

But unfortunately the Gyros suffer from what is known as “drifts” i.e. over long term these Gyros will not provide correct angular rate signals. They are prone to noise & produce incorrect results over a period of time. To overcome this problem, acceleration signals from Accelerometer is combined with the angular rate signals from the Gyros. The Gyros will provide very good results for the short term and the long-term drift of the Gyros is eliminated by the Accelerometer.

With this background, we are now in a position to define what “Sensor Filtering” means. It simply means combining 2 different measurement sources to “estimate” one variable by appropriately choosing “weighting factors” for the 2 different sources. This is realized in what is known as “Complementary Filter” or “Balance Filter”.

## Complementary Filter

Often, there are cases where we have two different measurement sources for estimating one variable and the noise properties of the two measurements are such that one source gives good information only in low frequency region while the other is good only in high frequency region.

A “Complementary Filter” is a simple method for integrating the Accelerometer & Gyros measurements for achieving a balanced platform.

Hence it is sometimes also known as “Balanced Filter”.

### To calculate the angle from x-axis Accelerometer Value

y-axis

(Ref. platform)

x-axis

Figure : Quadrocopter lying on the reference platform

If the x-axis of the quadrocopter lies exactly on the Reference platform, then the value read by the Accelerometer (x-axis) = 0 g.

y-axis

(Ref. platform)

a

θ b

c x-axis

Figure : Quadrocopter tilted w.r.t the reference platform

where c is 1g

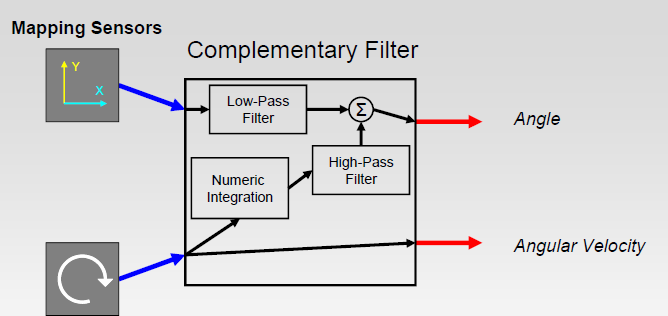
b is a component of 1g (i.e. acceleration value around x-axis from

the sensor)

θ is the tilt angle in radians

For small angles we can approximate.

Hence,

Figure : Fusion of Accelerometer & Gyro to get the filtered angle estimate

With this arrangement, it is now easier to calculate the angle estimate by this simple formula which is depicted in the figure shown above

(1.7)

The 3 essential parts of the Complementary Filter are the “Numeric Integration”, “Low-Pass Filter” & the “High-Pass Filter”.

* Numerical Integration: Gyros give the angular rates (or velocities). To get the angle information at each sample time, we take previous information & then add the change in angle to get the new information.

For the balancing platform this is given by

(1.7a)

* Low-Pass filter: This part of the Complementary filter ensures that only the long-term changes are passed through it by filtering out the short term fluctuations.

(1.7b)

* High-Pass filter: This does exactly the opposite to the Low-Pass filter i.e. it allows short-duration signals to pass through while filtering out signals that are steady over time. This can be used to cancel out drift.

(1.7c)

The above 3 equations (1.7a), (1.7b), (1.7c) are combined to get the complete equation for the Complementary filter.

Besides these 3 parts, 2 other relatively important parts influence the response from the Complementary filter – “Sample Time” & “Time Constant”.

* Sample Time: This is the amount of time between each successive program loop execution of the control algorithm. If the sample rate is 100Hz, the sample time is 10ms.

Time Constant: The time constant of a filter is the relative duration of signal it will act on. For a low-pass filter, signals much longer than the time constant pass through unaltered while signals shorter than the time constant are filtered out. The opposite is true for a high-pass filter. The time constant,, of a digital low-pass filter,

y = (a) \* (y) + (1-a) \* (x),

running in a loop with sample period, , can be found like this:

# Model Transformation into Embedded version Realization

Until now we have described the mathematical equations & system blocks as shown in () in terms of the Simulink® block diagrams. Since we do not intend to generate the code from the Matlab automatically (since it would be little difficult to comprehend its syntaxes & semantics from the auto code!), we plan to manually hand-code the function blocks that were built in Simulink®. For that we need to transform these blocks into a suitable embedded version with the appropriate discretization applied to the signals, so that we get a proper integer representation of all the signals with reasonable resolutions (accuracies). This is realized from the Matlab’s S-Functions feature.

## Brief Info about S-Functions

An S-function is a computer language description of a dynamic system. S-functions can be written using MATLAB or C. C language S-functions are compiled as MEX-files using the *mex* utility or as DLL files using *Visual Studio*. As with other MEX-files, they are dynamically linked into MATLAB when needed. S-functions use a special calling syntax that enables us to interact with Simulink®’s equation solvers. This interaction is very similar to the interaction that takes place between the solvers and built-in Simulink® blocks.

The form of an S-function is very general and can accommodate continuous, discrete, and hybrid systems. As a result, nearly all Simulink® models can be described as S-functions.

MATLAB S-functions are an effective way to embed object code into a Simulink® model. These S functions can be written in a few languages, C being the one most relevant for our purposes.

### Simulation Stages and S-Function Routines

Simulink® makes repeated calls during specific stages of simulation to each block in the model, directing it to perform tasks such as computing its outputs, updating its discrete states, or computing its derivatives. Additional calls are made at the beginning and end of a simulation to perform initialization and termination tasks.

The figure below illustrates how Simulink® performs a simulation. First,

Simulink® initializes the model; this includes initializing each block, including S-functions. Then Simulink® enters the simulation loop, where each pass through the loop is referred to as a simulation step. During each simulation step, Simulink® executes our S-function block. This continues until the simulation is complete:

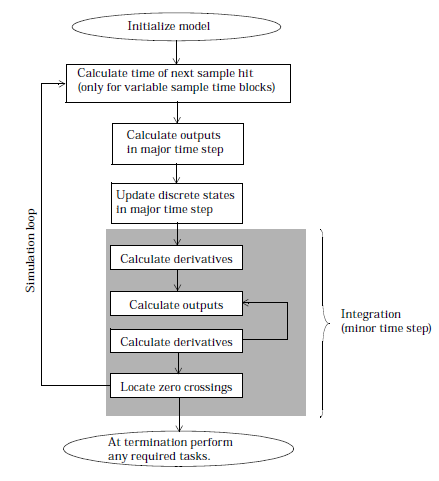


Figure : How Simulink® Performs Simulation

Simulink® makes repeated calls to S-functions in the model. During these

calls, Simulink® calls S-function routines (also called methods), which perform tasks required at each stage. These tasks include:

* Initialization — Prior to the first simulation loop, Simulink® initializes the S-function. During this stage, Simulink®:
* Initializes the *SimStruct*, a simulation structure that contains information about the S-function
* Sets the number and size of input and output ports
* Sets the block sample time(s)
* Allocates storage areas and the sizes array
* Calculation of next sample hit — If you’ve selected a variable step

integration routine, this stage calculates the time of the next variable hit, that is, it calculates the next step size

* Calculation of outputs in the major time step — After this call is complete, all the output ports of the blocks are valid for the current time step
* Update discrete states in the major time step—In this call, all blocks should perform once-per-time-step activities such as updating discrete states for next time around the simulation loop
* Integration — This applies to models with continuous states and/or non-sampled zero crossings. If the S-function has continuous states, Simulink® calls the output and derivative portions of the S-function at minor time steps. This is so Simulink® can compute the state(s) for the S-function. If the S-function (C MEX only) has non-sampled zero crossings, then Simulink® will call the output and zero crossings portion of the S-function at minor time steps, so that it can locate the zero crossings

## Implemented S-Function blocks of Quadrocopter

Following S-Function blocks are implemented in Simulink®’s S-Function (C-Code).

* Quadrocopter Controller
* Sensor Filter

### S-Function block Quadrocopter Controller

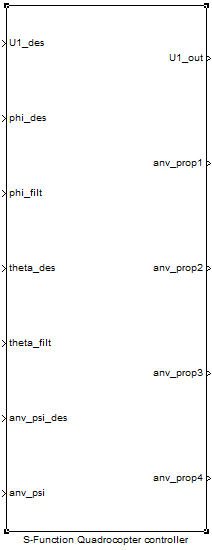


Figure : S-Function Quadrocopter controller

Description of the signals:

#### Inputs:

U1\_des: Desired total force commanded from the remote controller

phi\_des: Desired roll angle commanded from the remote controller

phi\_filt: Filtered roll angle as indicated by the sensors (accelerometer & gyro)

theta\_des: Desired pitch angle commanded from the remote controller

theta\_filt: Filtered pitch angle as indicated by the sensors (accelerometer & gyro)

anv\_psi\_des: Desired yaw velocity commanded from the remote controller

anv\_psi: yaw velocity as indicated by the sensor (gyro)

#### Outputs:

U1\_out: Total force output from all the propellers

anv\_prop1: Speed of propeller 1 (Front propeller)

anv\_prop2: Speed of propeller 2 (Left propeller)

anv\_prop3: Speed of propeller 3 (Rear propeller)

anv\_prop4: Speed of propeller 4 (Right propeller)

### S-Function block Sensor Filter

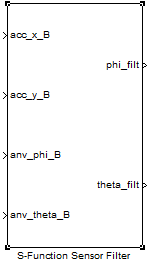


Figure : S-Function Sensor Filter

Description of the signals:

#### Inputs:

acc\_x\_B: Acceleration of the Body frame in the X direction as indicated by the sensor (accelerometer)

acc\_y\_B: Acceleration of the Body frame in the Y direction as indicated by the sensor (accelerometer)

anv\_phi\_B: Roll velocity as indicated by the sensor (gyro)

anv\_theta\_B: Pitch velocity as indicated by the sensor (gyro)

#### Outputs:

phi\_filt: Filtered roll angle as indicated by the sensors (accelerometer & gyro)

theta\_filt: Filtered pitch angle as indicated by the sensors (accelerometer & gyro)

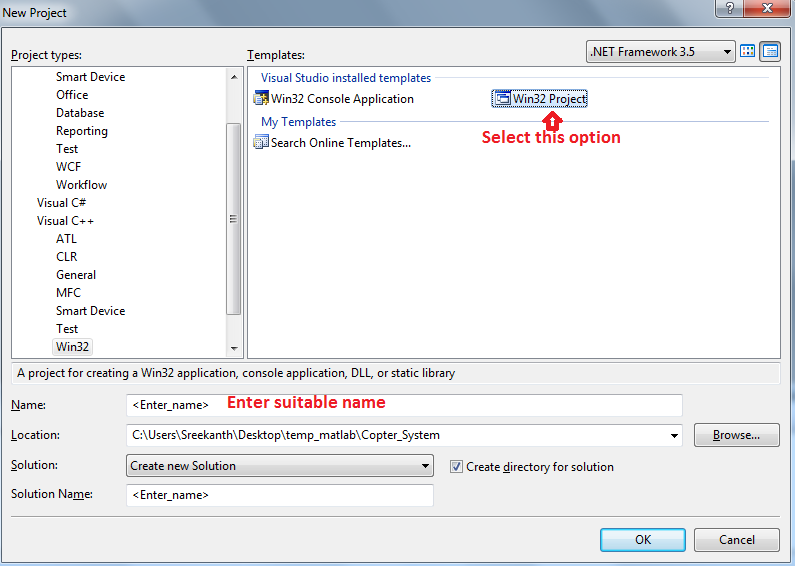
For more details on these subsystems, please see Appendix A.

## *Visual Studio* settings for compiling S-Functions

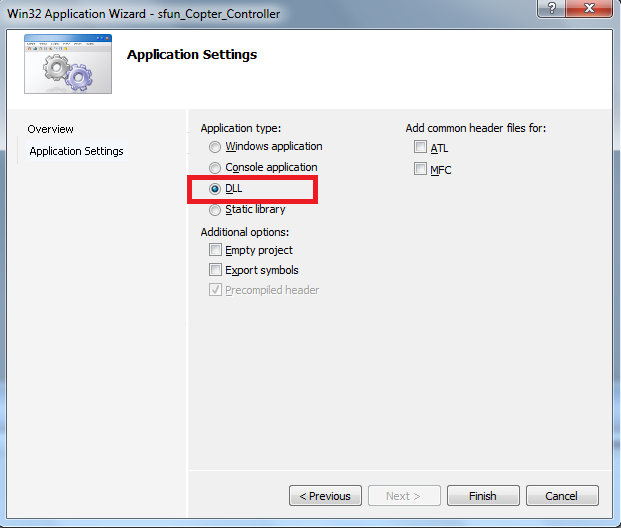
S-Functions are written & compiled into .dll files by Microsoft® Visual Studio, since it offers the possibility to debug the code by interfacing it with the Matlab Simulink®.

Here are the general settings to create the Visual Studio project & to compile the source code into .dll files.

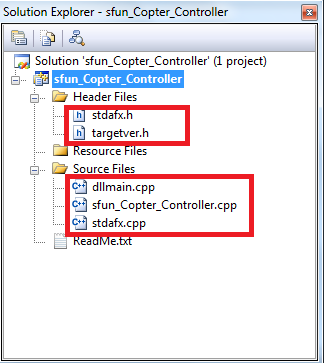
1. Create a new Win32 application & suitably name the project.



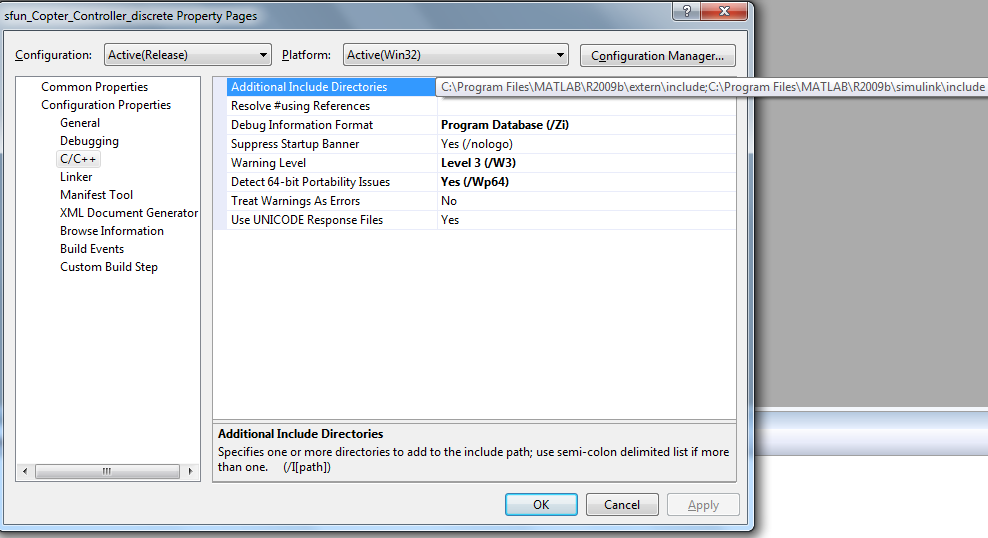
1. Choose the “Application Type” as DLL.



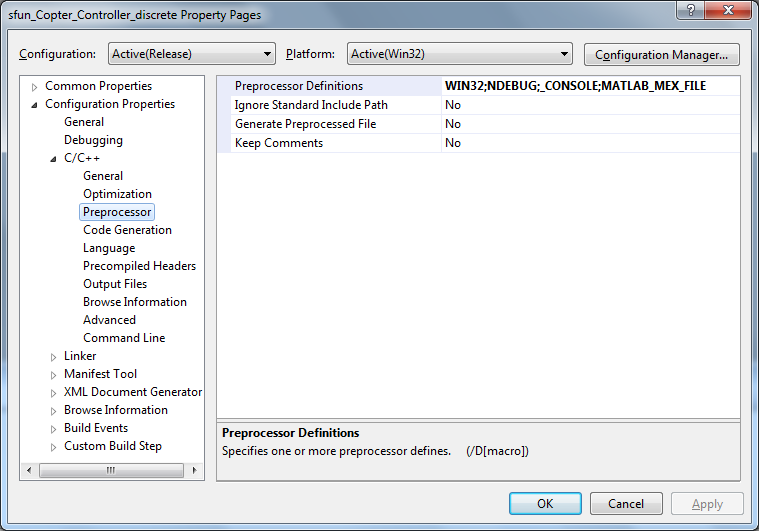
1. Remove these files & add your own source & header files for compilation.



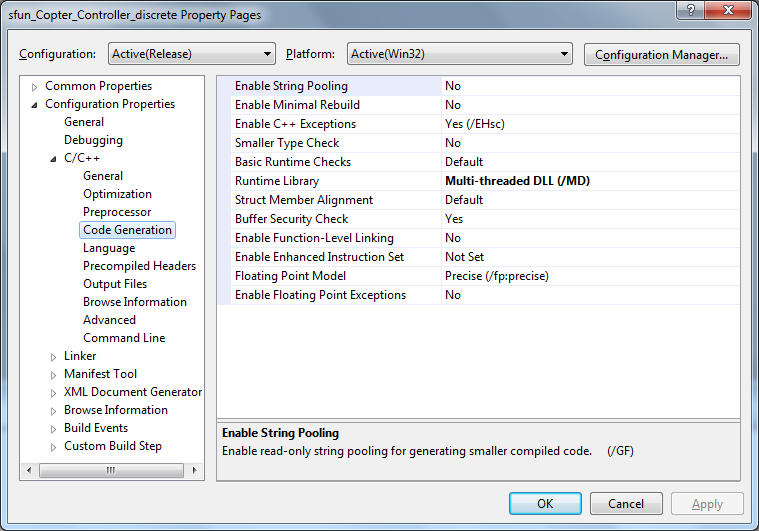
1. Select the highlighted options for the field “C/C++".



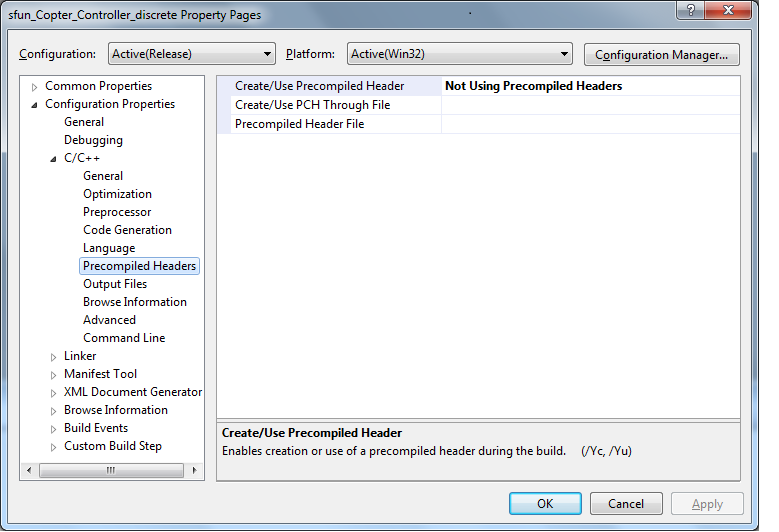
1. Select the highlighted options for the field “Preprocessor".



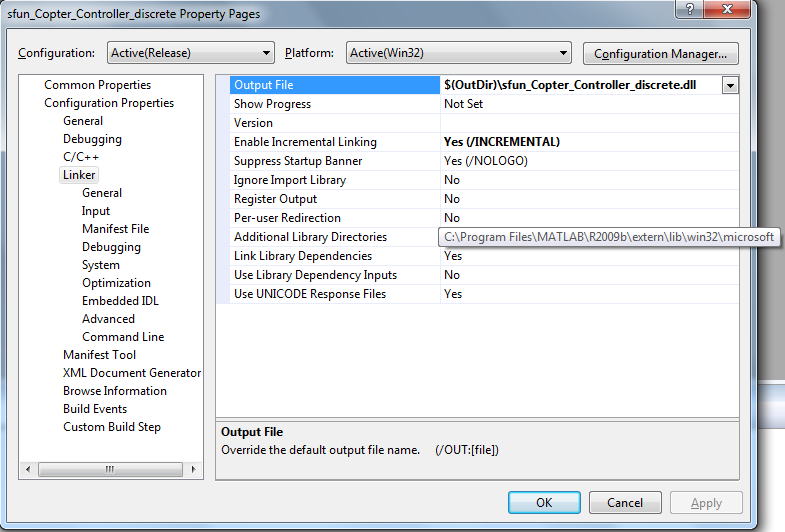
1. Select the highlighted options for the field “Code Generation".



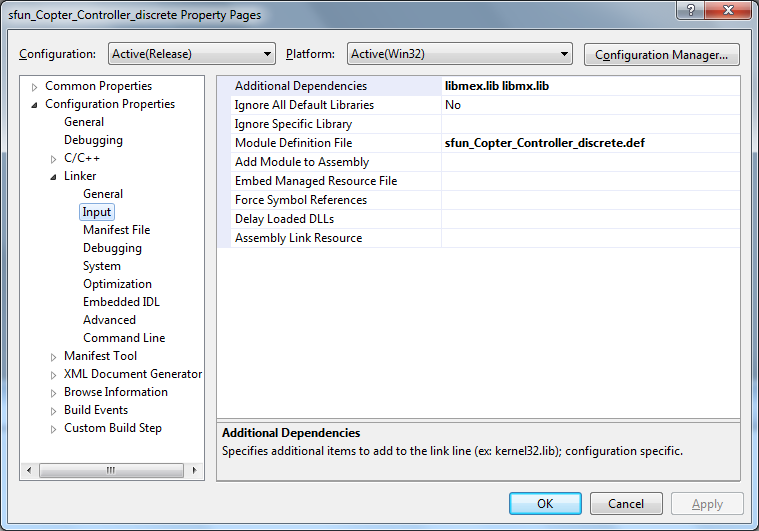
1. Select the highlighted options for the field “Precompiled Headers".



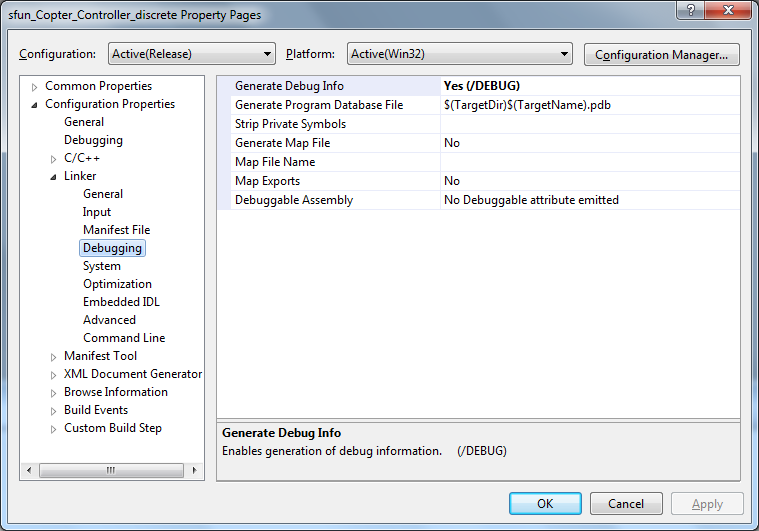
1. Select the highlighted options for the field “Linker".



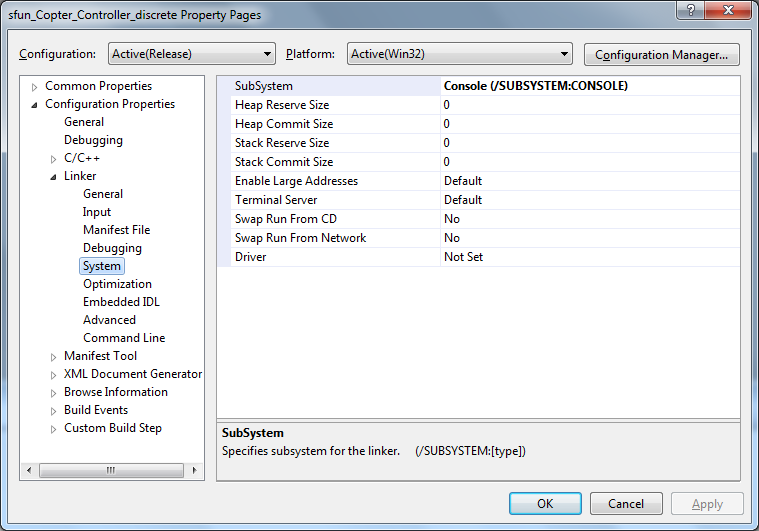
1. Select the highlighted options for the field “Input".



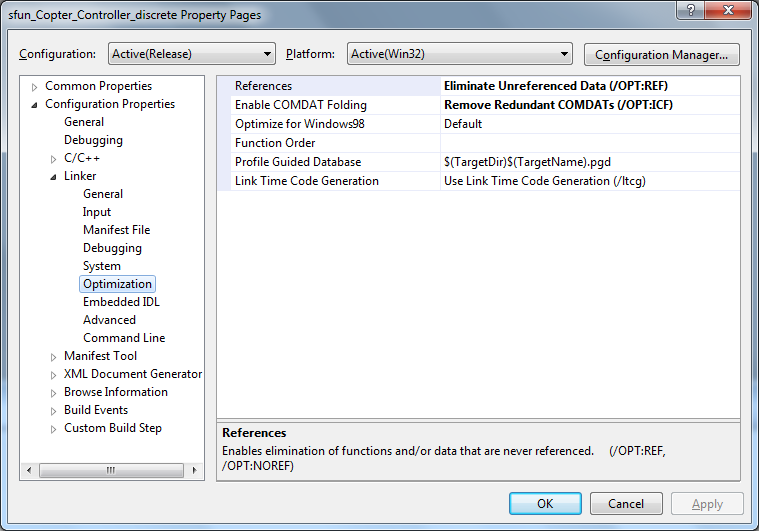
1. Select the highlighted options for the field “Debugging".



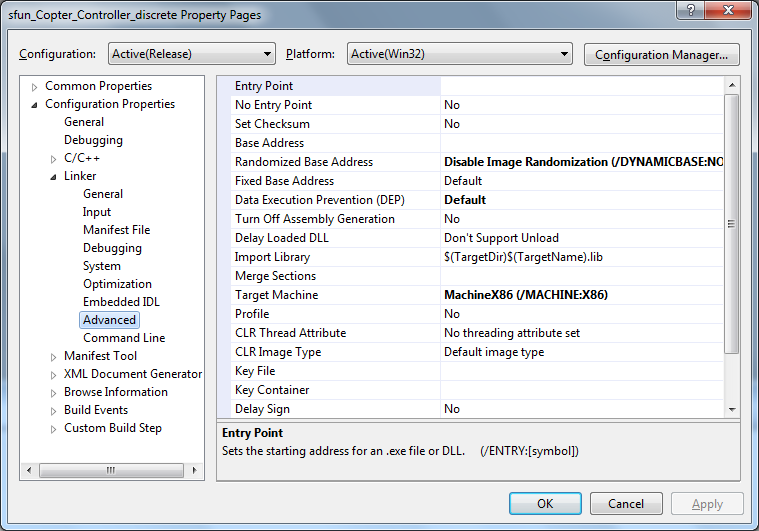
1. Select the highlighted options for the field “System".



1. Select the highlighted options for the field “Optimization".



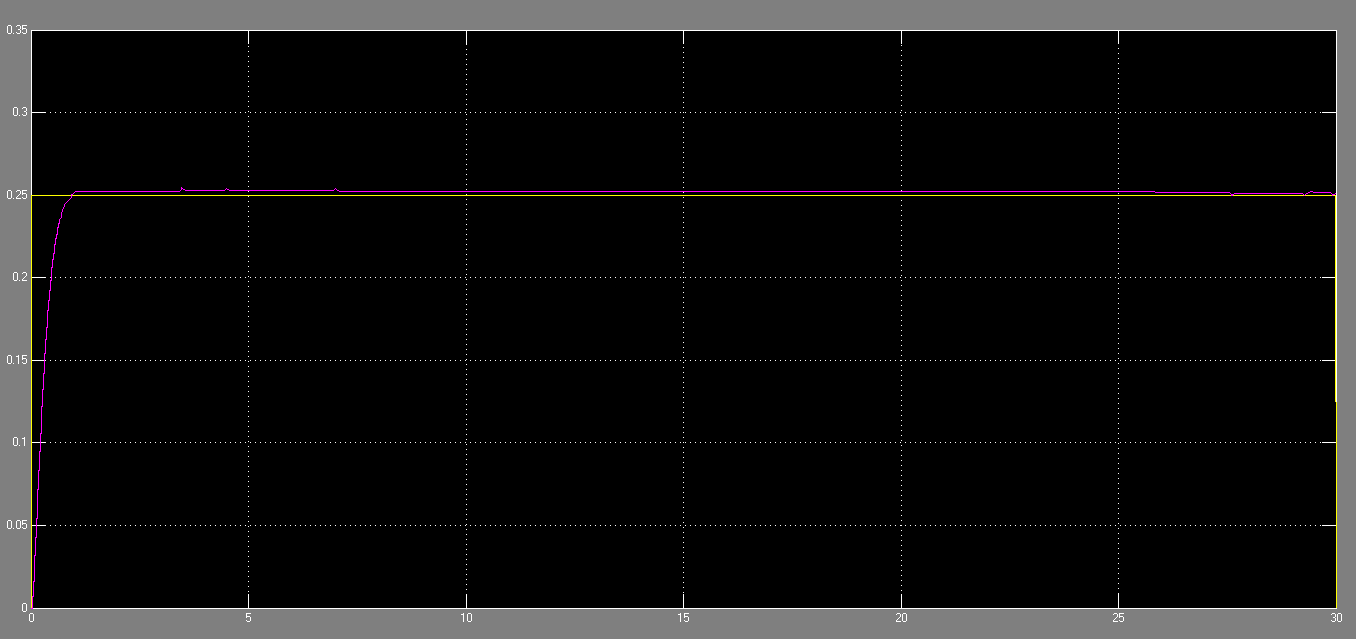
1. Select the highlighted options for the field “Advanced".



# Test cases & Simulation results

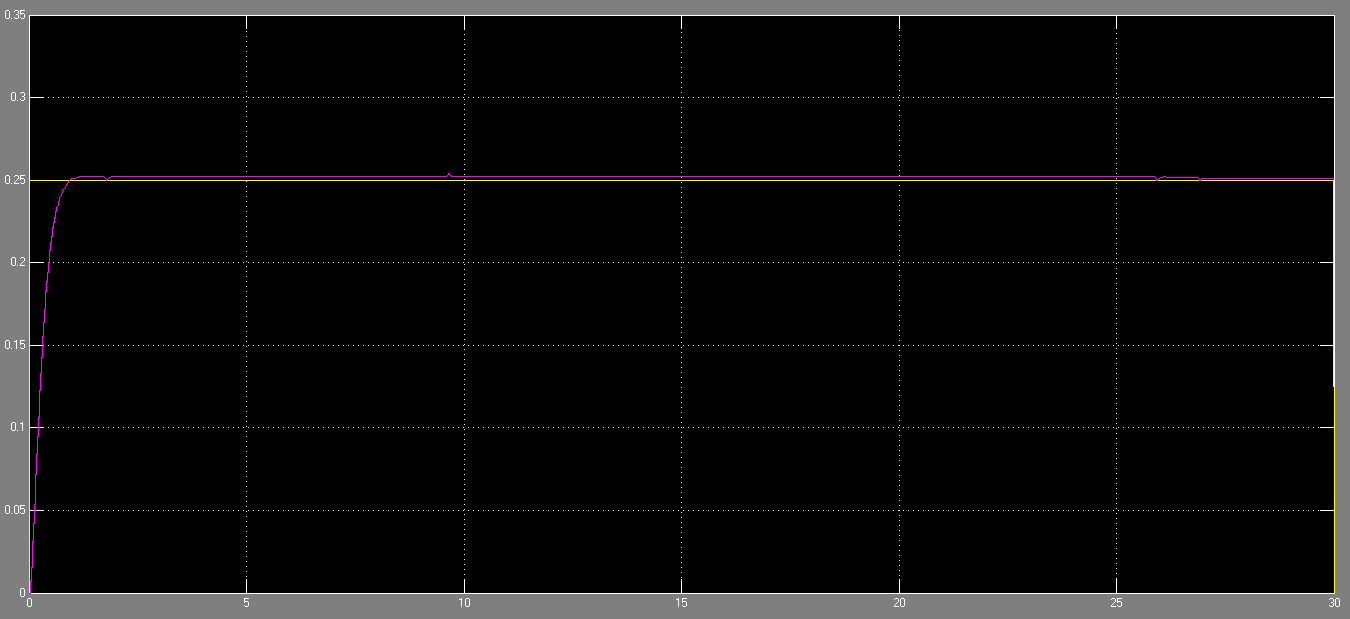
1. Step Roll angle simulation

Desired Roll angle = 0 to 0.25 rad step input in 0.01 sec



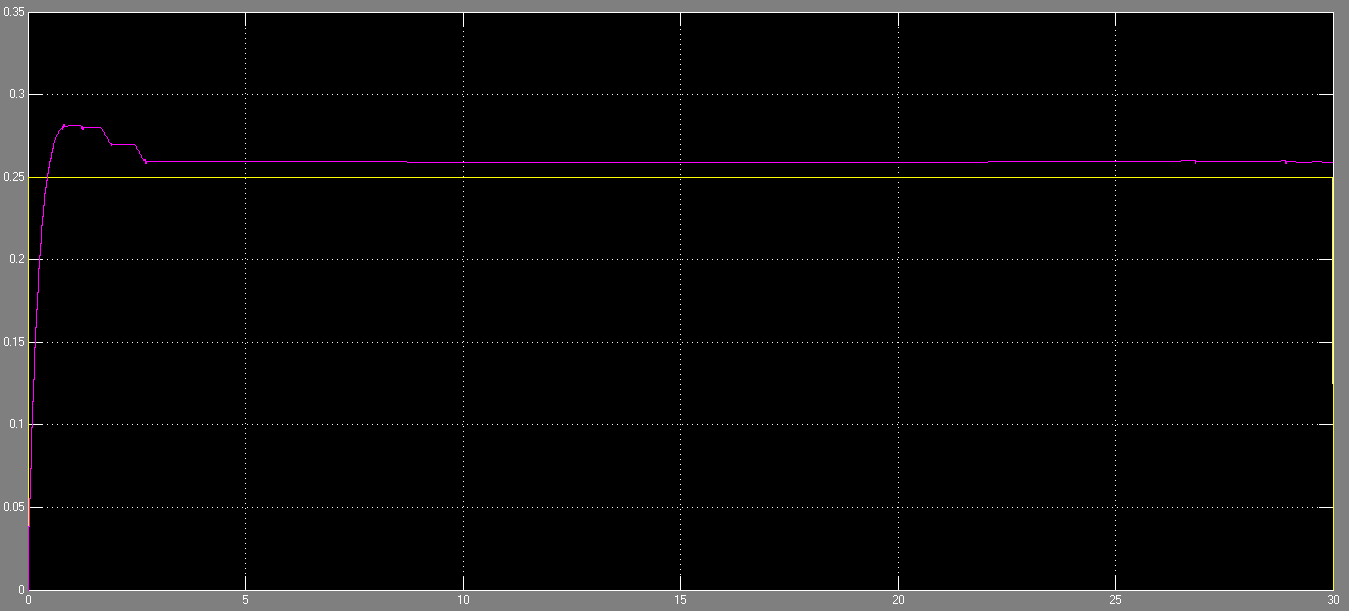
1. Step Pitch angle simulation

Desired Pitch angle = 0 to 0.25 rad step input in 0.01 sec



1. Step yaw velocity simulation

Desired yaw velocity = 0 to 0.25 rad/s step input in 0.01 sec



# Improvements & Conclusions

The main goal of the project was to have a stable flight while hovering as well as during quick maneuvers. Hence we have chosen the PID controllers for the Roll angle, Pitch angle & Yaw velocity stabilization. Even though the simulations showed satisfactory results with this type of controller, final result was not robust enough & the actual flight was not fully autonomous. Possible reasons could be,

* The simulated Quadrotor model might be insufficient in addressing all the non-linearities (for eg: taking into account external disturbances like wind influences…)
* There was no model to account for the temperature compensations for the sensors (accelerometer & gyros)
* To some extent we feel the Sensor filter block implemented with “Complementary Filter” might not give a good estimate of the angles. Although one could argue that we can get reasonable values from such a filtering estimate, one could use more sophisticated estimating techniques like “Kalman Filter” instead of “Complementary Filter”. But the mathematical overhead of such complicated estimating techniques sometimes far outweigh the actual benefits achieved from such methods.

Hence we feel the probable improvements would be in the area of accurate Quadrotor dynamics modeling, designing the compensation blocks for the sensors (temperature) & to some extent incorporate a better filtering technique to get the angle estimates.

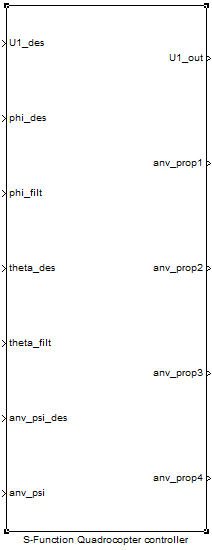
# Appendix A: Subsystem Interfaces

This section gives an overview of the various interfaces (like Signal, Type, Resolution, Unit) between the subsystems of Copter system as implemented in the MATLAB/Simulink®.

As mentioned in the chapter 3.2, following subsystems are implemented in Simulink®’s S-Function (C-Code).

* Quadrocopter Controller
* Sensor Filter

## A 1 Subsystem Quadrocopter Controller



### A 1.1 Inputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** |
| U1\_des | int16 | 0.001 | N |
| phi\_des | int16 | 0.001 | rad |
| phi\_filt | int16 | 0.001 | rad |
| theta\_des | int16 | 0.001 | rad |
| theta\_filt | int16 | 0.001 | rad |
| anv\_psi\_des | int16 | 0.001 | rad/s |
| anv\_psi | int16 | 0.001 | rad/s |

### 

### A 1.2 Outputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** |
| U1\_out | int16 | 0.001 | N |
| anv\_prop1 | int16 | 1 | rpm |
| anv\_prop2 | int16 | 1 | rpm |
| anv\_prop3 | int16 | 1 | rpm |
| anv\_prop4 | int16 | 1 | rpm |

### 

### A 1.3 Parameters

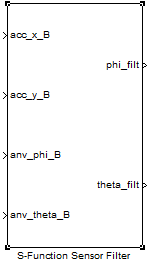
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** | **Default Value** |
| Kp\_phi | int16 | 0.01 | 1/s2 | 20.00 |
| Ki\_phi | int16 | 0.01 | 1/s3 | 5.00 |
| Kd\_phi | int16 | 0.01 | 1/s | 3.00 |
| Kp\_theta | int16 | 0.01 | 1/s2 | 20.00 |
| Ki\_theta | int16 | 0.01 | 1/s3 | 5.00 |
| Kd\_theta | int16 | 0.01 | 1/s | 3.00 |
| Kp\_anv\_psi | int16 | 0.01 | 1/s | 20 |
| Ki\_anv\_psi | int16 | 0.01 | 1/s2 | 2 |
| Kd\_anv\_psi | int16 | 0.01 | - | 2 |
| I\_x | int16 | 0.001 | Nms2 | 0.009 |
| I\_y | int16 | 0.001 | Nms2 | 0.009 |
| I\_z | int16 | 0.001 | Nms2 | 0.016 |
| l\_boom | int16 | 0.001 | m | 0.166 |

### A 1.4 Constants

|  |  |
| --- | --- |
| **Signal** | **Value** |
| S\_FUNCTION\_NAME | sfun\_Copter\_Controller\_discrete |
| S\_FUNCTION\_LEVEL | 2 |
| NUMINPORTS | 7 |
| NUMOUTPORTS | 5 |
| NUMPARAMS | 1 |
| THRUST2RPM\_DATAPOINTS | 10 |

## 

## A 2 Subsystem Sensor Filter



### A 2.1 Inputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** |
| acc\_x\_B | int16 | 0.1 | m/s2 |
| acc\_y\_B | int16 | 0.1 | m/s2 |
| anv\_phi\_B | int16 | 0.01 | rad/s |
| anv\_theta\_B | int16 | 0.01 | rad/s |

### A 2.2 Outputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** |
| phi\_filt | int16 | 0.0001 | rad |
| theta\_filt | int16 | 0.0001 | rad |

### A 2.3 Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal** | **Type** | **Resolution** | **Unit** | **Default Value** |
| LPFP | int16 | 0.01 | - | 0.05 |
| HPFP | int16 | 0.01 | - | 0.95 |
| LPFR | int16 | 0.01 | - | 0.05 |
| HPFR | int16 | 0.01 | - | 0.95 |

### A 2.4 Constants

|  |  |
| --- | --- |
| **Signal** | **Value** |
| S\_FUNCTION\_NAME | sfun\_Sensor\_Filter\_discrete |
| S\_FUNCTION\_LEVEL | 2 |
| NUMINPORTS | 4 |
| NUMOUTPORTS | 2 |
| NUMPARAMS | 1 |

# References

1. *“Design and Control of an Indoor Micro Quadrotor”,* by Samir Bouabdallah, Pierpaolo Murrieri, Roland Siegwart
2. *“Modelling, Identification and Control of a Quadrotor Helicopter”,* by Tommaso Bresciani, Oct 2008
3. *“Design of a Four-Rotor Aerial Robot”,* by P.Pounds, R.Mahony, P.Hynes & J.Roberts, Proc.2002, Australasian Conference on Robotics & Automation, Auckland, 27-29 November 2002
4. *“Dynamic modeling and nonlinear control strategy for an under actuated quad rotor rotorcraft”,* by Ashfaq Ahmad Mian, Dao-bo Wang, Journal of Zhejiang University SCIENCE A, 2008 9(4):539-545
5. *“Modeling, Simulation and Flight Testing of an Autonomous Quadrotor”,* byRahul Goel, Sapan M. Shah, Nitin K. Gupta, N. Ananthkrishnan, Proceedings of ICEAE 2009
6. *“Modelling and Control of a Quad-Rotor Robot”,* by Paul Pounds, Robert Mahony, Peter Corke
7. *“Modelling the Dynamics of Flight Control Surfaces Under Actuation Compliances and Losses”,* by Ashok Joshi
8. *“The Balance Filter - A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform”,* by Shane Colton, Rev.1: Submitted as a Chief Delphi white paper - June 25, 2007
9. *“An Introduction to the Kalman Filter”,* by Greg Welch and Gary Bishop, UNC-Chapel Hill, TR 95-041, July 24, 2006
10. *“Estimation of attitudes from a low-cost miniaturized inertial platform using Kalman Filter-based sensor fusion algorithm”, by* N Shanta Kumar and T Jann, Sadhana Vol. 29, Part 2, April 2004, pp. 217–235
11. *“Simulink® 7 Writing S-Functions”,* © COPYRIGHT 1998–2007 by The MathWorks, Inc
12. *“http://www.mikrokopter.de/”*
13. *“http://www.jeroenhermans.nl/daedalus”*