

# FARGNC

F A L C O - 4

Fly-A-Rocket (G)uidance (N)avigation and (C)ontrol Software

## Authors

COLLAUD Xavier  
GASPAR Marting Mikael  
MALYUTA Danylo  
MASLOV John  
MULLIN Nikolay  
PICTET Raimondo  
ROUAZE Gautier

Project made possible thanks to:



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE



Generated by Doxygen 1.8.6

Sun May 31 2015



# Contents

<b>1</b>	<b>Project description</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Contributors . . . . .	1
1.3	License . . . . .	2
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	bcm2835_peripheral Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.2	Control_loop Struct Reference . . . . .	7
4.2.1	Detailed Description . . . . .	8
4.3	MATRIX Struct Reference . . . . .	8
4.3.1	Detailed Description . . . . .	8
4.4	SPI_data Struct Reference . . . . .	8
4.4.1	Detailed Description . . . . .	9
<b>5</b>	<b>File Documentation</b>	<b>11</b>
5.1	control_funcs.c File Reference . . . . .	11
5.1.1	Detailed Description . . . . .	11
5.1.2	Function Documentation . . . . .	11
5.1.2.1	Fpitch_loop_control_setup . . . . .	11
5.1.2.2	Fyaw_loop_control_setup . . . . .	12
5.1.2.3	Mroll_loop_control_setup . . . . .	12
5.1.3	Variable Documentation . . . . .	12
5.1.3.1	VALVE__MAX_THRUST . . . . .	12
5.2	control_header.h File Reference . . . . .	12
5.2.1	Detailed Description . . . . .	12
5.3	imu_funcs.c File Reference . . . . .	13

5.3.1	Detailed Description	14
5.3.2	Function Documentation	14
5.3.2.1	Calibrate_IMU	14
5.3.2.2	close_port	14
5.3.2.3	construct_zeroed_DCM	14
5.3.2.4	Find_raw_Euler_angular_velocities	15
5.3.2.5	get_filtered_attitude_parallel	15
5.3.2.6	get_old_attr	15
5.3.2.7	Kalman_filter	15
5.3.2.8	min_of_set	15
5.3.2.9	open_serial_port	16
5.3.2.10	read_IMU_parallel	16
5.3.2.11	reset_old_attr_port	16
5.3.2.12	set_new_attr	16
5.3.2.13	set_to_blocking	17
5.3.2.14	TO_DEG	17
5.3.2.15	Treat_reply	17
5.3.2.16	zero_Euler_angles	17
5.4	imu_header.h File Reference	17
5.4.1	Detailed Description	20
5.5	la_funcs.c File Reference	20
5.5.1	Detailed Description	21
5.5.2	Function Documentation	21
5.5.2.1	initMatrix	21
5.5.2.2	madd	21
5.5.2.3	minverse_1x1	21
5.5.2.4	mmultiply	21
5.5.2.5	msubtract	22
5.5.2.6	transpose	22
5.6	la_header.h File Reference	22
5.6.1	Detailed Description	22
5.7	master.c File Reference	22
5.7.1	Detailed Description	25
5.7.2	Function Documentation	25
5.7.2.1	main	25
5.7.3	Variable Documentation	25
5.7.3.1	gpio	25
5.8	master_funcs.c File Reference	26
5.8.1	Detailed Description	26
5.8.2	Function Documentation	26

5.8.2.1	<a href="#">check_time</a>	26
5.8.2.2	<a href="#">open_error_file</a>	27
5.8.2.3	<a href="#">open_file</a>	27
5.8.2.4	<a href="#">passive_wait</a>	27
5.8.2.5	<a href="#">write_to_file_custom</a>	27
5.9	<a href="#">master_header.h File Reference</a>	28
5.9.1	<a href="#">Detailed Description</a>	32
5.9.2	<a href="#">Variable Documentation</a>	32
5.9.2.1	<a href="#">VALVE__MAX_THRUST</a>	32
5.10	<a href="#">msp430_funcs.c File Reference</a>	32
5.10.1	<a href="#">Detailed Description</a>	32
5.10.2	<a href="#">Function Documentation</a>	33
5.10.2.1	<a href="#">MSP430_UART_receive</a>	33
5.10.2.2	<a href="#">MSP430_UART_write</a>	33
5.10.2.3	<a href="#">MSP430_UART_write_PWM</a>	33
5.11	<a href="#">msp430_header.h File Reference</a>	34
5.11.1	<a href="#">Detailed Description</a>	34
5.12	<a href="#">pressure_funcs.c File Reference</a>	35
5.12.1	<a href="#">Detailed Description</a>	35
5.12.2	<a href="#">Function Documentation</a>	36
5.12.2.1	<a href="#">get_readings_SPI_parallel</a>	36
5.12.2.2	<a href="#">pressure_sensor_SPI_connect</a>	37
5.13	<a href="#">pressure_header.h File Reference</a>	37
5.13.1	<a href="#">Detailed Description</a>	38
5.14	<a href="#">rpi_gpio_funcs.c File Reference</a>	38
5.14.1	<a href="#">Detailed Description</a>	38
5.14.2	<a href="#">Function Documentation</a>	38
5.14.2.1	<a href="#">map_peripheral</a>	38
5.14.2.2	<a href="#">unmap_peripheral</a>	39
5.15	<a href="#">rpi_gpio_header.h File Reference</a>	39
5.15.1	<a href="#">Detailed Description</a>	39
5.15.2	<a href="#">Variable Documentation</a>	40
5.15.2.1	<a href="#">gpio</a>	40
5.16	<a href="#">simplex_funcs.c File Reference</a>	40
5.16.1	<a href="#">Detailed Description</a>	40
5.16.2	<a href="#">Function Documentation</a>	41
5.16.2.1	<a href="#">get_simplex_solution</a>	41
5.16.2.2	<a href="#">simp1</a>	41
5.16.2.3	<a href="#">simp2</a>	41
5.16.2.4	<a href="#">simp3</a>	41

---

5.16.2.5	<a href="#">simplex</a>	41
5.17	<a href="#">simplex_header.h File Reference</a>	42
5.17.1	<a href="#">Detailed Description</a>	43
5.18	<a href="#">spycam_funcs.c File Reference</a>	43
5.18.1	<a href="#">Detailed Description</a>	43
5.18.2	<a href="#">Function Documentation</a>	44
5.18.2.1	<a href="#">startVideo</a>	44
5.18.2.2	<a href="#">stopVideo</a>	44
5.19	<a href="#">spycam_header.h File Reference</a>	44
5.19.1	<a href="#">Detailed Description</a>	44
<b>Index</b>		<b>45</b>

# Chapter 1

## Project description

### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

### Version

1.0

## 1.1 Introduction

This project presents the cumulative effort of an Undergraduate student team to design, build and fly the first ever recorded attempt to actively stabilize a model rocket with an (R)action (C)ontrol (S)ystem. The end goal is to have 3 successful demonstration flights with the RCS control activated that show the rocket being maintained in the upright vertical orientation during the few seconds of low velocity around apogee. The testbed vehicle, the (F)light (A)ttitude (L)inearly (C)ontrolled (4)th iteration (FALCO-4) rocket, is the own design of the student team and has been made specifically to carry the RCS system and supporting avionics at a minimum size and cost.

## 1.2 Contributors

Contributors:

- Danylo Malyuta (GNC and avionics)
- Gautier Rouaze (RCS mechanical design)
- Xavier Collaud (Rocket airframe design)
- Nikolay Mullin (Rocket design and systems architecture supervisor)
- Mikael Gaspar (Launch systems and ground support)
- Raimondo Pictet (CFD analysis)
- John Maslov (GPS tracking)

Funding : eSpace Space Engineering Center EPFL.

A special thanks goes to the following people and companies for their support:

- Jürg Thüring from SpaceTec Rocketry, Tripoli and ARGOS (Advanced Rocket Group of Switzerland) for launch advice

- eSpace Space Engineer Center for financial and avionics support
- EPFL Swiss Institute of Technology in Lausanne
- LMAF laboratory at EPFL for manufacturing facilities
- Element AG for nose cone mold creation
- Loxam for portable electrical generator lease

### 1.3 License

The MIT License (MIT)

Copyright (c) 2015 Danylo Malyuta

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">bcm2835_peripheral</a>	7
<a href="#">Control_loop</a>	7
<a href="#">MATRIX</a>	8
<a href="#">SPI_data</a>	8



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">control_funcs.c</a>	
Control functions source file . . . . .	11
<a href="#">control_header.h</a>	
Control header file . . . . .	12
<a href="#">imu_funcs.c</a>	
IMU functions file (contains IMU comms, logs and filters) . . . . .	13
<a href="#">imu_header.h</a>	
IMU header file . . . . .	17
<a href="#">la_funcs.c</a>	
Linear Algebra functions source file . . . . .	20
<a href="#">la_header.h</a>	
Linear Algebra header file . . . . .	22
<a href="#">master.c</a>	
Master GNC source file (main function file) . . . . .	22
<a href="#">master_funcs.c</a>	
Master functions file . . . . .	26
<a href="#">master_header.h</a>	
Master header file . . . . .	28
<a href="#">msp430_funcs.c</a>	
MSP430 functions file . . . . .	32
<a href="#">msp430_header.h</a>	
MSP430 header file . . . . .	34
<a href="#">pressure_funcs.c</a>	
Honeywell pressure/temperature sensors functions file . . . . .	35
<a href="#">pressure_header.h</a>	
Honeywell pressure/temperature sensors header file . . . . .	37
<a href="#">rpi_gpio_funcs.c</a>	
GPIO functions file . . . . .	38
<a href="#">rpi_gpio_header.h</a>	
MSP430 header file . . . . .	39
<a href="#">simplex_funcs.c</a>	
Simplex functions file . . . . .	40
<a href="#">simplex_header.h</a>	
Simplex header file . . . . .	42
<a href="#">spycam_funcs.c</a>	
Raspberry Pi spy camera functions file . . . . .	43
<a href="#">spycam_header.h</a>	
Raspberry Pi spy camera header file . . . . .	44



## Chapter 4

# Class Documentation

### 4.1 bcm2835\_peripheral Struct Reference

```
#include <rpi_gpio_header.h>
```

#### Public Attributes

- unsigned long [addr\\_p](#)  
*Address in physical map that we want this memory block to expose.*
- int [mem\\_fd](#)  
*File descriptor to physical memory virtual file '/dev/mem'.*
- void \* **map**
- volatile unsigned int \* [addr](#)  
*Refers to register address.*

#### 4.1.1 Detailed Description

Holds what's needed to access the Raspberry Pi GPIO port.

The documentation for this struct was generated from the following file:

- [rpi\\_gpio\\_header.h](#)

### 4.2 Control\_loop Struct Reference

```
#include <control_header.h>
```

#### Public Attributes

- float [K](#)  
*Proportional term coefficient.*
- float [Td](#)  
*Derivative term coefficient.*
- float [satur](#)  
*Absolute ceiling of possible control loop output value.*
- float [control\\_range](#)  
*At what angle from the vertical orientation to we begin applying maximum control input?*

### 4.2.1 Detailed Description

This structure holds all of the variables relating to a control loop of the GNC algorithm. Namely, there are three such control loops used:

- For the pitch force, Fpitch
- For the yaw force, Fyaw
- For the roll moment, Mroll

The combination of these three control loops stabilizes the rocket to point vertically up at all time

The documentation for this struct was generated from the following file:

- [control\\_header.h](#)

## 4.3 MATRIX Struct Reference

```
#include <la_header.h>
```

### Public Attributes

- `size_t` [rows](#)  
*Number of rows.*
- `size_t` [cols](#)  
*Number of columns.*
- `float **` [matrix](#)  
*The dynamic 2D array.*

### 4.3.1 Detailed Description

This is the structure for a Matrix.

The documentation for this struct was generated from the following file:

- [la\\_header.h](#)

## 4.4 SPI\_data Struct Reference

```
#include <pressure_header.h>
```

### Public Attributes

- unsigned char [mode](#)  
*SPI mode.*
- unsigned char [bits](#)  
*Number of bits per SPI transmission.*
- unsigned long int [max\\_speed](#)  
*Frequency of SPI transmission (in [Hz])*
- unsigned char [buffer\\_length](#)  
*Length of SPI transmit/receive buffer.*

- unsigned int [P\\_OUT\\_\\_MAX](#)  
*Maximum decimal value received when recording max pressure, refer to [datasheet](#) page 13 for second to last model number ("Transfer Function") given HSC D LN N 100MD S A 5.*
- unsigned int [P\\_OUT\\_\\_MIN](#)  
*Minimum decimal value received when recording min pressure, refer to [datasheet](#) page 13 for second to last model number ("Transfer Function") given HSC D LN N 100MD S A 5.*
- float [P\\_\\_MAX](#)  
*[mbar] maximum sensor pressure reading*
- float [P\\_\\_MIN](#)  
*[mbar] minimum sensor pressure reading*
- unsigned int [radial\\_sensor\\_fd](#)  
*Radial sensor connection handle.*
- unsigned int [axial\\_sensor\\_fd](#)  
*Axial sensor connection handle.*

#### 4.4.1 Detailed Description

This structure contains all info necessary to communicate with and to interpret incoming data from the Honeywell HSC sensors. Please refer to the [datasheet](#) and [Honeywell SPI companion](#) for info on parameters like P\_OUT\_MAX, P\_OUT\_\_MIN, etc. Note that the specific pressure sensors that we use are: HSC D LN N 100MD S A 5.

The documentation for this struct was generated from the following file:

- [pressure\\_header.h](#)





## Chapter 5

# File Documentation

### 5.1 control\_funcs.c File Reference

Control functions source file.

```
#include <math.h>
#include "master_header.h"
#include "control_header.h"
```

#### Functions

- void [Fpitch\\_loop\\_control\\_setup](#) ()
- void [Fyaw\\_loop\\_control\\_setup](#) ()
- void [Mroll\\_loop\\_control\\_setup](#) ()

#### Variables

- float [VALVE\\_\\_MAX\\_THRUST](#) =0.5  
*Maximum thrust of RCS solenoid valves (i.e. when fully opened) // TODO: confirm with Gautier!*

#### 5.1.1 Detailed Description

Control functions source file.

##### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

##### Version

1.0

This file contains functions regarding the control algorithm.

#### 5.1.2 Function Documentation

##### 5.1.2.1 void [Fpitch\\_loop\\_control\\_setup](#) ( )

This function setups up all control parameters relating to the pitch control.

#### 5.1.2.2 void Fyaw\_loop\_control\_setup ( )

This function sets up all control parameters relating to the yaw control.

#### 5.1.2.3 void Mroll\_loop\_control\_setup ( )

This function sets up all control parameters relating to the roll control.

### 5.1.3 Variable Documentation

#### 5.1.3.1 float VALVE\_\_MAX\_THRUST =0.5

Maximum thrust of RCS solenoid valves (i.e. when fully opened) // TODO: confirm with Gautier!

Maximum thrust of RCS solenoid valves (i.e. when fully opened)

## 5.2 control\_header.h File Reference

Control header file.

### Classes

- struct [Control\\_loop](#)

### Variables

#### Control loop group

*These structures define fully the control of the rocket - i.e. the control gains.*

- struct [Control\\_loop](#) Fpitch\_loop  
*Pitch control loop, uses feedback on [theta\\_filt](#) to tell what pitching corrective force we need.*
- struct [Control\\_loop](#) Fyaw\_loop  
*Yaw control loop, uses feedback on [psi\\_filt](#) to tell what yawing corrective force we need.*
- struct [Control\\_loop](#) Mroll\_loop  
*Roll control loop, uses feedback on [phi\\_dot\\_filt](#) to tell what corrective rolling moment we need.*

### 5.2.1 Detailed Description

Control header file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This is the header to [control\\_funcs.c](#) containing necessary definitions and initializations.

## 5.3 imu\_funcs.c File Reference

IMU functions file (contains IMU comms, logs and filters).

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/termios.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <math.h>
#include <pthread.h>
#include "imu_header.h"
#include "master_header.h"
#include "la_header.h"
```

### Functions

- void [open\\_serial\\_port](#) (int \*fd, char \*directory)
- void [close\\_port](#) (int fd)
- void [get\\_old\\_attr](#) (int fd, struct termios \*old\_options)
- void [reset\\_old\\_attr\\_port](#) (int fd, struct termios \*old\_options)
- void [set\\_new\\_attr](#) (int fd, struct termios \*old\_options, struct termios \*new\_options)
- void [set\\_to\\_blocking](#) (int fd)
- float [min\\_of\\_set](#) (float now, float before)
- void [Find\\_raw\\_Euler\\_angular\\_velocities](#) ()
- void [Treat\\_reply](#) (char \*comparison\_string)
- void [construct\\_zeroed\\_DCM](#) ()
- float [TO\\_DEG](#) (float angle)
- void [zero\\_Euler\\_angles](#) ()
- void [Calibrate\\_IMU](#) ()
- void [Kalman\\_filter](#) (struct [MATRIX](#) \*x, struct [MATRIX](#) \*P, float z, struct [MATRIX](#) Q, struct [MATRIX](#) R, float dt, struct [MATRIX](#) EYE2)
- void \* [read\\_IMU\\_parallel](#) (void \*args)
- void \* [get\\_filtered\\_attitude\\_parallel](#) (void \*args)

### Variables

- unsigned char [IMU\\_SYNCED](#) =0  
*If =1, then the IMU and Raspberry Pi UART communication has been synced, =0 otherwise.*
- unsigned char [IMU\\_TX](#) [2] = "##"  
*Buffer holding transmit message to IMU (call to send Euler angles NOW)*
- int [num\\_av\\_vars](#) =0  
*How many angles have we collected to average?*
- float [dt](#)  
*The timestep for derivatives (time passed in [s] between current and last iteration)*

#### Average Euler angles group

*Average Euler angle values obtained during calibration (zeroing) period*

- float `psi_av` =0  
*Average value of psi (yaw) during calibration period.*
- float `theta_av` =0  
*Average value of theta (pitch) during calibration period.*
- float `phi_av` =0  
*Average value of phi (roll) during calibration period.*

### Last read Euler angles group

*These are the last set of Euler angles that were read in from IMU during the previous iteration (1 time step ago).*

- float `psi_save_last` =-9999.0  
*Last read value of psi.*
- float `theta_save_last` =-9999.0  
*Last read value of theta.*
- float `phi_save_last` =-9999.0  
*Last read value of phi.*

## 5.3.1 Detailed Description

IMU functions file (contains IMU comms, logs and filters).

### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

### Version

1.0

This file contains all necessary functions regarding the communication with the IMU and the processing (filtering) of received data.

## 5.3.2 Function Documentation

### 5.3.2.1 void Calibrate\_IMU ( )

Zero the IMU data, which means spend some time to get an average reading for psi, theta and phi and then use that average reading to construct a matrix that would zero all three angles for the rocket orientation at which the rocket is in when this function executes (i.e. rocket *s\_t\_a\_t\_i\_o\_n\_n\_a\_r\_y* on launch pad).

### 5.3.2.2 void close\_port ( int fd )

This function closes the serial connection identified by handle fd.

#### Parameters

<code>fd</code>	The file handle.
-----------------	------------------

### 5.3.2.3 void construct\_zeroed\_DCM ( )

Construct a DCM (Direct Cosine Matrix), i.e. a rotation from body (non-inertial) ==> world (inertial) coordinates based on currently stored yaw, pitch and roll matrices.

Meaning of "zeroed" : we pre-multiply by `R_MATRIX` to get the DCM which is `=[]` ==> the IMU is in the orientation at which it was calibrated ("zeroed").

## 5.3.2.4 void Find\_raw\_Euler\_angles ( )

Take a numerical derivative of the euler angles to get angular rates [(rad)/s].

## 5.3.2.5 void \* get\_filtered\_attitude\_parallel ( void \* args )

This (p)thread does the sole job of filtering received data from the IMU. It is cadenced at the 1/IMU\_READ\_TIME-STEP [MHz] frequency and so, each time that an iteration is done, it collects the most recently available data from the IMU (stored in psi,theta,phi,accelX, accelY and accelZ variables) and processes/filters it, then saves it to a log file.

## Parameters

<i>args</i>	A pointer to the input arguments (we have none for this thread)
-------------	---

## 5.3.2.6 void get\_old\_attr ( int fd, struct termios \* old\_options )

This function saves the current serial connection attributes old\_attributes of the connection identified by handle fd.

## Parameters

<i>fd</i>	The file handle.
<i>old_options</i>	the old UART connection attributes.

## 5.3.2.7 void Kalman\_filter ( struct MATRIX \* x, struct MATRIX \* P, float z, struct MATRIX Q, struct MATRIX R, float dt, struct MATRIX EYE2 )

This function applies a real-time Kalman filter on the signal z. Consequently, this function is called each time a new value of z is read.

## Parameters

<i>x</i>	Predicted a priori and then updated a posteriori state estimate (it's the matrix version of the filtered value!).
<i>P</i>	Predicted a priori and then updated a posteriori estimate covariance.
<i>z</i>	The input, i.e. the noisy signal.
<i>Q</i>	Covariance matrix of process noise (i.e. how much noise is there in the actual physics of the plant system?).
<i>R</i>	Covariance matrix of observation (measurement) noise (i.e. how much noise is there is our sensors?).
<i>dt</i>	The time step.
<i>EYE2</i>	A [2x2] identity matrix.

## 5.3.2.8 float min\_of\_set ( float now, float before )

This function is used to overcome the difficulty of the atan2() function used in the Razor IMU firmware wrapping in the +/-M\_PI band, hence producing great discontinuities in the signal that would make filtering useless. What we do is, given a value before, we return now2=now+x\*2\*M\_PI where x is such that the difference between now2 and before is minimized.

Example : if before=+179 and now=-178 (because atan2() wrapped), then min\_of\_set(now,before) will return now2=now+1\*360=182 hence we don't have the wrapping problem anymore! Note that in the example degrees were used to facilitate understanding; the GNC program works in radians directly!

## Parameters

<i>now</i>	The value we want to adjust, which has possible wrapped and we want to "unwrap".
<i>before</i>	The previously collected value, e.g. <a href="#">psi_save_last</a> , which has itself ALSO been adjusted in the previous iteration by <a href="#">min_of_set()</a> !

## 5.3.2.9 void open\_serial\_port ( int \* fd, char \* directory )

This function opens the serial connection with the serial device connected to directory (e.g. /dev/ttyUSB0) and saves the file handle to \*fd.

## Parameters

<i>fd</i>	Pointer to the file handle.
<i>directory</i>	Pointer to the string which defines the directory.

## 5.3.2.10 void \* read\_IMU\_parallel ( void \* args )

This is a (p)thread which does only one thing : read the raw IMU values:

- float psi
- float theta
- float phi
- float accelX
- float accelY
- float accelZ

Once read, these values become available to be saved & used by other threads (such as the filtering thread).

## Parameters

<i>args</i>	A pointer to the input arguments (we have none for this thread)
-------------	---

## 5.3.2.11 void reset\_old\_attr\_port ( int fd, struct termios \* old\_options )

This function restores to the connection identified by handle fd its old settings, which were saved back right after the connection was opened.

## Parameters

<i>fd</i>	The file handle.
<i>old_options</i>	the old UART connection attributes.

## 5.3.2.12 void set\_new\_attr ( int fd, struct termios \* old\_options, struct termios \* new\_options )

This function sets new\_options for the connection identified by handle fd.

## Parameters

<i>fd</i>	The file handle.
<i>old_options</i>	The old UART connection attributes.
<i>new_options</i>	The new UART connection attributes that we want to use.

#### 5.3.2.13 void set\_to\_blocking ( int *fd* )

We always open a serial connection with O\_NONBLOCK in order to avoid the open() function hanging forever due to a badly configured connection from before (not the fault of our program). If we wish to use this connection with O\_NONBLOCK disabled (i.e. in blocking mode) then we call this function, which removes the O\_NONBLOCK, thus setting the serial mode to **blocking**

##### Parameters

<i>fd</i>	The file handle
-----------	-----------------

#### 5.3.2.14 float TO\_DEG ( float *angle* )

Convert from radians to degrees

#### 5.3.2.15 void Treat\_reply ( char \* *comparison\_string* )

This function handles user input, loops until the user inputs the right input (and gives cues to the user if he/she doesn't input the right input).

##### Parameters

<i>comparison_string</i>	The string that the user must enter.
--------------------------	--------------------------------------

#### 5.3.2.16 void zero\_Euler\_angles ( )

Convert the Euler angles received from IMU and adjusted for wrapping by [min\\_of\\_set\(\)](#) function into the "zeroed" Euler angles in that they would all be =0.0 if the rocket is in the orientation at which it was calibrated (i.e. at which its Euler angles were "zeroed"/at which the "zero point" of the Euler angles was taken).

## 5.4 imu\_header.h File Reference

IMU header file.

```
#include <termios.h>
```

### Macros

- `#define MAX_BUFFER 24`  
The max buffer size for receiving data from IMU.

### Variables

- unsigned char `IMU_RX [MAX_BUFFER]`  
Buffer holding received values via UART from Razor IMU.

- char [IMU\\_SYNC\\_RECEIVE](#) [1]  
*Buffer used for receiving the 2-character (2-byte) synch token from the IMU during synchronization.*
- unsigned char [IMU\\_TX](#) [2]  
*Buffer holding transmit message to IMU (call to send Euler angles NOW)*
- int [RAZOR\\_UART](#)  
*Holds Razor IMU connection file.*
- struct termios [new\\_razor\\_uart\\_options](#)  
*New IMU UART connection options.*
- struct termios [old\\_razor\\_uart\\_options](#)  
*Old IMU UART connection options (those that were initially present when program started)*
- unsigned long int [CALIB\\_\\_TIME](#)  
*Calibration time [us], i.e. microseconds.*
- int [num\\_av\\_vars](#)  
*How many angles have we collected to average?*
- char [reply](#) [30]  
*User input string.*
- float [temp1](#)  
*Temp variable used in [min\\_of\\_set\(\)](#)*
- float [temp2](#)  
*Temp variable used in [min\\_of\\_set\(\)](#)*
- struct [MATRIX R\\_MATRIX](#)  
*Matrix which zeroes the Euler angles for the calibrated orientation.*
- struct [MATRIX DCM\\_MATRIX](#)  
*Direct Cosine Matrix.*
- float [dt](#)  
*The timestep for derivatives (time passed in [s] between current and last iteration)*
- struct [MATRIX EYE2](#)  
*[2x2] identity matrix, used in [Kalman\\_filter\(\)](#)*

### IMU reception group

*variables holdin the raw values received from IMU*

- float [psi](#)  
*Yaw angle.*
- unsigned char [yaw\\_bytes](#) [4]  
*Yaw angle bytes used for conversion to float.*
- float [theta](#)  
*Pitch angle.*
- unsigned char [pitch\\_bytes](#) [4]  
*Pitch angle bytes used for conversion to float.*
- float [phi](#)  
*Roll angle.*
- unsigned char [roll\\_bytes](#) [4]  
*Roll angle bytes used for conversion to float.*
- float [accelX](#)  
*X-acceleration.*
- unsigned char [accelX\\_bytes](#) [4]  
*X-acceleration angle bytes used for conversion to float.*
- float [accelY](#)  
*Y-acceleration.*
- unsigned char [accelY\\_bytes](#) [4]  
*Y-acceleration angle bytes used for conversion to float.*
- float [accelZ](#)



*Z-acceleration.*

- unsigned char [accelZ\\_bytes](#) [4]

*Z-acceleration angle bytes used for conversion to float.*

### Current accelerations

These are the accelerations saved into the filtering thread [get\\_filtered\\_attitude\\_parallel\(\)](#) when it does an iteration, hence they are equal to the most recent [accelX](#), [accelY](#) and [accelZ](#) that have been read from the IMU. The filtering thread simply logs them into the [imu\\_log](#) file.

- float [accelX\\_save](#)  
*Saved X-acceleration.*
- float [accelY\\_save](#)  
*Saved Y-acceleration.*
- float [accelZ\\_save](#)  
*Saved Z-acceleration.*

### Current Euler angles group

These angles are the ones saved into the filtering thread [get\\_filtered\\_attitude\\_parallel\(\)](#) when it does an iteration, hence they are equal to the most recent [psi](#), [theta](#) and [phi](#) that have been read from the IMU.

- float [psi\\_save](#)  
*Saved yaw angle.*
- float [theta\\_save](#)  
*Saved pitch angle.*
- float [phi\\_save](#)  
*Saved roll angle.*

### Last read Euler angles group

These are the last set of Euler angles that were read in from IMU during the previous iteration (1 time step ago).

- float [psi\\_save\\_last](#)  
*Last read value of psi.*
- float [theta\\_save\\_last](#)  
*Last read value of theta.*
- float [phi\\_save\\_last](#)  
*Last read value of phi.*

### Filtered Euler angles and angular rates

These are the filtered versions of the [psi\\_save](#), [theta\\_save](#) and [phi\\_save](#) signals and their numerical derivatives (see [Find\\_raw\\_Euler\\_angular\\_velocities\(\)](#)).

- float [psi\\_filt](#)  
*Filtered yaw.*
- float [psi\\_dot\\_filt](#)  
*Filtered yaw rate.*
- float [theta\\_filt](#)  
*Filtered pitch.*
- float [theta\\_dot\\_filt](#)  
*Filtered pitch rate.*
- float [phi\\_filt](#)  
*Filtered roll.*
- float [phi\\_dot\\_filt](#)  
*Filtered roll rate.*

### Average Euler angles group

Average Euler angle values obtained during calibration (zeroing) period

- float [psi\\_av](#)  
Average value of psi (yaw) during calibration period.
- float [theta\\_av](#)  
Average value of theta (pitch) during calibration period.
- float [phi\\_av](#)  
Average value of phi (roll) during calibration period.

### Euler angular rates group

The unfiltered, noisy numerical time derivatives of the Euler angles read in from the IMU

- float [psi\\_dot](#)  
Time derivative of psi.
- float [theta\\_dot](#)  
Time derivative of theta.
- float [phi\\_dot](#)  
Time derivative of phi.

### Body rates group

These are the BODY rates (angular velocity about X Y and Z body axes of rocket)

- float [wx](#)  
X-body rate.
- float [wy](#)  
Y-body rate.
- float [wz](#)  
Z-body rate.

## 5.4.1 Detailed Description

IMU header file.

Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

Version

1.0

This is the header to [imu\\_funcs.c](#) containing necessary definitions and initializations.

## 5.5 la\_funcs.c File Reference

Lienar Algebra functions source file.

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "la_header.h"
```

### Functions

- struct [MATRIX](#) [initMatrix](#) (size\_t rows, size\_t cols)
- struct [MATRIX](#) [mmultiply](#) (struct [MATRIX](#) A, struct [MATRIX](#) B)
- struct [MATRIX](#) [madd](#) (struct [MATRIX](#) A, struct [MATRIX](#) B)
- struct [MATRIX](#) [msubtract](#) (struct [MATRIX](#) A, struct [MATRIX](#) B)
- struct [MATRIX](#) [minverse\\_1x1](#) (struct [MATRIX](#) A)
- struct [MATRIX](#) [transpose](#) (struct [MATRIX](#) A)

### 5.5.1 Detailed Description

Lienar Algebra functions source file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This file contains necessary linear algebra functions such as matrix initialization, multiplication, addition, subtraction, etc. These functions are used primarily in the [Kalman\\_filter\(\)](#) function (found in [imu\\_funcs.c](#))

### 5.5.2 Function Documentation

#### 5.5.2.1 struct MATRIX initMatrix ( size\_t rows, size\_t cols )

This function allocates space for a [rows x cols] size matrix. Attention : returned matrix A is just pointers that don't point to any default value!

##### Parameters

<i>rows</i>	Number of matrix rows.
<i>cols</i>	Number of matrix columns.

#### 5.5.2.2 struct MATRIX madd ( struct MATRIX A, struct MATRIX B )

This function return  $C=A+B$  (matrix addition).

##### Parameters

<i>A</i>	The first matrix.
<i>B</i>	The second matrix.

#### 5.5.2.3 struct MATRIX minverse\_1x1 ( struct MATRIX A )

This function inverses a 1 by 1 matrix - i.e. a scalar, but in [MATRIX](#) format (as struct [MATRIX](#)) that is usable in a matrix expression, i.e. doing  $([1 \times 2] * [2 \times 1]) + [1 \times 1]^{(-1)}$ .

##### Parameters

<i>A</i>	[1x1] matrix.
----------	---------------

#### 5.5.2.4 struct MATRIX mmultiply ( struct MATRIX A, struct MATRIX B )

This fuction returns  $C=A*B$  where C is returned from the function and A,B are previously defined matrices

##### Parameters

<i>A</i>	The pre-multiplied matrix
----------	---------------------------

<i>B</i>	The post-multiplied matrix
----------	----------------------------

#### 5.5.2.5 struct **MATRIX** msubtract ( struct **MATRIX** *A*, struct **MATRIX** *B* )

This function returns  $C=A-B$  (matrix subtraction)

##### Parameters

<i>A</i>	The first matrix.
<i>B</i>	The second matrix.

#### 5.5.2.6 struct **MATRIX** transpose ( struct **MATRIX** *A* )

This function transposes a matrix, returning  $B=A'$  (real transpose of *A*).

##### Parameters

<i>A</i>	A matrix.
----------	-----------

## 5.6 la\_header.h File Reference

Linear Algebra header file.

### Classes

- struct [MATRIX](#)

### 5.6.1 Detailed Description

Linear Algebra header file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This is the header to [la\\_funcs.c](#) containing necessary definitions and initializations.

## 5.7 master.c File Reference

Master GNC source file (main function file).

```

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <stdlib.h>
#include <termios.h>
#include <linux/spi/spidev.h>
#include <signal.h>
#include <sys/time.h>
#include <math.h>
#include <string.h>
#include <pthread.h>
#include "control_header.h"
#include "master_header.h"
#include "imu_header.h"
#include "la_header.h"
#include "msp430_header.h"
#include "simplex_header.h"
#include "rpi_gpio_header.h"
#include "spycam_header.h"
#include "pressure_header.h"

```

## Functions

- int [main](#) (void)

## Variables

- unsigned long long int [ENGINE\\_\\_BURN\\_TIME](#) =1100000  
*Upper bound on time [us] between engine start and engine burnout // TODO : change this with Xavier!*
- unsigned long long int [ACTIVE\\_\\_CONTROL\\_TIME](#) =20000000  
*Time [us] during which control loop will be active // TODO : change this with Xavier!*
- unsigned long long int [DESCENT\\_\\_TIME](#) =15000000  
*Time [us] for rocket descent with parachute (i.e. between parachutes opening and a soft touchdown) // TODO : change this with Xavier!*
- unsigned long long int [CONTROL\\_\\_TIME\\_STEP](#) =20000  
*=1/(control loop frequency [MHz]), the time interval between applying control, in [us]*
- unsigned long long int [SPI\\_\\_READ\\_TIMESTEP](#) =20000  
*Time intervals [us] at which we read over SPI (for pressure/temperature Honeywell sensors).*
- unsigned long long int [IMU\\_\\_READ\\_TIMESTEP](#) =20000  
*Time intervals [us] at which we read over UART the IMU data.*
- unsigned long int [CALIB\\_\\_TIME](#) = 5000000  
*Calibration time [us], i.e. microseconds.*
- unsigned char [SPI\\_quit](#) =0  
*==0 by default, ==1 signals the SPI reading thread ([get\\_readings\\_SPI\\_parallel\(\)](#)) to exit.*
- unsigned char [IMU\\_quit](#) =0  
*==0 by default, ==1 signals the IMU reading and filtering threads ([read\\_IMU\\_parallel\(\)](#)) and [get\\_filtered\\_attitude\\_parallel\(\)](#) to exit.*
- unsigned char [PWM1](#) =0  
*PWM value for the R1 valve.*
- unsigned char [PWM2](#) =0  
*PWM value for the R2 valve.*

- unsigned char `PWM3` =0  
*PWM value for the R3 valve.*
- unsigned char `PWM4` =0  
*PWM value for the R4 valve.*
- double `R1` =0  
*Valve R1 thrust.*
- double `R2` =0  
*Valve R2 thrust.*
- double `R3` =0  
*Valve R3 thrust.*
- double `R4` =0  
*Valve R4 thrust.*
- double `d` =0.005  
*[m] offset distance of RCS valves from centerline (for roll control)*
- double `Fpitch` =0  
*Pitch force (parallel to body -Z axis, so as to produce positive pitch rate when  $Fpitch > 0$  (right hand rule))*
- double `Fyaw` =0  
*Yaw force (parallel to body +Y axis, so as to produce positive yaw rate when  $Fyaw > 0$  (right hand rule))*
- double `Mroll` =0  
*Roll moment (positive about +X axis, so as to produce positive roll rate when  $Mroll > 0$  (right hand rule))*
- int `N` =4  
*Number of variables in cost function. Our variables are R1, R2, R3, R4 so N=4.*
- int `M1` =0  
*No ( $\leq$ ) type constraints.*
- int `M2` =0  
*No ( $\geq$ ) type constraints.*
- int `M3` =3  
*3 ( $=$ ) type constraints (for Fpitch, Fyaw, Mroll)*
- int `M` =3  
*Total number of constraints ( $M=M1+M2+M3$ )*
- char `flight_type` =0  
*=1 for active control flight, =0 for passive flight (i.e. only data logging)*
- FILE \* `error_log` =NULL  
*Error log (stores errors)*
- FILE \* `pressure_log` =NULL  
*Pressure log (stores pressures and temperatures collected by Honeywell HSC TruStability sensors)*
- FILE \* `imu_log` =NULL  
*IMU log (stores raw and filtered Euler angles and angular rates, the body rates and the accelerometer data)*
- FILE \* `control_log` =NULL  
*Control log (stores the control loop data such as computed `Fpitch`, `Fyaw`, `Mroll`, the optimally distributed valves thrusts `R1`,...,`R4` and the computed PWM signals `PWM1`,...,`PWM4`)*
- struct `bcm2835_peripheral gpio` = {`GPIO_BASE`}  
*Our access register to the Raspberry Pi's GPIOs.*
- unsigned char `launch_detect_gpio` =12  
*Number of GPIO (i.e.  $GPIO<num>$ ) to which the launch umbilical cable is connected and hence which detects the launch.*

### Control algorithm input variables

Below 6 variables are the ones that the control algorithm "sees", as in that they are updated at our CONTROL frequency, which we decide, while the angles read from the IMU are read as fast as possible to data-log all information on rocket orientation!

- float `psi_cont` =0  
Yaw angle.
- float `psidot_cont` =0  
Yaw rate.
- float `theta_cont` =0  
Pitch angle.
- float `thetadot_cont` =0  
Pitch rate.
- float `phi_cont` =0  
Roll angle.
- float `wx_cont` =0  
X-body rate.

### Reference angles and rates

Below are the values that we want to achieve when controlling the rocket. In our case they are 0 because we want to achieve a steady, perfectly vertical orientation. But they can be even time-varying in other applications - e.g. for guiding the rocket along a trajectory.

- float `psi_ref` =0  
Yaw angle reference [(rad)].
- float `theta_ref` =0  
Pitch angle reference [(rad)].
- float `wx_ref` =0  
Roll rate reference [(rad)/s].

#### 5.7.1 Detailed Description

Master GNC source file (main function file).

##### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

##### Version

1.0

This is the master file of the GNC program which manages all parallel threads and calls all necessary functions for the collection and processing of data as well as for rocket control.

#### 5.7.2 Function Documentation

##### 5.7.2.1 `int main ( void )`

**This is the main function of the flight software.** This function is what defines the sequence of steps that occur during pre-flight and flight and what orchestrates all processes that occur, such as the starting of parallel threads, of active control and of data logging. It accepts no input parameters.

#### 5.7.3 Variable Documentation

##### 5.7.3.1 `struct bcm2835_peripheral gpio = {GPIO_BASE}`

Our access register to the Raspberry Pi's GPIOs.

The GPIO port access variable.

## 5.8 master\_funcs.c File Reference

Master functions file.

```
#include <sys/time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include "master_header.h"
#include "spycam_header.h"
```

### Functions

- void [write\\_to\\_file\\_custom](#) (FILE \*file\_ptr, char \*string, FILE \*[error\\_log](#))
- void [check\\_time](#) (struct timeval \*now, struct timeval before, struct timeval elapsed, unsigned long long int \*time)
- void [open\\_file](#) (FILE \*\*log, char \*path, char \*setting, FILE \*[error\\_log](#))
- void [open\\_error\\_file](#) (FILE \*\*[error\\_log](#), char \*path, char \*setting)
- void [passive\\_wait](#) (struct timeval \*now, struct timeval \*before, struct timeval \*elapsed, unsigned long long int \*time, unsigned long long int TIME\_\_STEP)
- void [search\\_PWM](#) (double thrust, unsigned char \*pwm)

### Variables

- unsigned int [PWM\\_valve\\_charac](#) [[VALVE\\_CHARAC\\_RESOLUTION](#)] = {0,6,14,25,39,50,63,75,87,98,106,115,127}  
*PWM value of characteristic thrust curve.*
- double [R\\_valve\\_charac](#) [[VALVE\\_CHARAC\\_RESOLUTION](#)] = {0.0000,0.0091,0.0478,0.0981,0.1656,0.-2245,0.2816,0.3344,0.3737,0.4166,0.4406,0.4676,0.5000}  
*Thrust (R) value of characteristic thrust value.*

### 5.8.1 Detailed Description

Master functions file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This file contains some of the primary, "every-day" functions used by the GNC algorithm to achieve its tasks. The file is called "master\_funcs" because the functions here are general in that they tend to appear everywhere in the code to perform routine tasks like file I/O, waiting, etc.

### 5.8.2 Function Documentation

#### 5.8.2.1 void [check\\_time](#) ( struct timeval \* now, struct timeval before, struct timeval elapsed, unsigned long long int \* time )

This function the value pointed to by time with the time that has passed between \*now and the last time gettimeofday(&before,NULL) was called.



## Parameters

<i>now</i>	The current time (struct timeval).
<i>before</i>	The previous time (struct timeval).
<i>elapsed</i>	The time elapsed between now and before (struct timeval).
<i>time</i>	The time in [us] that elapsed between now and before.

## 5.8.2.2 void open\_error\_file ( FILE \*\* error\_log, char \* path, char \* setting )

This file opens the error file. The function is separate from [open\\_file\(\)](#) because in case of error we can't write to the error file.

## Parameters

<i>error_log</i>	Pointer to the error log.
<i>path</i>	This is the path to the error log.
<i>setting</i>	This is the mode in which we want to open the error log (e.g 'w', write only).

## 5.8.2.3 void open\_file ( FILE \*\* log, char \* path, char \* setting, FILE \* error\_log )

This function opens a file at \*path in the mode \*setting. If unsuccessful, it writes the error into the error\_log (which may itself be unsuccessful and throws an error).

## Parameters

<i>log</i>	This is the pointer to the file we want to open.
<i>path</i>	This is the path to the file.
<i>setting</i>	This is the mode in which we want to open the file (e.g. 'w', write only).
<i>error_log</i>	Pointer to the error log, used in case of error opening log (argument 1).

## 5.8.2.4 void passive\_wait ( struct timeval \* now, struct timeval \* before, struct timeval \* elapsed, unsigned long long int \* time, unsigned long long int TIME\_STEP )

This function does a passive (instead of a busy) wait.

## Parameters

<i>now</i>	The time now (struct timeval).
<i>before</i>	The time before (struct timeval).
<i>elapsed</i>	The time elapsed (struct timeval).
<i>time</i>	The time in [us] between now and before.
<i>TIME_STEP</i>	The time we want to iterate at within the loop from which <a href="#">passive_wait()</a> was called.

## 5.8.2.5 void write\_to\_file\_custom ( FILE \* file\_ptr, char \* string, FILE \* error\_log )

This function allows to write a custom string to a file

## Parameters

<i>file_ptr</i>	Pointer to file.
<i>string</i>	The file path.
<i>error_log</i>	Pointer to error log file.

## 5.9 master\_header.h File Reference

Master header file.

```
#include <stdint.h>
#include <stdio.h>
#include <sys/time.h>
#include <pthread.h>
#include "la_header.h"
```

### Macros

- `#define VALVE_CHARAC_RESOLUTION 13`  
*The number of points there are in the calibrated valve thrust curve (flow rate vs. PWM)*

### Variables

- char `ERROR_MESSAGE` [200]  
*Allocate buffer for an error message to be printed into `error_log` if errors occur.*
- struct timeval `GLOBAL__TIME_STARTPOINT`  
*Structure holding the time when the program started (very first line of `main()`)*
- unsigned char `IMU_SYNCED`  
*If =1, then the IMU and Raspberry Pi UART communication has been synced, =0 otherwise.*
- unsigned long long int `SPI__READ_TIMESTEP`  
*Time intervals [us] at which we read over SPI (for pressure/temperature Honeywell sensors).*
- unsigned char `SPI_quit`  
*==0 by default, ==1 signals the SPI reading thread (`get_readings_SPI_parallel()`) to exit.*
- unsigned long long int `IMU__READ_TIMESTEP`  
*Time intervals [us] at which we read over UART the IMU data.*
- unsigned char `IMU_quit`  
*==0 by default, ==1 signals the IMU reading and filtering threads (`read_IMU_parallel()` and `get_filtered_attitude_parallel()`) to exit.*
- unsigned char `PWM1`  
*PWM value for the R1 valve.*
- unsigned char `PWM2`  
*PWM value for the R2 valve.*
- unsigned char `PWM3`  
*PWM value for the R3 valve.*
- unsigned char `PWM4`  
*PWM value for the R4 valve.*
- double `R1`  
*Valve R1 thrust.*
- double `R2`  
*Valve R2 thrust.*
- double `R3`  
*Valve R3 thrust.*
- double `R4`  
*Valve R4 thrust.*
- unsigned int `PWM_valve_charac` [`VALVE_CHARAC_RESOLUTION`]  
*PWM value of characteristic thrust curve.*
- double `R_valve_charac` [`VALVE_CHARAC_RESOLUTION`]

- *Thrust (R) value of characteristic thrust value.*
- double **d**  
*[m] offset distance of RCS valves from centerline (for roll control)*
- double **Fpitch**  
*Pitch force (parallel to body -Z axis, so as to produce positive pitch rate when Fpitch>0 (right hand rule))*
- double **Fyaw**  
*Yaw force (parallel to body +Y axis, so as to produce positive yaw rate when Fyaw>0 (right hand rule))*
- double **Mroll**  
*Roll moment (positive about +X axis, so as to produce positive roll rate when Mroll>0 (right hand rule))*
- int **N**  
*Number of variables in cost function. Our variables are R1, R2, R3, R4 so N=4.*
- int **M1**  
*No (<=) type constraints.*
- int **M2**  
*No (>=) type constraints.*
- int **M3**  
*3 (=) type constraints (for Fpitch, Fyaw, Mroll)*
- int **M**  
*Total number of constraints (M=M1+M2+M3)*
- float **VALVE\_\_MAX\_THRUST**  
*Maximum thrust of RCS solenoid valves (i.e. when fully opened)*
- char **MESSAGE** [700]  
*Message buffer string sometimes used for putting together a string, then writing it to a file.*
- pthread\_mutex\_t **error\_log\_write\_lock**  
*Mutex to protect multiple threads from writing to the error log file at once.*
- struct **MATRIX A\_kalman**  
*Temporary matrix for Kalman filtering, dynamic equation  $\dot{x}=A\_kalman*x$  ;  $y=C\_kalman*x$ , but in discrete time!*
- struct **MATRIX C\_kalman**  
*Temporary matrix for Kalman filtering, dynamic equation  $\dot{x}=A\_kalman*x$  ;  $y=C\_kalman*x$ , but in discrete time!*
- struct **MATRIX inn**  
*Temporary matrix "Innovation or measurement residual" (see [Wikipedia](#))*
- struct **MATRIX S**  
*Temporary matrix "Innovation (or residual) covariance" (see [Wikipedia](#))*
- struct **MATRIX K**  
*Temporary matrix "Optimal Kalman gain" (see [Wikipedia](#))*
- struct **MATRIX z\_temp**  
*Temporary matrix used to convert the current scalar unfiltered signal value (e.g. [psi](#) or [theta](#), etc.) into a [1x1] **MATRIX** used in calculating [inn](#) (see [Kalman\\_filter\(\)](#)), needed due to how our linear algebra functions in [la\\_header.h](#) operate.*

### IMU timing

Contains the timing structures and variables necessary for setting the IMU filtering thread loop frequency (see [get\\_filtered\\_attitude\\_parallel\(\)](#)).

- struct timeval **now\_imu**
- struct timeval **before\_imu**
- struct timeval **elapsed\_imu**
- unsigned long long int **time\_imu**

### Pressure sensor timing

Contains the timing structures and variables necessary to set the the pressure/temperature logging thread loop frequency (see [get\\_readings\\_SPI\\_parallel\(\)](#))

- struct timeval **now\_pressure**

- struct timeval **before\_pressure**
- struct timeval **elapsed\_pressure**
- unsigned long long int **time\_pressure**

### General loop timing

*Contains the timing structures and variables necessary to make sure a loop executes a given amount of time*

- struct timeval **now\_loop**
- struct timeval **before\_loop**
- struct timeval **elapsed\_loop**
- unsigned long long int **time\_loop**

### Control loop timing

*Contains the timing structures and variables necessary for timing necessary to set the control loop frequency (see [main\(\)](#))*

- struct timeval **now\_control**
- struct timeval **before\_control**
- struct timeval **elapsed\_control**
- unsigned long long int **time\_control**

### IMU filter get global time

*Contains the timing structures and variables necessary for getting the global time within the IMU data filtering loop (see [get\\_filtered\\_attitude\\_parallel\(\)](#))*

- struct timeval **now\_imu\_glob**
- struct timeval **elapsed\_imu\_glob**
- unsigned long long int **time\_imu\_glob**

### Pressure read get global time

*Contains the timing structures and variables necessary for getting the global time within the pressure/temperature sensor data logging loop (see [get\\_readings\\_SPI\\_parallel\(\)](#))*

- struct timeval **now\_pressure\_glob**
- struct timeval **elapsed\_pressure\_glob**
- unsigned long long int **time\_pressure\_glob**

### Control loop get global time

*Contains the timing structures and variables necessary for getting the global time within the control loop (see [main\(\)](#))*

- struct timeval **now\_control\_glob**
- struct timeval **elapsed\_control\_glob**
- unsigned long long int **time\_control\_glob**

### Log files group

*Contains the pointers to files we use for recording flight data.*

- FILE \* [error\\_log](#)  
*Error log (stores errors)*
- FILE \* [pressure\\_log](#)  
*Pressure log (stores pressures and temperatures collected by Honeywell HSC TruStability sensors)*
- FILE \* [imu\\_log](#)  
*IMU log (stores raw and filtered Euler angles and angular rates, the body rates and the accelerometer data)*
- FILE \* [control\\_log](#)  
*Control log (stores the control loop data such as computed [Fpitch](#), [Fyaw](#), [Mroll](#), the optimally distributed valves thrusts [R1](#),...,[R4](#) and the computed PWM signals [PWM1](#),...,[PWM4](#))*

### Yaw Kalman filtering matrices

These matrices pertain to the real-time Kalman filtering of the yaw angle and angular rate

- struct [MATRIX P\\_psi](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [psi\\_filt](#) estimate.
- struct [MATRIX P\\_psidot](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [psi\\_dot\\_filt](#) estimate.
- struct [MATRIX x\\_psi](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [psi\\_filt](#))
- struct [MATRIX x\\_psidot](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [psi\\_dot\\_filt](#))
- struct [MATRIX Q\\_psi](#)  
Covariance matrix of process noise of [psi](#).
- struct [MATRIX Q\\_psidot](#)  
Covariance matrix of process noise of [psi\\_dot](#).
- struct [MATRIX R\\_psi](#)  
Covariance matrix of observation of [psi](#).
- struct [MATRIX R\\_psidot](#)  
Covariance matrix of observation of [psi\\_dot](#).

### Pitch Kalman filtering matrices

These matrices pertain to the real-time Kalman filtering of the pitch angle and angular rate

- struct [MATRIX P\\_theta](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [theta\\_filt](#) estimate.
- struct [MATRIX x\\_theta](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [theta\\_filt](#))
- struct [MATRIX Q\\_theta](#)  
Covariance matrix of process noise of [theta](#).
- struct [MATRIX R\\_theta](#)  
Covariance matrix of observation of [theta](#).
- struct [MATRIX P\\_thetadot](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [theta\\_dot\\_filt](#) estimate.
- struct [MATRIX x\\_thetadot](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [theta\\_dot\\_filt](#))
- struct [MATRIX Q\\_thetadot](#)  
Covariance matrix of process noise of [theta\\_dot](#).
- struct [MATRIX R\\_thetadot](#)  
Covariance matrix of observation of [theta\\_dot](#).

### Roll Kalman filtering matrices

These matrices pertain to the real-time Kalman filtering of the roll angle and angular rate

- struct [MATRIX P\\_phi](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [phi\\_filt](#) estimate.
- struct [MATRIX x\\_phi](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [phi\\_filt](#))
- struct [MATRIX Q\\_phi](#)  
Covariance matrix of process noise of [phi](#).
- struct [MATRIX R\\_phi](#)  
Covariance matrix of observation of [phi](#).
- struct [MATRIX P\\_phidot](#)  
Predicted a priori and then updated a posteriori estimate covariance matrix of the [phi\\_dot\\_filt](#) estimate.
- struct [MATRIX x\\_phidot](#)  
Predicted a priori and then updated a posteriori state estimate (the [MATRIX](#) version of [phi\\_dot\\_filt](#))
- struct [MATRIX Q\\_phidot](#)  
Covariance matrix of process noise of [phi\\_dot](#).
- struct [MATRIX R\\_phidot](#)  
Covariance matrix of observation of [phi\\_dot](#).

### 5.9.1 Detailed Description

Master header file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This is the header to [master\\_funcs.c](#) containing necessary definitions and initializations.

### 5.9.2 Variable Documentation

#### 5.9.2.1 float VALVE\_\_MAX\_THRUST

Maximum thrust of RCS solenoid valves (i.e. when fully opened)

Maximum thrust of RCS solenoid valves (i.e. when fully opened)

## 5.10 msp430\_funcs.c File Reference

MSP430 functions file.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/termios.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "msp430_header.h"
#include "master_header.h"
```

### Functions

- void [MSP430\\_UART\\_receive](#) ()
- void [MSP430\\_UART\\_write](#) (char MSP430\_TX[3])
- void [MSP430\\_UART\\_write\\_PWM](#) (unsigned char PWMA, unsigned char PWMB, unsigned char PWMC)

### 5.10.1 Detailed Description

MSP430 functions file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

## Version

1.0

This file contains functions necessary for communication with the MSP430 slave microcontroller that is used to do hardware PWM (which the Raspberry Pi is not capable of doing by itself 4 independent times) to drive the 4 RCS valves.

## 5.10.2 Function Documentation

## 5.10.2.1 void MSP430\_UART\_receive ( )

This function receives a single byte (over UART) from the MSP430. When this byte is received (it's a BLOCKING read), we know that the MSP430 has successfully processed the byte we previously sent it and hence is ready to receive another byte.

Note : MSP430 sends the byte '!', but note that we do not actually check that the byte equals '!' (0x21) because:

- The connections has been tested to never fail during numerous pre-tests
- Even if the byte is not '!' due to signal noise, what can we do? The communication speed is optimised for speed and not robustness with many failsafes - thus we have no way of resending the MSP430 the previous byte in the case that we do not receive '!'
- As in the above point, we optimised the code for speed, so checking for equality to '!' is an additional time spent.

## 5.10.2.2 void MSP430\_UART\_write ( char MSP430\_TX[3] )

This function sends the MSP430 a 3 character string which is:

- "@s!" : arm the MSP430 for PWM generation
- "@e!" : stop PWM transmission and do a software reset, which puts the MSP430 into a state where it again waits for "@s!"

## Parameters

<i>MSP430_TX</i>	Contains the 3-byte (3-character) string to send to the MSP430
------------------	--

## 5.10.2.3 void MSP430\_UART\_write\_PWM ( unsigned char PWMA, unsigned char PWMB, unsigned char PWMC )

This function sends PWM values to the MSP430. Bit field conversion below: // TODO : in MSP430 program, add condition "not reading PWM" for # and @ -> avoids that new PWM sent if PWM of 35 is sent!

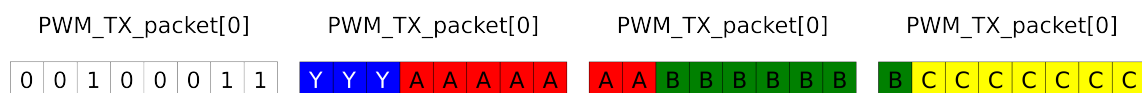


Figure 5.1: 4 byte packet send by Raspberry Pi to MSP430 to update PWM values

4 bytes are hence sent to the MSP430 where:

- PWM\_TX\_packet[0] (byte 1) : send "#" message which tells MSP430 that the following 3 bytes are PWM values

- PWM\_TX\_packet[1] (byte 2) : YYY is a code which says:
  - YYY==001 : PWM1 is 0
  - YYY==010 : PWM2 is 0
  - YYY==011 : PWM3 is 0
  - YYY==100 : PWM4 is 0 This is because at every point in time only 3 valves are assigned thrusts - this is the consequence of the RCS physics and optimal thrust assignment. It's useless to waste 7 bits sending a 0 value, so we use just 3 to identify which of the PWM values is 0

In the above image you can see how the PWMA, PWMB and PWMC signals are distributed amongst the three bytes PWM\_TX\_packet[1] to PWM\_TX\_packet[3]. In the figure, the MSB is on the left and LSB on the right for each series of A, B and C. We call them PWMA, PWMB and PWMC because these are, in rising order from 1 to 4 the other 3 non-zero PWMs. For example:

YYY==001, therefore PWM1 is 0 so PWMA=PWM2, PWMB=PWM2, PWMC=PWM4 YYY==011, therefore PWM3 is 0 so PWMA=PWM1, PWMB=PWM2, PWMC=PWM4

You see that we simply go from PWM1 to PWM4, skipping the PWM that is 0.

## 5.11 msp430\_header.h File Reference

MSP430 header file.

### Macros

- #define `MSP430_MAX_BUFFER` 1  
*Buffer size for receiving messages from MSP430 (just '!' so 1 byte buffer is used)*

### Variables

- char `MSP430_RX` [`MSP430_MAX_BUFFER`]  
*Buffer holding received values via UART from Razor IMU.*
- int `MSP430_UART`  
*Holds Razor IMU connection file.*
- char `MSP430_reply_string`  
*String holding the MSP430 reply.*
- unsigned char `PWM_TX_packet` [4]  
*Packet of 1 byte for "#" and 5 bytes containing the 4 PWM values, to send to MSP430.*
- unsigned char `which_zero`  
*Cypher which indicates which of the PWM signals is zero.*
- struct termios `new_msp430_uart_options`  
*The new options we set for communicating the the MSP430 UART after opening it.*
- struct termios `old_msp430_uart_options`  
*The old options we save after opening the MSP430 UART connection; we retribute them before closing the connection at the end of the program.*

### 5.11.1 Detailed Description

MSP430 header file.



**Author**

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This is the header to [msp430\\_funcs.c](#) containing necessary definitions and initializations.

## 5.12 pressure\_funcs.c File Reference

Honeywell pressure/temperature sensors functions file.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <linux/spi/spidev.h>
#include <pthread.h>
#include "pressure_header.h"
#include "master_header.h"
```

**Functions**

- void [pressure\\_sensor\\_SPI\\_connect](#) (const char \*directory, unsigned int \*fd, unsigned char mode, unsigned char bits, unsigned long int max\_speed)
- void \* [get\\_readings\\_SPI\\_parallel](#) (void \*args)

**Variables**

- const char [RADIAL\\_SENSOR](#) [] = "/dev/spidev0.0"  
*File path for the radial pressure sensor SPI connection.*
- const char [AXIAL\\_SENSOR](#) [] = "/dev/spidev0.1"  
*File path for the axial pressure sensor SPI connection.*

### 5.12.1 Detailed Description

Honeywell pressure/temperature sensors functions file.

**Author**

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This file contains functions that log data from the Honeywell HSC D LN N 100MD S A 5 pressure/temperature sensors (HSC - High Accuracy, Compensated/Amplified - TruStability Series) over SPI communication.

## 5.12.2 Function Documentation

### 5.12.2.1 `void * get_readings_SPI_parallel ( void * args )`

This is a (p)thread which does the sole job of reading data from the Honeywell HSC sensors (pressure and temperature).

## Parameters

<i>args</i>	A pointer to the input arguments. We pass the SPI connection struct pointer as a void pointer and then typecast it back to a struct pointer (see <a href="#">example</a> ).
-------------	---

5.12.2.2 void `pressure_sensor_SPI_connect` ( const char \* *directory*, unsigned int \* *fd*, unsigned char *mode*, unsigned char *bits*, unsigned long int *max\_speed* )

This function opens the SPI connection to the pressure sensor connected to \*directory.

## Parameters

<i>directory</i>	The file path.
<i>fd</i>	The connection handle once opened.
<i>mode</i>	SPI mode (0, 1, 2 or 3). Honeywell HSC pressure sensors use mode 0, so that's what we use in this program!
<i>bits</i>	Number of bits per SPI transmission.
<i>max_speed</i>	The SPI connection speed.

## 5.13 pressure\_header.h File Reference

Honeywell pressure/temperature sensors header file.

## Classes

- struct [SPI\\_data](#)

## Macros

- #define [BYTE\\_NUMBER](#) 4  
*How many bytes we want to receive from the pressure sensor per reading.*

## Variables

- const char [RADIAL\\_SENSOR](#) []  
*File path for the radial pressure sensor SPI connection.*
- const char [AXIAL\\_SENSOR](#) []  
*File path for the axial pressure sensor SPI connection.*
- struct [SPI\\_data](#) [SPI\\_config](#)  
*Holds the SPI configuration.*
- unsigned char [radial\\_status](#)  
*Holds status of radial sensor.*
- float [radial\\_pressure](#)  
*Holds differential pressure reading of radially mounted pressure sensor.*
- float [radial\\_temperature](#)  
*Holds compensated temperature reading of radially mounted pressure sensor.*
- char [axial\\_status](#)  
*Holds status of axial sensor.*
- float [axial\\_pressure](#)  
*Holds differential pressure reading of axially mounted pressure sensor.*

- float [axial\\_temperature](#)  
*Holds compensated temperature reading of axially mounted pressure sensor.*
- unsigned char [data](#) [[BYTE\\_NUMBER](#)]  
*We will receive 4 bytes from the pressure sensor.*
- struct spi\_ioc\_transfer [transfer](#) [[BYTE\\_NUMBER](#)]  
*SPI transfer structure.*

### 5.13.1 Detailed Description

Honeywell pressure/temperature sensors header file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This is the header to [pressure\\_funcs.c](#) containing necessary definitions and initializations.

## 5.14 rpi\_gpio\_funcs.c File Reference

GPIO functions file.

```
#include <fcntl.h>
#include "rpi_gpio_header.h"
```

### Functions

- int [map\\_peripheral](#) (struct [bcm2835\\_peripheral](#) \*p)
- void [unmap\\_peripheral](#) (struct [bcm2835\\_peripheral](#) \*p)

### 5.14.1 Detailed Description

GPIO functions file.

#### Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This file contains GPIO low-level access functions. This file is a slightly modified version of the code provided by Pieter-Jan Van de Maele [here](#).

### 5.14.2 Function Documentation

#### 5.14.2.1 int map\_peripheral ( struct bcm2835\_peripheral \* p )

This function maps a GPIO port for access in the software. Exposes the physical address defined in the passed structure using mmap on /dev/mem.

## Parameters

<i>p</i>	Pointer to the GPIO port structure.
----------	-------------------------------------

5.14.2.2 void unmap\_peripheral ( struct bcm2835\_peripheral \* *p* )

This function unmaps a GPIO port.

## Parameters

<i>p</i>	Pointer to the GPIO port structure.
----------	-------------------------------------

## 5.15 rpi\_gpio\_header.h File Reference

MSP430 header file.

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

## Classes

- struct [bcm2835\\_peripheral](#)

## Macros

- #define [BCM2708\\_PERI\\_BASE](#) 0x20000000  
*Physical address at which the peripheral registers start.*
- #define [GPIO\\_BASE](#) ([BCM2708\\_PERI\\_BASE](#) + 0x200000)  
*Address of the GPIO controller (expressed as an offset with respect to [BCM2708\\_PERI\\_BASE](#))*
- #define **BLOCK\_SIZE** (4\*1024)
- #define [INP\\_GPIO](#)(g) \*(gpio.addr + ((g)/10)) &= ~(7<<(((g)%10)\*3))  
*Set pin as input.*
- #define [GPIO\\_READ](#)(g) \*(gpio.addr + 13) &= (1<<(g))  
*Read an input pin's state.*

## Variables

- struct [bcm2835\\_peripheral](#) [gpio](#)  
*The GPIO port access variable.*

## 5.15.1 Detailed Description

MSP430 header file.

## Author

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This is the header to [rpi\\_gpio\\_funcs.c](#) containing necessary definitions and initializations. This header is a slightly reduced version (to the bare bones that the GNC program needs) of the header provided by Pieter-Jan Van de Maele [here](#).

**5.15.2 Variable Documentation****5.15.2.1 struct bcm2835\_peripheral gpio**

The GPIO port access variable.

The GPIO port access variable.

**5.16 simplex\_funcs.c File Reference**

Simplex functions file.

```
#include <stdio.h>
#include <math.h>
#include "simplex_header.h"
```

**Functions**

- void [simplex](#) ([MAT](#) a, int m, int n, int m1, int m2, int m3, int \*icase, int \*izrov, int \*iposv)
- void [simp1](#) ([MAT](#) a, int mm, int \*ll, int nll, int iabf, int \*kp, [REAL](#) \*bmax)
- void [simp2](#) ([MAT](#) a, int m, int n, int \*l2, int nl2, int \*ip, int kp, [REAL](#) \*q1)
- void [simp3](#) ([MAT](#) a, int i1, int k1, int ip, int kp)
- void [get\\_simplex\\_solution](#) (int ICASE, int \*IPOSV, [MAT](#) A, int M, int N, double \*R1, double \*R2, double \*R3, double \*R4)

**5.16.1 Detailed Description**

Simplex functions file.

**Author**

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This file contains Simplex linear programming (linear optimization) functions that are used in optimally allocating thrusts between the 4 valves given Fpitch, Fyaw and Mroll that we desire to exert on the rocket. This code is taken from the kindly provided code by Jean-Pierre Moreau [here](#). where only [get\\_simplex\\_solution\(\)](#) function is new (i.e. not found at the above site) and is used to conveniently extract the optimal solution directly into the valve thrusts R1, R2, R3, R4 that are defined in the GNC program (see [master\\_header.h](#)).

License from the original .cpp file:

```

*****
*          LINEAR PROGRAMMING: THE SIMPLEX METHOD          *
*-----*
*-----*
* Reference: "Numerical Recipes By W.H. Press, B. P. Flannery,*
*           S.A. Teukolsky and W.T. Vetterling, Cambridge *
*           University Press, 1986" [BIBLI 08].           *
*-----*
*                   C++ Release 1.0 By J-P Moreau, Paris  *
*                   (www.jpmoreau.fr)                     *
*****

```

## 5.16.2 Function Documentation

**5.16.2.1 void get\_simplex\_solution ( int *ICASE*, int \* *IPOSV*, MAT *A*, int *M*, int *N*, double \* *R1*, double \* *R2*, double \* *R3*, double \* *R4* )**

This function writes the result of the Simplex optimization into R1, R2, R3 and R4 (the valve thrusts).

### Parameters

<i>A</i>	Simplex table.
<i>M</i>	Total number of constraints.
<i>N</i>	Total number of variables in cost function.
<i>R1</i>	Pointer to the memory holding the R1 valve thrust.
<i>R2</i>	Pointer to the memory holding the R2 valve thrust.
<i>R3</i>	Pointer to the memory holding the R3 valve thrust.
<i>R4</i>	Pointer to the memory holding the R4 valve thrust.

**5.16.2.2 void simp1 ( MAT *a*, int *mm*, int \* *ll*, int *nll*, int *iabf*, int \* *kp*, REAL \* *bmax* )**

Determines the maximum of those elements whose index is contained in the supplied list ll, either with or without taking the absolute value, as flagged by iabf.

### Parameters

<i>a</i>	Simplex table.
----------	----------------

**5.16.2.3 void simp2 ( MAT *a*, int *m*, int *n*, int \* *l2*, int *nl2*, int \* *ip*, int *kp*, REAL \* *q1* )**

Locate a pivot element, taking degeneracy into account.

### Parameters

<i>a</i>	Simplex table.
----------	----------------

**5.16.2.4 void simp3 ( MAT *a*, int *i1*, int *k1*, int *ip*, int *kp* )**

Matrix operations to exchange a left-hand and right-hand variable (see text).

**5.16.2.5 void simplx ( MAT *a*, int *m*, int *n*, int *m1*, int *m2*, int *m3*, int \* *icase*, int \* *izrov*, int \* *iposv* )**

USES simp1,simp2,simp3 Simplex method for linear programming. Input parameters a, m, n, mp, np, m1, m2, and m3, and output parameters a, icase, izrov, and iposv are described above (see reference). Parameters: MMAX is the maximum number of constraints expected; NMAX is the maximum number of variables expected; EPS is the absolute precision, which should be adjusted to the scale of your variables.

## Parameters

$a$	Simplex table.
$m$	Total number of constraints ( $m=m1+m2+m3$ ).
$n$	Number of variables in cost function.
$m1$	Number of ( $\leq$ ) type inequality constraints.
$m2$	Number of ( $\geq$ ) type inequality constraints.
$m3$	Number of ( $=$ ) type constraints.

## 5.17 simplex\_header.h File Reference

Simplex header file.

## Macros

- `#define MMAX 5`  
*Number of rows of the simplex table.*
- `#define NMAX 6`  
*Number of columns of the simplex table.*
- `#define REAL double`  
*Alias for a double.*

## Typedefs

- `typedef REAL MAT [MMAX][NMAX]`  
*A [MMAXxNMAX] matrix.*

## Variables

- `MAT A`  
*Simplex table.*
- `int IPOSV [MMAX]`
- `int IZROV [NMAX]`
- `int i`  
*Index variable for loops.*
- `int j`  
*Index variable for loops.*
- `int ICASE`
- `int N`  
*Number of variables in cost function. Our variables are R1, R2, R3, R4 so N=4.*
- `int M`  
*Total number of constraints ( $M=M1+M2+M3$ )*
- `int M1`  
*No ( $\leq$ ) type constraints.*
- `int M2`  
*No ( $\geq$ ) type constraints.*
- `int M3`  
*3 ( $=$ ) type constraints (for Fpitch, Fyaw, Mroll)*



### 5.17.1 Detailed Description

Simplex header file.

#### Author

Simplex header file [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

#### Version

1.0

This is the header to [simplex\\_funcs.c](#) containing necessary definitions and initializations. This header supports, but was not provided with, the code kindly provided code by Jean-Pierre Moreau [here](#). where the header definitions in this file were directly incorporated into the code at the above url.

License from the original cpp file:

```
*****
*          LINEAR PROGRAMMING: THE SIMPLEX METHOD          *
*-----*
*-----*
* Reference: "Numerical Recipes By W.H. Press, B. P. Flannery,*
*           S.A. Teukolsky and W.T. Vetterling, Cambridge *
*           University Press, 1986" [BIBLI 08].           *
*-----*
*           C++ Release 1.0 By J-P Moreau, Paris          *
*           (www.jpmoreau.fr)                             *
*****
```

## 5.18 spycam\_funcs.c File Reference

Raspberry Pi spy camera functions file.

```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "spycam_header.h"
```

### Functions

- void [startVideo](#) (char \*filename, char \*options)
- void [stopVideo](#) (void)

### Variables

- pid\_t [pid](#) =0

*Process ID variable for the Raspberry Pi spy camera video recording.*

### 5.18.1 Detailed Description

Raspberry Pi spy camera functions file.

**Author**

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This file contains functions to start and stop camera recording with particular settings. The code is taken directly from the very kindly provided code by ceptimus [here](#).

No license file nor text was provided with this source code, so none is included here.

**5.18.2 Function Documentation****5.18.2.1 void startVideo ( char \* filename, char \* options )**

This function starts the Raspberry Pi camera recording with specific recording options \*options stored at the adress pointed to by options. If you want to enable preview/monitoring then make the obvious change to remove the -n (no preview) option from the source code of this function.

**Parameters**

<i>filename</i>	A string specifying the name of the resulting video file (must have .h264 ending).
<i>options</i>	Normal raspivid options. Avoid -t, -n, -o, and -s as the code fills those in for you.

**5.18.2.2 void stopVideo ( void )**

This functions stops the camera recording process.

**5.19 spycam\_header.h File Reference**

Raspberry Pi spy camera header file.

**Variables**

- `pid_t pid`

*Process ID variable for the Raspberry Pi spy camera video recording.*

**5.19.1 Detailed Description**

Raspberry Pi spy camera header file.

**Author**

Danylo Malyuta [danylo.malyuta@gmail.com](mailto:danylo.malyuta@gmail.com)

**Version**

1.0

This is the header to [spycam\\_funcs.c](#) containing necessary function and variable declarations.

# Index

- bcm2835\_peripheral, [7](#)
- Calibrate\_IMU
  - imu\_funcs.c, [14](#)
- check\_time
  - master\_funcs.c, [26](#)
- close\_port
  - imu\_funcs.c, [14](#)
- construct\_zeroed\_DCM
  - imu\_funcs.c, [14](#)
- control\_funcs.c, [11](#)
  - Fpitch\_loop\_control\_setup, [11](#)
  - Fyaw\_loop\_control\_setup, [11](#)
  - Mroll\_loop\_control\_setup, [12](#)
- control\_header.h, [12](#)
- Control\_loop, [7](#)
- Find\_raw\_Euler\_angular\_velocities
  - imu\_funcs.c, [14](#)
- Fpitch\_loop\_control\_setup
  - control\_funcs.c, [11](#)
- Fyaw\_loop\_control\_setup
  - control\_funcs.c, [11](#)
- get\_filtered\_attitude\_parallel
  - imu\_funcs.c, [15](#)
- get\_old\_attr
  - imu\_funcs.c, [15](#)
- get\_readings\_SPI\_parallel
  - pressure\_funcs.c, [36](#)
- get\_simplex\_solution
  - simplex\_funcs.c, [41](#)
- gpio
  - master.c, [25](#)
  - rpi\_gpio\_header.h, [40](#)
- imu\_funcs.c, [13](#)
  - Calibrate\_IMU, [14](#)
  - close\_port, [14](#)
  - construct\_zeroed\_DCM, [14](#)
  - Find\_raw\_Euler\_angular\_velocities, [14](#)
  - get\_filtered\_attitude\_parallel, [15](#)
  - get\_old\_attr, [15](#)
  - Kalman\_filter, [15](#)
  - min\_of\_set, [15](#)
  - open\_serial\_port, [16](#)
  - read\_IMU\_parallel, [16](#)
  - reset\_old\_attr\_port, [16](#)
  - set\_new\_attr, [16](#)
  - set\_to\_blocking, [17](#)
  - TO\_DEG, [17](#)
  - Treat\_reply, [17](#)
  - zero\_Euler\_angles, [17](#)
- imu\_header.h, [17](#)
- initMatrix
  - la\_funcs.c, [21](#)
- Kalman\_filter
  - imu\_funcs.c, [15](#)
- la\_funcs.c, [20](#)
  - initMatrix, [21](#)
  - madd, [21](#)
  - minverse\_1x1, [21](#)
  - mmultiply, [21](#)
  - msubtract, [22](#)
  - transpose, [22](#)
- la\_header.h, [22](#)
- MATRIX, [8](#)
- MSP430\_UART\_receive
  - mmsp430\_funcs.c, [33](#)
- MSP430\_UART\_write
  - mmsp430\_funcs.c, [33](#)
- MSP430\_UART\_write\_PWM
  - mmsp430\_funcs.c, [33](#)
- madd
  - la\_funcs.c, [21](#)
- main
  - master.c, [25](#)
- map\_peripheral
  - rpi\_gpio\_funcs.c, [38](#)
- master.c, [22](#)
  - gpio, [25](#)
  - main, [25](#)
- master\_funcs.c, [26](#)
  - check\_time, [26](#)
  - open\_error\_file, [27](#)
  - open\_file, [27](#)
  - passive\_wait, [27](#)
  - write\_to\_file\_custom, [27](#)
- master\_header.h, [28](#)
- min\_of\_set
  - imu\_funcs.c, [15](#)
- minverse\_1x1
  - la\_funcs.c, [21](#)
- mmultiply
  - la\_funcs.c, [21](#)
- Mroll\_loop\_control\_setup
  - control\_funcs.c, [12](#)

- msp430\_funcs.c, [32](#)
  - MSP430\_UART\_receive, [33](#)
  - MSP430\_UART\_write, [33](#)
  - MSP430\_UART\_write\_PWM, [33](#)
- msp430\_header.h, [34](#)
- msubtract
  - la\_funcs.c, [22](#)
- open\_error\_file
  - master\_funcs.c, [27](#)
- open\_file
  - master\_funcs.c, [27](#)
- open\_serial\_port
  - imu\_funcs.c, [16](#)
- passive\_wait
  - master\_funcs.c, [27](#)
- pressure\_funcs.c, [35](#)
  - get\_readings\_SPI\_parallel, [36](#)
  - pressure\_sensor\_SPI\_connect, [37](#)
- pressure\_header.h, [37](#)
- pressure\_sensor\_SPI\_connect
  - pressure\_funcs.c, [37](#)
- read\_IMU\_parallel
  - imu\_funcs.c, [16](#)
- reset\_old\_attr\_port
  - imu\_funcs.c, [16](#)
- rpi\_gpio\_funcs.c, [38](#)
  - map\_peripheral, [38](#)
  - unmap\_peripheral, [39](#)
- rpi\_gpio\_header.h, [39](#)
  - gpio, [40](#)
- SPI\_data, [8](#)
- set\_new\_attr
  - imu\_funcs.c, [16](#)
- set\_to\_blocking
  - imu\_funcs.c, [17](#)
- simp1
  - simplex\_funcs.c, [41](#)
- simp2
  - simplex\_funcs.c, [41](#)
- simp3
  - simplex\_funcs.c, [41](#)
- simplex\_funcs.c, [40](#)
  - get\_simplex\_solution, [41](#)
  - simp1, [41](#)
  - simp2, [41](#)
  - simp3, [41](#)
  - simplx, [41](#)
- simplex\_header.h, [42](#)
- simplx
  - simplex\_funcs.c, [41](#)
- spycam\_funcs.c, [43](#)
  - startVideo, [44](#)
  - stopVideo, [44](#)
- spycam\_header.h, [44](#)
- startVideo
  - spycam\_funcs.c, [44](#)
- stopVideo
  - spycam\_funcs.c, [44](#)
- TO\_DEG
  - imu\_funcs.c, [17](#)
- transpose
  - la\_funcs.c, [22](#)
- Treat\_reply
  - imu\_funcs.c, [17](#)
- unmap\_peripheral
  - rpi\_gpio\_funcs.c, [39](#)
- VALVE\_\_MAX\_THRUST
  - control\_funcs.c, [12](#)
  - master\_header.h, [32](#)
- write\_to\_file\_custom
  - master\_funcs.c, [27](#)
- zero\_Euler\_angles
  - imu\_funcs.c, [17](#)