FARGNC
FALCO-4

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

Authors

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

Project made possible thanks to:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

eSpace SPACE ENGINEERING CENTER

Generated by Doxygen 1.8.6

Sun May 31 2015

Contents

Chapter 1

Project description

Author

Danylo Malyuta 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 stabiilize a model rocket with an (R)eaction (C)ontrol (S)system. 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)controlled (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

Chapter 2

FlyARocket_GNC

Guidance, Navigation and Control software for the Fly-A-Rocket model rocket active vertical stabilization Reaction Control System.

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Class Documentation

5.1 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.

5.1.1 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

5.2 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?

5.2.1 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

5.3 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.

5.3.1 Detailed Description

This is the structure for a Matrix.

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

- la_header.h

5.4 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.

5.4.1 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 6

File Documentation

6.1 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!

6.1.1 Detailed Description

Control functions source file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This file contains functions regarding the control algorithm.

6.1.2 Function Documentation

6.1.2.1 void Fpitch_loop_control_setup ( )

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

6.1.2.2 void Fyaw_loop_control_setup ( )

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

6.1.2.3 void Mroll_loop_control_setup ( )

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

6.1.3 Variable Documentation

6.1.3.1 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)

6.2 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.

6.2.1 Detailed Description

Control header file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This is the header to control_funcs.c containing necessary definitions and initializations.

6.3 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_SYNCHED =0

  If =1, then the IMU and Raspberry Pi UART communication has been synced, =0 otherwise.
- unsigned char IMU_TX [2] ="#f"

  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.

6.3.1 Detailed Description

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

Author

Danylo Malyuta 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.

6.3.2 Function Documentation

6.3.2.1 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).

6.3.2.2 void close_port (int *fd*)

This function closes the serial connection identified by handle fd.

Parameters

fd	The file handle.

6.3.2.3 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 =[I] <==> the IMU is in the orientation at which it was calibrated ("zeroed").

6.3.2.4 void Find_raw_Euler_angular_velocities ( )

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

6.3.2.5 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 interation 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

args	A pointer to the input arguments (we have none for this thread)

6.3.2.6 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

fd	The file handle.
old_options	the old UART connection attributes.

6.3.2.7 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

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

6.3.2.8 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 degress were used to facilitate understanding; the GNC program works in radians directly!

Parameters

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

6.3.2.9 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

fd	Pointer to the file handle.
directory	Pointer to the string which defines the directory.

6.3.2.10 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

args	A pointer to the input arguments (we have none for this thread)

6.3.2.11 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

fd	The file handle.
old_options	the old UART connection attributes.

6.3.2.12 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

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

6.3.2.13 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

fd	The file handle

6.3.2.14 float TO_DEG (float *angle*)

Convert from radians to degrees

6.3.2.15 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

comparison_string	The string that the user must enter.

6.3.2.16 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).

6.4 imu_header.h File Reference

IMU header file.

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

Macros

- #define MAX_BUFFER 24

    The max buffer size for receving data from IMU.

Variables

- unsigned char IMU_RX [MAX_BUFFER]

    Buffer holding received values via UART from Razor IMU.

- char IMU_SYNCH_RECEIVE [1]

  Buffer used for receving the 2-character (2-byte) synch token from the IMU during sychronization.
- 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.

6.4.1 Detailed Description

IMU header file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This is the header to imu_funcs.c containing necessary definitions and initializations.

6.5 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)

6.5.1 Detailed Description

Lienar Algebra functions source file.

Author

Danylo Malyuta 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)

6.5.2 Function Documentation

6.5.2.1 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

rows	Number of matrix rows.
cols	Number of matrix columns.

6.5.2.2 struct MATRIX madd (struct MATRIX *A*, struct MATRIX *B*)

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

Parameters

A	The first matrix.
B	The second matrix.

6.5.2.3 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 ([1x2]∗[2x1])+[1x1]^(-1).

Parameters

A	[1x1] matrix.

6.5.2.4 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

A	The pre-multiplied matrix

B	The post-multiplied matrix

6.5.2.5 struct MATRIX msubtract (struct MATRIX *A*, struct MATRIX *B*)

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

Parameters

A	The first matrix.
B	The second matrix.

6.5.2.6 struct MATRIX transpose (struct MATRIX *A*)

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

Parameters

A	A matrix.

6.6 la_header.h File Reference

Linear Algebra header file.

Classes

- struct MATRIX

6.6.1 Detailed Description

Linear Algebra header file.

Author

Danylo Malyuta `danylo.malyuta@gmail.com`

Version

1.0

This is the header to la_funcs.c containing necessary definitions and initializations.

6.7 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 boundt 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 (<=) type constraints.
- int M2 =0

  No (>=) 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 tempeartures 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 umbillical 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 frequence, 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 steay, 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].

6.7.1 Detailed Description

Master GNC source file (main function file).

Author

Danylo Malyuta `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.

6.7.2 Function Documentation

6.7.2.1 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.

6.7.3 Variable Documentation

6.7.3.1 struct **bcm2835_peripheral** gpio = {**GPIO_BASE**}

Our access register to the Raspberry Pi's GPIOs.

The GPIO port access variable.

6.8 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.

6.8.1 Detailed Description

Master functions file.

Author

Danylo Malyuta `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.

6.8.2 Function Documentation

6.8.2.1 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

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

6.8.2.2 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

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

6.8.2.3 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

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

6.8.2.4 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

now	The time now (struct timeval).
before	The time before (struct timeval).
elapsed	The time elapsed (struct timeval).
time	The time in [us] between now and before.
TIME__STEP	The time we want to iterate at within the loop from which passive_wait() was called.

6.8.2.5 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

file_ptr	Pointer to file.
string	The file path.
error_log	Pointer to error log file.

6.9 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_SYNCHED

  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 x_dot=A_kalman∗x ; y=C_kalman∗x, but in discrete time!
- struct MATRIX C_kalman

  Temporary matrix for Kalman filtering, dynamic equation x_dot=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/tempearture 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 pointes to files we use for recording flight data.

- FILE ∗ error_log

  Error log (stores errors)
- FILE ∗ pressure_log

  Pressure log (stores pressures and tempeartures 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.

6.9.1 Detailed Description

Master header file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This is the header to master_funcs.c containing necessary definitions and initializations.

6.9.2 Variable Documentation

6.9.2.1 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)

6.10 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)

6.10.1 Detailed Description

MSP430 functions file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This file contains functions necessary for communicatin with the MSP430 salve 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.

6.10.2 Function Documentation

6.10.2.1 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 successfuly 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.

6.10.2.2 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

MSP430_TX	Contains the 3-byte (3-character) string to send to the MSP430

6.10.2.3 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!

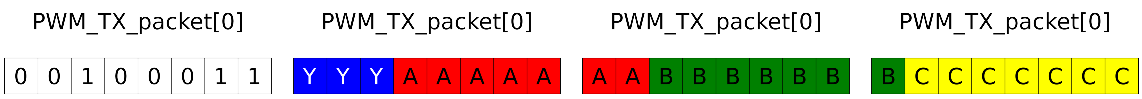

Figure 6.1: 4 byte packet send by Raspberry Pi to MSP430 to update PWM values

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 so from PWM1 to PWM4, skipping the PWM that is 0.

6.11 msp430_header.h File Reference

MSP430 header file.

Macros

- #define MSP430_MAX_BUFFER 1

  Buffer size for receving 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 restitute them before closing the connection at the end of the program.

6.11.1 Detailed Description

MSP430 header file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This is the header to msp430_funcs.c containing necessary definitions and initializations.

6.12 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.

6.12.1 Detailed Description

Honeywell pressure/temperature sensors functions file.

Author

Danylo Malyuta 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.

6.12.2 Function Documentation

6.12.2.1 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

args	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 `example`).

6.12.2.2 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

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

6.13 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.

6.13.1 Detailed Description

Honeywell pressure/temperature sensors header file.

Author

Danylo Malyuta `danylo.malyuta@gmail.com`

Version

1.0

This is the header to pressure_funcs.c containing necessary definitions and initializations.

6.14 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)

6.14.1 Detailed Description

GPIO functions file.

Author

Danylo Malyuta `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`.

6.14.2 Function Documentation

6.14.2.1 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

p	Pointer to the GPIO port structure.

6.14.2.2 void unmap_peripheral (struct bcm2835_peripheral * *p*)

This function unmaps a GPIO port.

Parameters

p	Pointer to the GPIO port structure.

6.15 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 (expresset 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.

6.15.1 Detailed Description

MSP430 header file.

Author

Danylo Malyuta 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`.

6.15.2 Variable Documentation

6.15.2.1 struct **bcm2835_peripheral** gpio

The GPIO port access variable.

The GPIO port access variable.

6.16 simplex_funcs.c File Reference

Simplex functions file.

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

Functions

- void simplx (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)

6.16.1 Detailed Description

Simplex functions file.

Author

Danylo Malyuta `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)           *
**************************************************************
```

6.16.2 Function Documentation

6.16.2.1 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

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

6.16.2.2 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

a	Simplex table.

6.16.2.3 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

a	Simplex table.

6.16.2.4 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).

6.16.2.5 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 (<=) type inequality constraints.
m2	Number of (>=) type inequality constraints.
m3	Number of (=) type constraints.

6.17 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 (<=) type constraints.
- int M2

  No (>=) type constraints.
- int M3

  3 (=) type constraints (for Fpitch, Fyaw, Mroll)

6.17.1 Detailed Description

Simplex header file.

Author

Simplex header file 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)          *
***************************************************************
```

6.18 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.

6.18.1 Detailed Description

Raspberry Pi spy camera functions file.

Author

Danylo Malyuta 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.

6.18.2 Function Documentation

6.18.2.1 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

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

6.18.2.2 void stopVideo (void)

This functions stops the camera recording process.

6.19 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.

6.19.1 Detailed Description

Raspberry Pi spy camera header file.

Author

Danylo Malyuta danylo.malyuta@gmail.com

Version

1.0

This is the header to spycam_funcs.c containing necessary function and variable declarations.

Index