Velocity Determination of a Quad-Rotor UAV

Software Manual

Generated by Doxygen 1.8.8

Wed Apr 6 2016 17:45:37

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

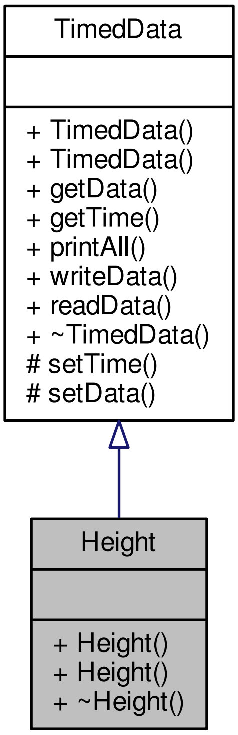
Chapter 4

Data Structure Documentation

4.1 Height Class Reference

```
#include <Height.hpp>
```

Inheritance diagram for Height:



Public Member Functions

- Height ()
- Height (float height, microseconds time)

- ∼Height (void)

Additional Inherited Members

4.1.1 Detailed Description

Store height data and time-stamp

Note

Inherits from the TimedData class

See also

TimedData.hpp

4.1.2 Constructor & Destructor Documentation

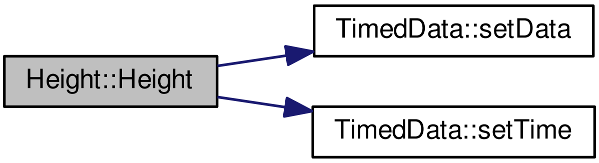
4.1.2.1 Height::Height ( )

Construct an empty Height object

Postcondition

an empty Height object is created

Here is the call graph for this function:



4.1.2.2 Height::Height (float *height*, microseconds *time*)

Construct a Height object containing a height and time

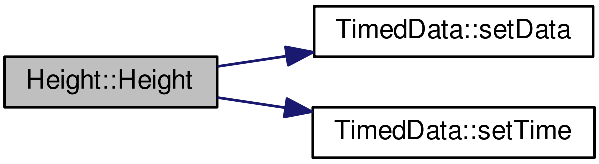
Parameters

`in`	*height*	the height of the object
`in`	*time*	the height reading was taken

Postcondition

A Height object containing a height reading and the corresponding time-stamp

Here is the call graph for this function:



4.1.2.3 Height::∼Height (void)

Destructor for Height object

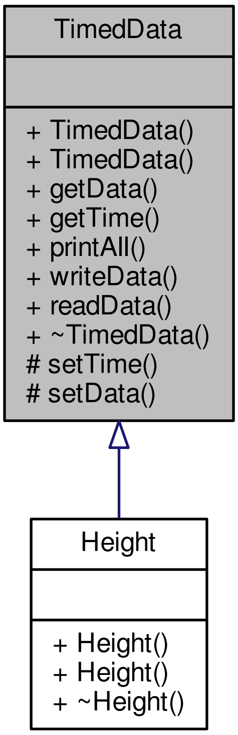
The documentation for this class was generated from the following files:

- Height.hpp
- Height.cpp

4.2 TimedData Class Reference

```
#include <TimedData.hpp>
```

Inheritance diagram for TimedData:



Public Member Functions

- TimedData ()
- TimedData (float data, microseconds time)
- float getData ()
- microseconds getTime ()
- void printAll ()

  print contents of TimedData to console
- void writeData (string filename, TimedData ∗h)
- void readData (string filename, TimedData ∗h)
- ∼TimedData (void)

Protected Member Functions

- void setTime (microseconds t)
- void setData (float d)

4.2.1 Detailed Description

Store TimedData data and time-stamp

4.2.2 Constructor & Destructor Documentation

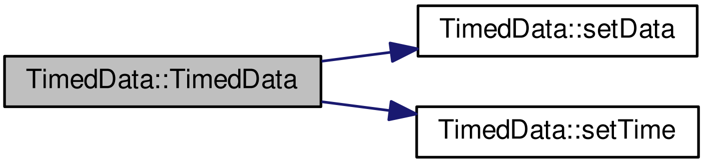
4.2.2.1 TimedData::TimedData ( )

Construct an empty TimedData object

Postcondition

an empty TimedData object is created

Here is the call graph for this function:



4.2.2.2 TimedData::TimedData (float *data*, microseconds *time*)

Construct a TimedData object with provided parameters

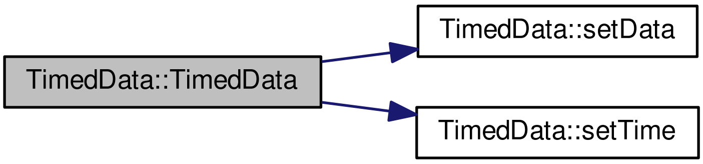
Parameters

in	*time*	the data was taken
in	*data*	any data

Postcondition

A TimedData object containing a TimedData reading and the corresponding time-stamp

Here is the call graph for this function:



4.2.2.3 TimedData::∼TimedData (void)

Destructor for TimedData object

4.2.3 Member Function Documentation

4.2.3.1 float TimedData::getData ( )

Returns

Data held in TimedData

4.2.3.2 microseconds TimedData::getTime ( )

Returns

time of Data

4.2.3.3 void TimedData::readData (string *filename,* TimedData ∗ *h*)

read TimedData object from file

Parameters

in	*filename*	name of file to be read
in	*h*	pointer to the TimedData object being read

Precondition

A file containing at least one TimedData object stored in binary
A TimedData object to store the data from file

Postcondition

A TimedData object containing data from the file being read

4.2.3.4 void TimedData::setData (float *d*) [protected]

set Data member

Parameters

in	*d*	the data

Postcondition

the Data member is set

4.2.3.5 void TimedData::setTime (microseconds *t*) [protected]

set time member

Parameters

in	*t*	the timestamp of the data

Postcondition

time member is set to t

4.2.3.6 void TimedData::writeData (string *filename*, TimedData ∗ *h*)

Save TimedData object to file

Parameters

`in`	*filename*	name of file to be written
`in`	*h*	pointer to the TimedData object being written

Precondition

An existing TimedData object

Postcondition

Object is saved to a binary file on disk

The documentation for this class was generated from the following files:

- TimedData.hpp
- TimedData.cpp

4.3 VerticalData Class Reference

Store two Height objects and calculate vertical velocity.

```
#include <VerticalData.hpp>
```

Public Member Functions

- void storeHeight (Height &ht)

  Store height object.
- void placeHeight (float height, microseconds time)

  create Height object and place in queue
- float getVelocity ()
- void setVelocity (float vel)
- void printAll ()

  Print contents of VerticalData object.

Protected Member Functions

- void calculateVelocity ()

  Calculate the vertical velocity.

4.3.1 Detailed Description

Store two Height objects and calculate vertical velocity.

4.3.2 Member Function Documentation

4.3.2.1 float VerticalData::getVelocity ( )

Returns

vertical velocity

4.3.2.2 void VerticalData::placeHeight (float *height,* microseconds *time*)

create Height object and place in queue

Parameters

in	*height*	height of UAV in meters
in	*time*	timestamp of height in microseconds

4.3.2.3 void VerticalData::setVelocity (float *vel*)

Parameters

vel	the velocity of the UAV in m/s

4.3.2.4 void VerticalData::storeHeight (Height & *ht*)

Store height object.

Parameters

in	*ht*	the height of the UAV in meters

The documentation for this class was generated from the following files:

- VerticalData.hpp
- VerticalData.cpp

Chapter 5

File Documentation

5.1 mega_sensor.h File Reference

get sensor data

```
#include <unistd.h>
#include <math.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <time.h>
#include <wiringPi.h>
#include "../include/lidarLite.h"
#include "../includes/sensor.h"
#include "../includes/output.h"
#include "../includes/bmp180.h"
```

Functions

- void MEGA_SENSOR (float ∗g_x, float ∗g_y, float ∗a_x, float ∗a_y, float ∗t, long ∗p, int ∗l, float ∗l_c, long ∗b_c, float ∗h, int fd)

  extracts data from sensors

5.1.1 Detailed Description

get sensor data

Author

Luke Protz

Date

March 27, 2016

5.1.2 Function Documentation

5.1.2.1 void MEGA_SENSOR (float ∗ *g_x*, float ∗ *g_y*, float ∗ *a_x*, float ∗ *a_y*, float ∗ *t*, long ∗ *p*, int ∗ *l*, float ∗ *l_c*, long ∗ *b_c*, float ∗ *h*, int *fd*)

extracts data from sensors

detemines most accurate data based sensors being used

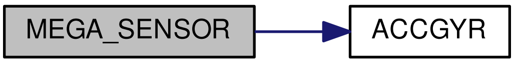
Precondition

The Lidar is initialized
The IMU is initialized

Parameters

`out`	*g_x*	pitch velocity in rad/s
`out`	*g_y*	roll velocity in rad/s
`out`	*a_x*	pitch in radians
`out`	*a_y*	roll in radians
`out`	*t*	temperature in degrees Celcius
`out`	*p*	pressure in pascales
`out`	*l*	distance from lidar in cm
`out`	*l_c*	height calculation held when uav reaches 40 m
`out`	*b_c*	pressure measurement held uav reaches 40 m
`out`	*h*	calculated height, corrected for attitude
`out`	*fd*	confirmation of lidar functionality, equals 1 when lidar operational

Here is the call graph for this function:



5.2 output.h File Reference

Interface with lidar manufacturer API.

```
#include "sensor.h"
#include <unistd.h>
#include <math.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <time.h>
#include <wiringPi.h>
```

Functions

- void INThandler (int sig)

Initializes Lidar.

- void ACCGYR (float ∗gyr_x, float ∗gyr_y, float ∗acc_x, float ∗acc_y)

  gathers information from IMU (accelerometer and gyroscope)

5.2.1 Detailed Description

Interface with lidar manufacturer API.

Author

Luke Protz

Date

March 25, 2016

5.2.2 Function Documentation

5.2.2.1 void ACCGYR (float ∗ *gyr_x,* float ∗ *gyr_y,* float ∗ *acc_x,* float ∗ *acc_y*)

gathers information from IMU (accelerometer and gyroscope)

Parameters

out	*gyr_x*	gyroscope pitch value
out	*gyr_y*	gyroscope roll value
out	*acc_x*	accelerometer pitch value
out	*acc_y*	accelerometer roll value

Postcondition

parameters contain most recent attitude measurements

5.2.2.2 void INThandler (int *sig*)

Initializes Lidar.

Parameters

in	*sig*	wake signal

Postcondition

lidar is operational

5.3 velocityCalculate.hpp File Reference

Calculate the Horizontal Velocity of a UAV.

```
#include <opencv2/video/tracking.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <ctype.h>
#include <iomanip>
#include <random>
```

Functions

- void fourPixelAverage (InputArray imgFull, OutputArray imgReduced)

  average four pixels
- float entropy (Mat seq, Size size, int index)

  calculate the entropy of an image
- Mat myEntropy (Mat seq, int histSize)

  calculates relative occurrence of different symbols within given input sequence using histogram
- void calculateAEAO (Mat prevGray, Mat nextGray, double cameraElevation, int frameRate, double pitch, double roll, double wPitch, double wRoll, double &Vx, double &Vy, double Vz, double &speed, double &direction)

  Calculates velocity in the x and y directions,speed and direction of an unmanned aerial vehicle(UAV).

5.3.1 Detailed Description

Calculate the Horizontal Velocity of a UAV.

Date

Mar 26, 2016

Author

Lance Pitka
Devon Haubold

5.3.2 Function Documentation

5.3.2.1 void calculateAEAO (Mat *prevGray,* Mat *nextGray,* double *cameraElevation,* int *frameRate,* double *pitch,* double *roll,* double *wPitch,* double *wRoll,* double & *Vx,* double & *Vy,* double *Vz,* double & *speed,* double & *direction*)

Calculates velocity in the x and y directions,speed and direction of an unmanned aerial vehicle(UAV).

Calculates confidence in calculations

Precondition

Two images, prevGray and nextGray, taken successively at a specified frame rate
Pitch, Height, Vz, Roll, change in pitch, change in Roll, at the time the images are captured
Vx, Vy, speed and Direction are declared

Parameters

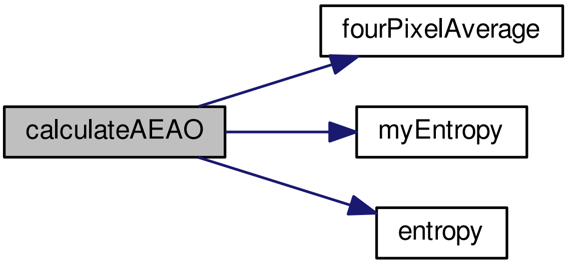
in	*prevGray*	the first single channel 8bit image
in	*nextGray*	the second single channel 8bit image
in	*cameraElevation*	the distance of the UAV from the ground
in	*frameRate*	rate at which the images are being captured in fps
in	*pitch*	the pitch of the UAV in radians
in	*roll*	the roll of the UAV in radians
in	*wPitch*	the UAV angular pitch velocity in radians/sec
in	*wRoll*	the UAV angular roll velocity in radians/sec
out	*Vx*	UAV velocity in the x direction

`out`	*Vy*	UAV velocity in the y direction
`in,out`	*Vz*	UAV Velocity in the z direction (vertical velocity) in m/s
`out`	*speed*	the speed of UAV in m/s
`out`	*direction*	direction of the UAV in radians relative to itself.

Postcondition

Vx contains the most recent velocity in the x direction
Vy contains the most recent Velocity in the y direction
speed contains the most recent speed
direction contains the most recent direction

Here is the call graph for this function:



5.3.2.2 float entropy (Mat *seq*, Size *size*, int *index*)

calculate the entropy of an image

Parameters

`in`	*seq*	a single channel 8 bit image
`in`	*size*	dimensions of image
`in`	*index*	pixel location in image

Returns

the entropy of the image at index

5.3.2.3 void fourPixelAverage (InputArray *imgFull*, OutputArray *imgReduced*)

average four pixels

Parameters

`in`	*imgFull*	the image to be averaged
`out`	*imgReduced*	an averaged image

5.3.2.4 Mat myEntropy (Mat *seq*, int *histSize*)

calculates relative occurrence of different symbols within given input sequence using histogram

Parameters

in	*seq*	a single channel 8 bit image
in	*histSize*	size of the histogram

Returns

histogram showing occurence of symbols in a sequence

Compute the histograms:

5.4 velocityTracking.cpp File Reference

The main program, stores data and calculates velocities.

```
#include "../includes/Height.hpp"
#include "../includes/VerticalData.hpp"
#include "../includes/velocityCalculate.hpp"
#include "../includes/output.h"
#include "../includes/sensor.h"
#include "../includes/bmp180.h"
#include "../includes/lidarLite.h"
#include "../includes/mega_sensor.h"
#include <iostream>
#include <cstdlib>
#include <thread>
#include <chrono>
#include <ctime>
#include <opencv2/highgui/highgui.hpp>
#include <raspicam/raspicam_cv.h>
```

Macros

- #define FRAME_RATE 90

Functions

- void initializeCamera (raspicam::RaspiCam_Cv &Camera)
- void twoImageCapture (Mat &image_1, Mat &image_2, bool &exit_flag, bool &ready_flag, bool &wait_flag)

  capture two consecutive images at 90 frames per second, loops until exit_flag is true
- void heightReporting (VerticalData &vertDataRef, bool &exit_flag, int lidar)
- int main (int argc, char ∗argv[])

5.4.1 Detailed Description

The main program, stores data and calculates velocities.

Date

Mar 21, 2016

Author

: Devon Haubold

5.4.2 Macro Definition Documentation

5.4.2.1 #define FRAME_RATE 90

The frame rate that the camera is set at

5.4.3 Function Documentation

5.4.3.1 void heightReporting (VerticalData & *vertDataRef,* bool & *exit_flag,* int *lidar*)

Get height data continuosly, calculate velocity

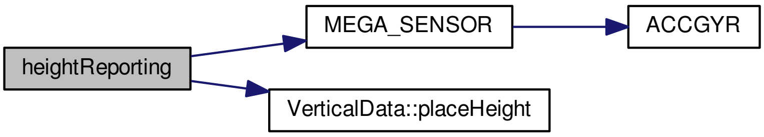
Precondition

a thread has been created and id assigned
the sensors have been initialized and are accessible

Parameters

`in`	*vertDataRef*	holds height data and calculates vertical velocity
`in`	*exit_flag*	flag to alert thread to exit
`in`	*lidar*	equals 1 if lidar is initialized

Here is the call graph for this function:



5.4.3.2 void initializeCamera (raspicam::RaspiCam_Cv & *Camera*)

Postcondition

camera image format set to single channel 8 bit
gain set to maximum value

Parameters

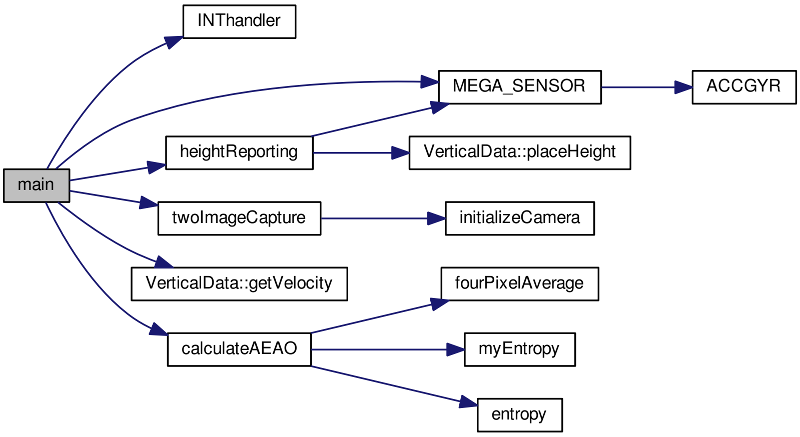
`in,out`	*Camera*	the camera interface

5.4.3.3 int main (int *argc,* char ∗ *argv[]*)

Declares input/output variables for sensor data Declares variables for calculated values Spawns threads using heightReporting and twoImageCapture Calls velocityCalculate with sensor data and calculated value variables $<$ equals 1 if lidar initialized

wait until 1s has passed to get velocity every second

Here is the call graph for this function:



5.4.3.4 void twoImageCapture (Mat & *image_1*, Mat & *image_2*, bool & *exit_flag*, bool & *ready_flag*, bool & *wait_flag*)

capture two consecutive images at 90 frames per second, loops until exit_flag is true

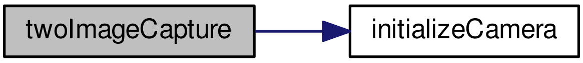
Precondition

a thread has been created and id assigned

Parameters

in,out	*image_1*	the first image to be captured
in,out	*image_2*	the second image to be captured
in	*exit_flag*	flag to alert thread to exit
in,out	*ready_flag*	to alert two images have been captured
in,out	*wait_flag*	flag to alert images are being used

Here is the call graph for this function:



Index