Instructions and guide to the AR Drone 2.0 GPS addition

Vsevolod Igorevitch Kiriouchine

0665387
02-11-2017

With thanks to:
dr. ir. R. Toth
dr. ir. I. Barosan
ir. S. Marx
ir. P.B. Cox
Daren Lee
Tod Kurt

Eindhoven University of Technology
Faculty of Mechanical Engineering
In cooperation with the Faculty of Electrical Engineering

Contents

0.1 Introduction

This document describes the GPS C driver and the use of the GPS incorporation of [1] into the Matlab ARDrone 2.0 toolbox of [2] as well as a standalone version. In this document all needed software is either linked if provided by another company, or is pasted inside this document in the Code section 0.4 and should be in the accompanying archived folder `Files_GPS_ardrone.zip`.

0.2 Standalone code

0.2.1 Software for standalone code

The standalone version does not require the use of Matlab. It uses the following software:

- A gcc compiler: here the Code Sourcery Gnueabi compiler is used from `https://sourcery.mentor.com/sgpp/lite/arm/portal/package8738/public/arm-none-linux-gnueabi/arm-2011.03-41-arm-none-linux-gnueabi.exe` , make sure to execute in Windows 7 compatibility mode when installing
- PuTTY: a free telnet program from `http://www.putty.org/`
- Ublox ucenter: the free GPS configuration (and preview) program
- Possibly WinFTP or another FTP program: `https://winscp.net/eng/download.php`

The incorporation of an external GPS module is based on the use of the USB port on top of the Parrot AR Drone. The configuration of the quad-copter and the GPS is shown in figure 1. How the TLL 'ttl-232r-3v3' module is connected to the GPS module (note the wire colours) is shown in figure 2.

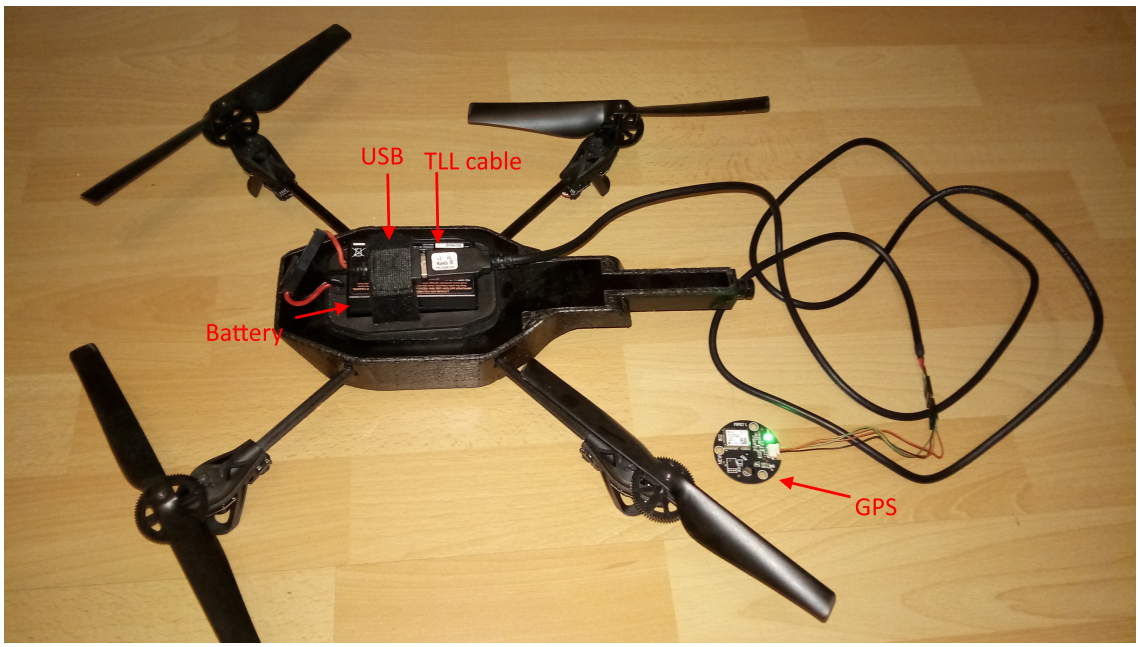

Figure 1: The configuration of the quad-copter and GPS module.

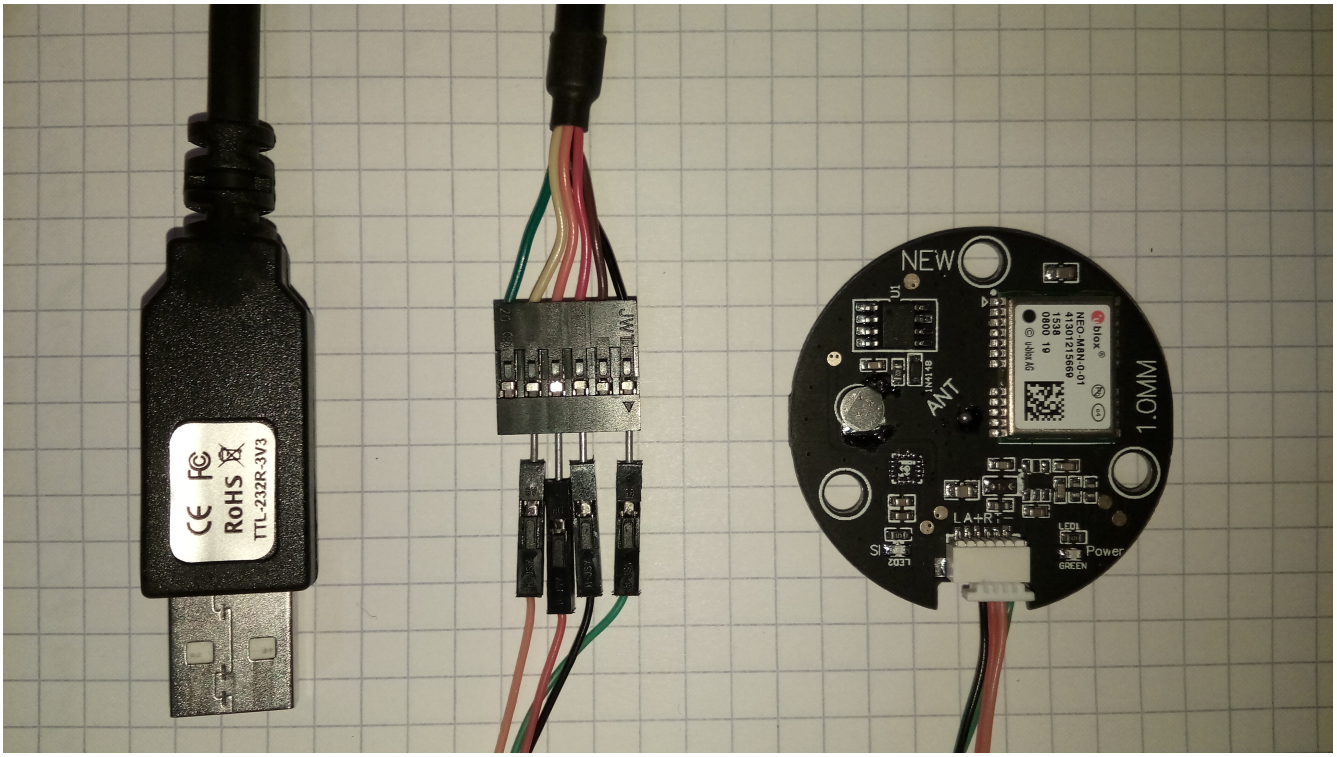

Figure 2: The used NEO-M8N-0-01 GPS module with the FTDI TTL 'ttl-232r-3v3' cable with the colour-coded transferring shown.

Before use, the module should be tested with Ublox ucenter software (available for free on Windows) using the usb to connect with your computer or laptop to make sure the Module is working well.

0.2.2 Compiling the standalone C driver

The first step in using the stand alone version is in installing a gcc compiler, such as: the Code Sourcery Gnueabi compiler is used from `https://sourcery.mentor.com/sgpp/lite/arm/portal/package8738/public/arm-none-linux-gnueabi/arm-2011.03-41-arm-none-linux-gnueabi.exe` , make sure to execute in Windows 7 compatibility mode when installing.

The compiling happens through your command prompt. For this search in windows for 'cmd' and open it. First make sure that the corresponding code is in the right folder. In this case there are 3 files needed: the C GPS driver ,named `gps_openC_backup_WorkingLongiLatiAutom.c` , shown in section 0.4.1. The `arduino-serial-lib.c` from `https://todbot.com/blog/2013/04/29/arduino-serial-updated/` [3]. And it's header `arduino-serial-lib.h` (in section 0.4.2)

In command prompt, navigate to the folder where you have placed the 3 aforementioned files and input the command:

`arm-none-linux-gnueabi-gcc-Wallgps_openC_backup_WorkingLongiLatiAutom.cc:/Users/vsevolod/arduino-serial-lib.c-ogpsOpenC.elf`

This uses the 'arm-none-linux-gnueabi-gcc' compiler with the main code `gps_openC_backup_WorkingLongiLatiAutom.c` and the extra code `arduino-serial-lib.c` to create the Linux executable file `gpsOpenC.elf`.

In this case the path is `c:/Users/vsevolod/` , to compile substitute your own path to the files.

Now that the `gpsOpenC.elf` is made, the PuTTY can be installed and opened. Upon opening PuTTY, make a wifi connection to the quad-copter and configure Putty. Create a new session with the IP 192.168.1.1 and the port 23 with the type Telnet, as seen in figure 3, then you can save the configuration for future use. Press 'Open' to connect to the quad-copter.

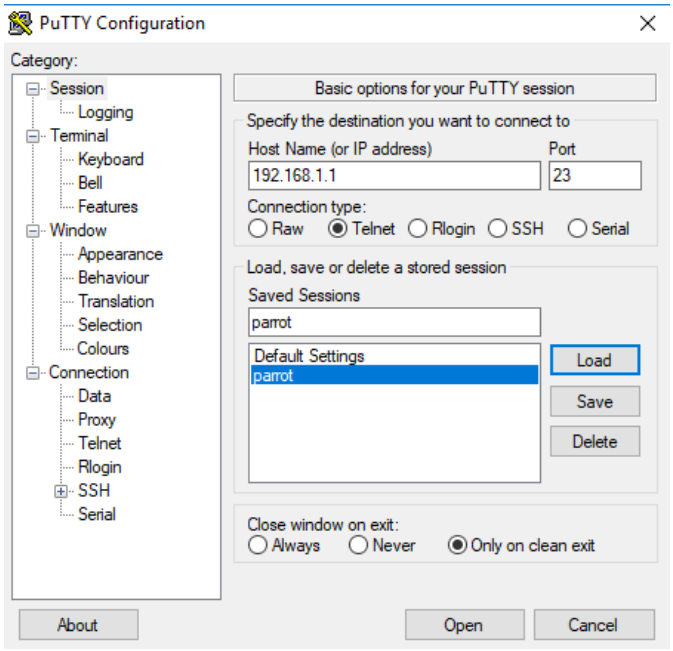

Figure 3: The PuTTY starting window

A new black window should appear. This window uses Linux commands to navigate, such as; 'ls' to view a list of folders, 'cd dev' to move to the dev folder (or any other), 'cd ..' to go up the folder path, and others.

For now type 'ls' and Enter to view the folders, then type 'cd dev' to go to the dev folder and type 'ls' again, as in figure 4. If the GPS module is connected properly the pink name ttyUSB0 or ttyUSB1 should be seen, this is the virtual directory path that the driver uses, it should be ttyUSB0, if it is not, reconnect the GPS usb module from the quad-copter until the ttyUSB0 is present here (usually this happens automatically and this step can be skipped).

Now move to the update folder, by using 'cd ..' and 'cd update'. When 'ls' is used, the only file present is version.txt or none. Here the file `gpsOpenC.elf` should be put onto the drone. This can be done using the WinSCP program or using Matlab's build in FTP as seen in figure 5. Note that the code is also given in the last section for using Matlab's FTP (Note that the transferred file `gpsOpenC.elf` originated from the same folder as is the working folder in Matlab), again however Matlab is not necessarily required for this step and another (free) FTP can be used, hence the name standalone.

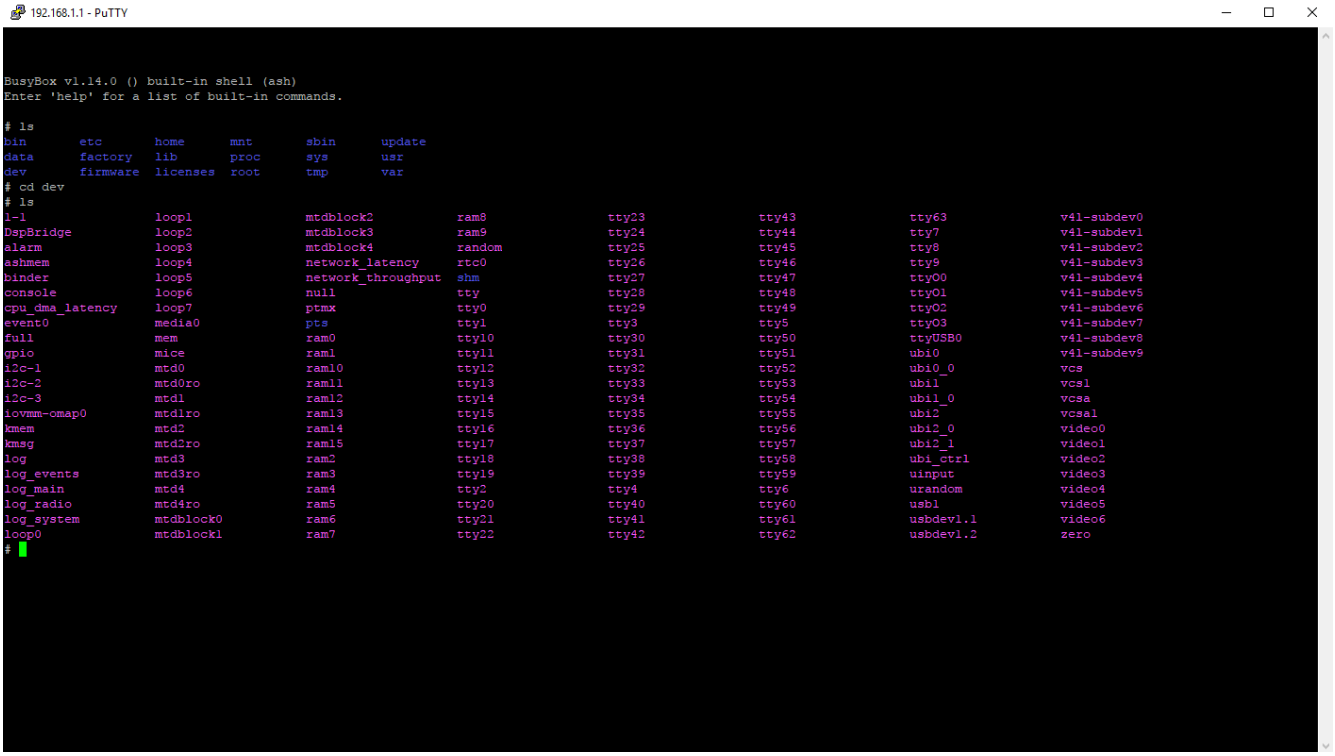

Figure 4: The PuTTY opened window

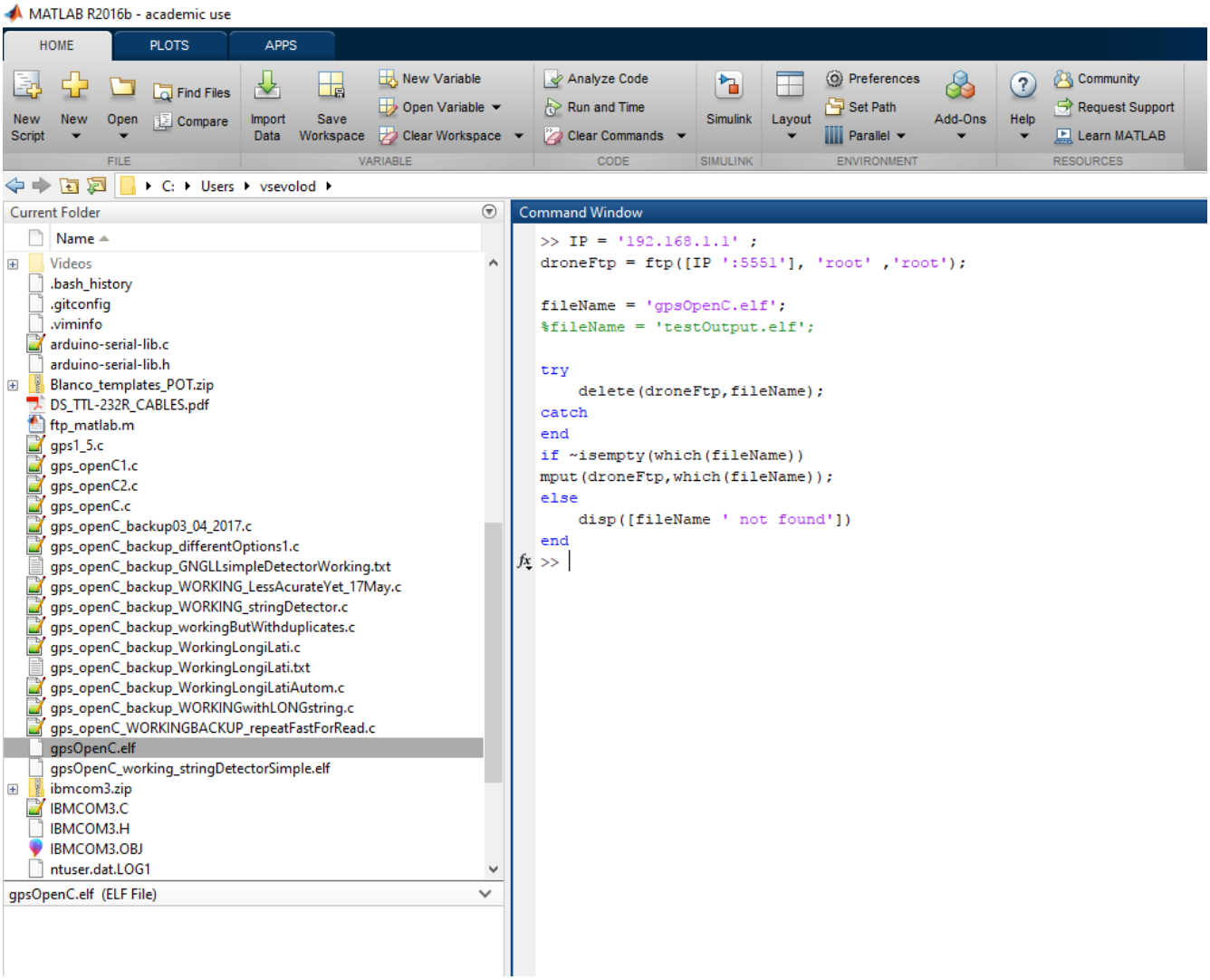

Figure 5: FTP via Matlab

Once the file `gpsOpenC.elf` is transferred into the update folder it should be made executable using the command:

chmod 777 gpsOpenC.elf

When the command 'ls' is used again, it should appear green in putty.

Figure 6: Green file

Next the file is run, using:

`./gpsOpenC.elf`

If used correctly the GPS gives the location:

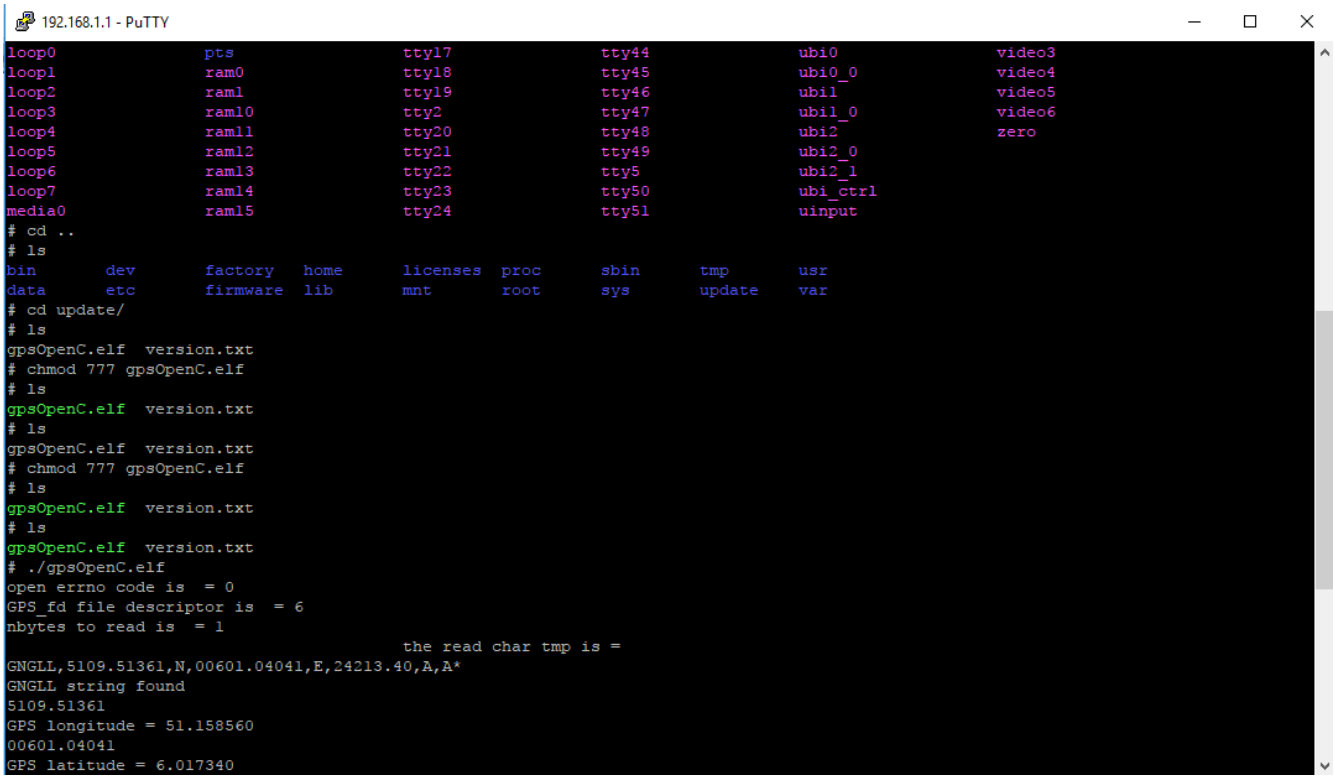

Figure 7: The GPS gives location

It is the longitude and latitude in degrees, named 'GPS longitude' and 'GPS latitude', and the degree and minute decimated output before it.

It is also possible that the program times out, in this case rerun it with `./gpsOpenC.elf` (for fast use press the up arrow to see the previous command), this stand alone version namely searches for the GNGLL string for a set amount of iterations, the Matlab version does this continuously and provides an updating stream of data.

It is also useful to use the Ublox ucenter software (available for free on Windows) for configuring the GPS module via USB using your computer. The module can be configured with Baud rates, for example the standalone version uses 9600.

The Matlab version uses 4800. The reason why Matlab uses this rate is update frequency. For higher accuracy the gps module can be configured in Ublox ucenter to update at 5Hz and only send the GNGLL string, this means that less inforamtion needs to be send and thus for robustness the baud rate is lowered to 4800. Thus when using the Matlab version, either configure the module in ucenter to the 4800 baud rate (in the PRT(Ports) configuration menu) or adapt the software.

Below is a screen with the menu in Ublox ucenter, for more information read the user guide. The configuration uses two screens. The first is where the PRT (port) menu is used to adapt the baud rate , once done click the send button below. The next is the NMEA menu with the to send strings, right click on all the strings and Disable Message for all the string except the GNGLL, and also click Send on the bottom of the screen to save the preferences. The GNSS(GNSS Config) on the left window can be used to use both GLONASS and GPS for higher accuracy. The CFG(Configuration) menu is used to select the devices for long storage with the Send command. And finally RATE(Rates) menu can be used to set the Measurement Period to 200ms to obtain a 5Hz sampling period for fast and accurate estimation.

The result should be a high frequency update signal, this is very usefull when used in Matlab with the filters described in [1].

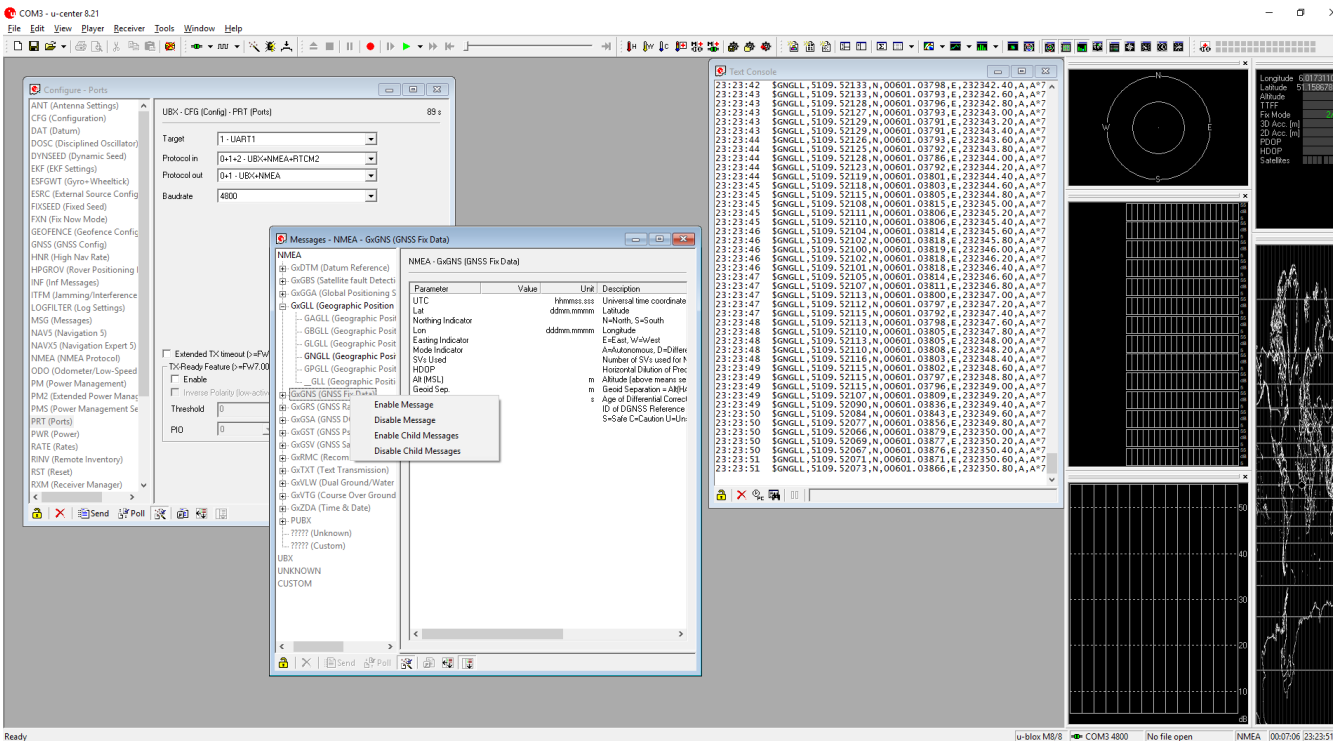

Figure 8: The GPS menu

0.3 Matlab version

For the Matlab the code does not need to be compiled manually and the Matlab version 2016a or 2016b is needed. Later and earlier versions will not work as the pathing of the Simulink files is different in other versions. This means that the entire toolbox will not be buildable [2], however the GPS driver portion will still work if the project path is adapted correctly, hence the high usefulness of the standalone GPS driver (as well as the usefulness of the Maltab version of the driver which could also be used in other applications).

For the Matlab version the toolbox needs to be installed, for the description view [2] . Once the toolbox is installed, files simply need to be copied and pasted in place.

Inside of `C:\Users\vsevolod\Desktop\presentation_a\AR_Drone_Target\blocks` (or your own path instead of `C:\Users\vsevolod\Desktop\presentation_a`) paste the following files:

- `arduino-serial-lib.c`
- `arduino-serial-lib.h`
- `GPS_Read.c` (see section 0.4.4)
- `GPS_Read.h`
- `Generate_AR_Drone_S_Functions.m` (Rewrite with the new version, seen in the code section 0.4.5

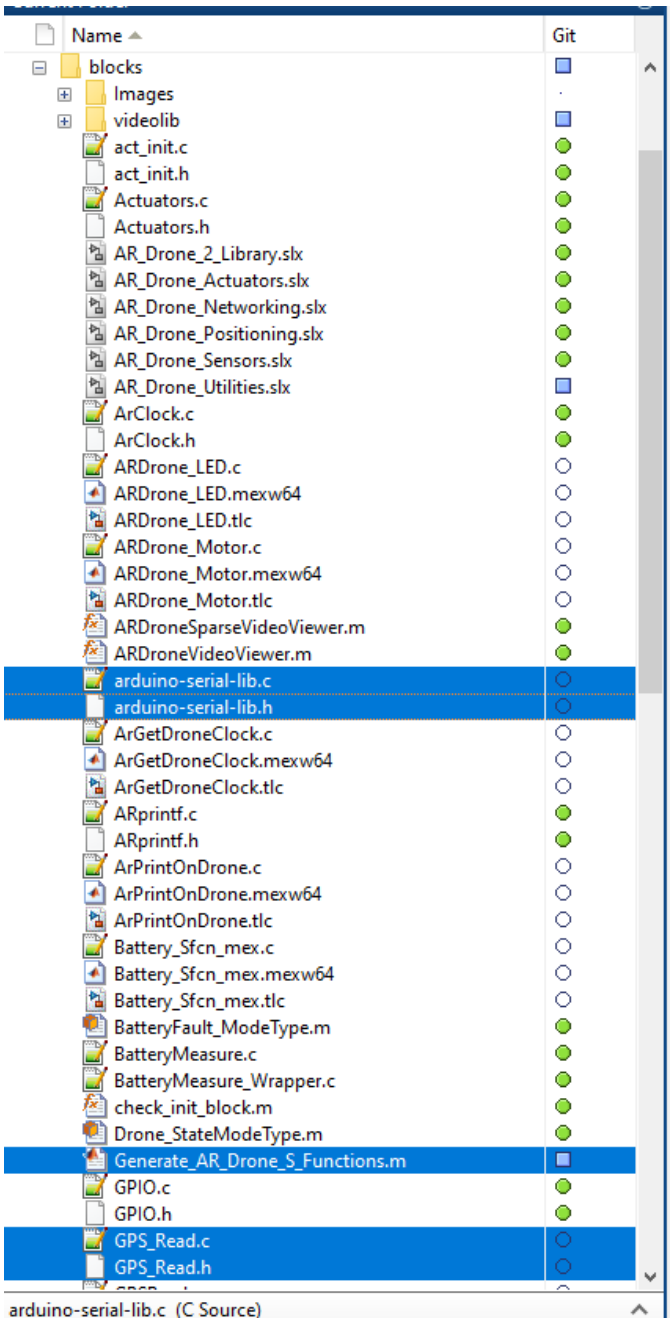

Figure 9: The files copied

Run the file `Generate_AR_Drone_S_Functions.m` inside Matlab to generate the blocks inside the library. The matlab m-file also creates the extra generated 'GPSRead.c', this however is not interesting and happened automatically, thus should be paid no attention to.

The generated block should be inside the Utilities library (some moving of the blocks might be necessary to see the gps block).

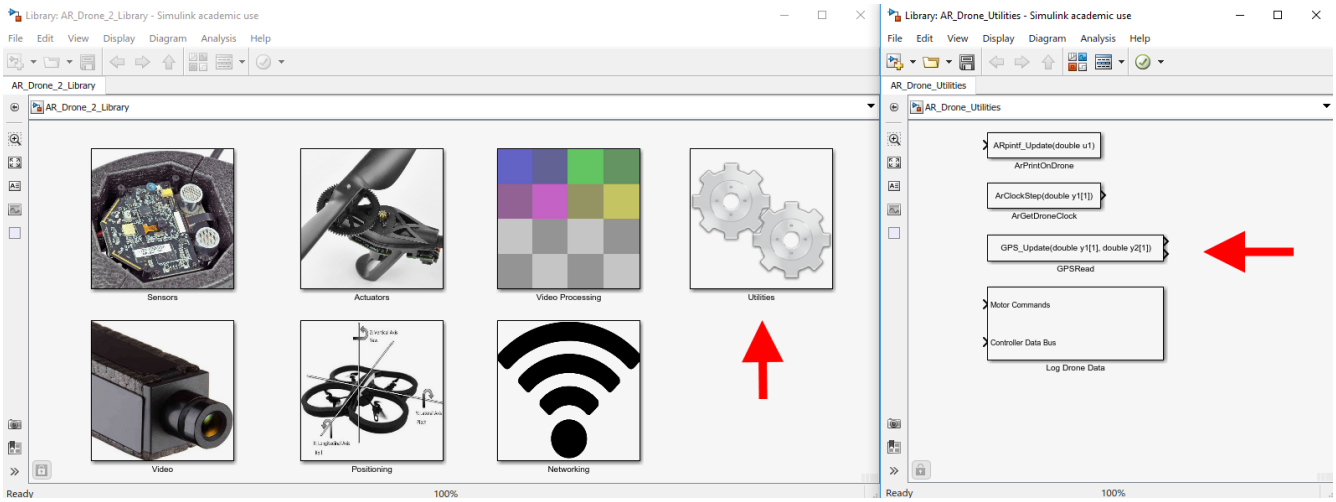

Figure 10: Utilities, you can click the lock button in the bottom left to move the blocks

Now the `GPS_Update` block can be used, where the first output y1 is the longitude (in degrees) and the y2 is the latitude (also in degrees), the data is not in the degree and minute decimated form, but is all in degrees.

The data comes in a spiked continuation, since the reading of the usb mostly outputs 0,0 when the GNGLL string is not found, and is only a relevant value when the GPS module is sending a string. To combat this a simple example is made in Simulink that uses memory blocks and triggers on a non zero value, thus always giving a relevant answer using the zero order hold method. This model can be in any folder, such as the root folder of the project, it is called `gps_test_query1.slx` and is sen in figure 11.

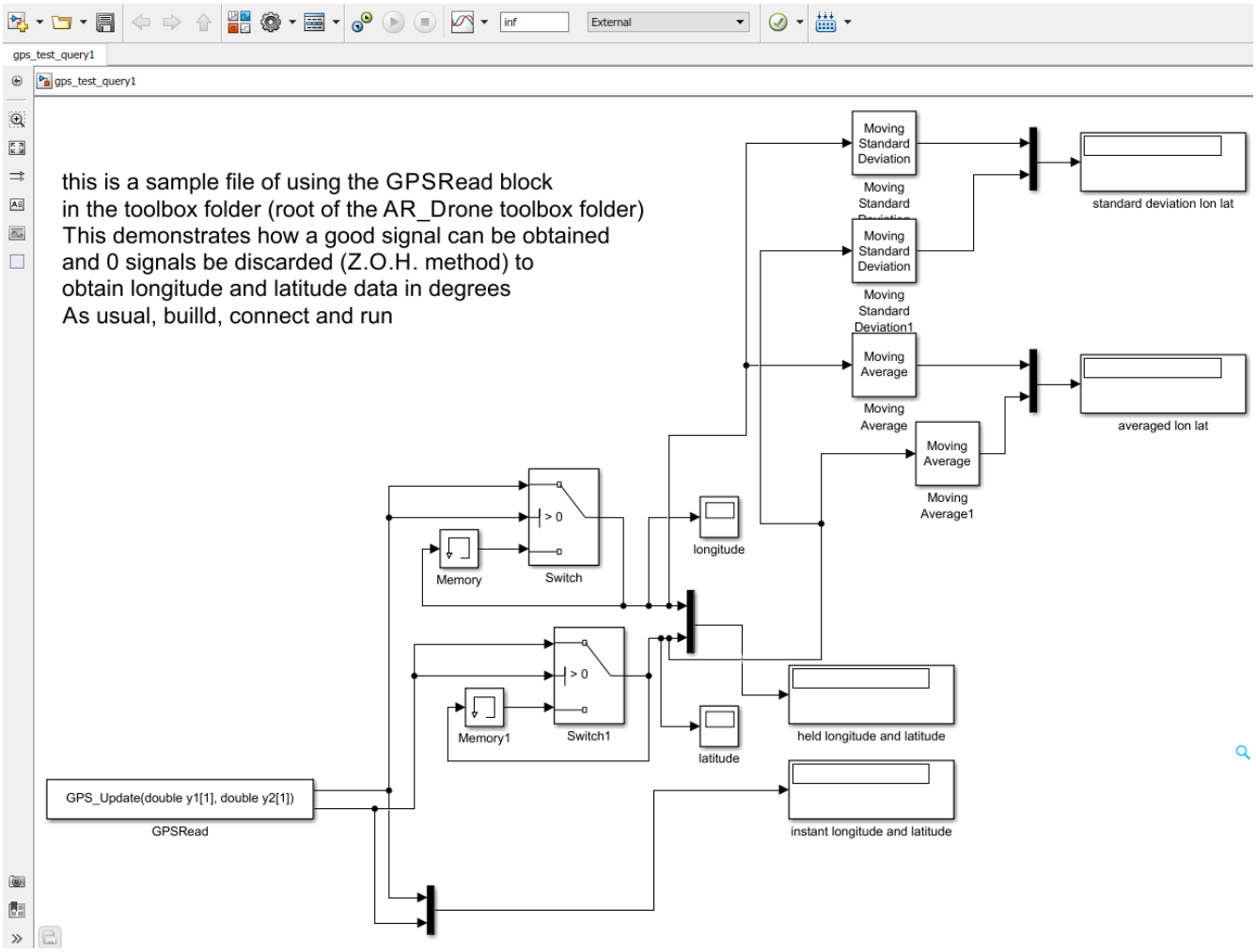

Figure 11: An example

The `GPS_Update` gives the values as seen in the blocks 'instant longitude and latitude' , when the project is Build, Connected and Ran (in External mode in this particular example) (like with other simulations in this toolbox), this block mainly outputs 0,0 and briefly the correct value.

To view the correct value continuously (with a 5Hz update frequency) a 'Switch' and 'Memory'block is used in tandem to skip the 0,0 values and only obtain the relevant values which are seen in the scopes. In htis simple example moving average filters are used for checking the accuracy, however it is recommended that other state observers are used for flight in the open [1].

Thank you for reading, for further information please contact `V.I.Kiriouchine*at*alumnus.tue.nl`, also please use proprietary and other code and software with the permission of the author(s) of their respective work.

0.4 Code

Feel free to adapt and use the driver code for your own purposes as well as reading the commented lines for more knowledge on the corresponding parts.

For the other codes, please ask the corresponding maker for permission.

0.4.1 standalone C GPS code

The standalone C GPS code, named `gps_openC_backup_WorkingLongiLatiAutom.c`:

```
//This is the STAND-ALSONE version of:
// GPS Driver by Vsevolod.I.Kiriouchine for AR_Drone toolbox by S.Marx and D.Lee
// at https://gitlab.com/SanneMarx/AR_Drone_2.0_Target
//This file should be in: YOUR-PROJECT-FOLDER-NAME\AR_Drone_Target\blocks
//Also include GPD_Read.h , arduino-serial-lib.c and .h in same folder
//arduino-serial-lib by T.Kurt from https://github.com/todbot/arduino-serial/

//This stand-alsone version can be put on the drone via FTP or SCP software such as WinSCP on the drone,
//Monitor the outputs via a Telnet program like PuTTY on ip 192.168.1.1 and port 23 or 5555

//Compile this program manually with a gcc compiler like:
  // arm-none-linux-gnueabi-gcc -Wall gps_openC.c c:/Users/vsevolod/arduino-serial-lib.c -o gpsOpenC.elf
//your linux working file is now gpsOpenC.elf

//to run this in command line via PuTTY go to the update folder:
  // cd update/
  // ls
  // chmod 777 gpsOpenC.elf
  // ./gpsOpenC.elf

// re-run this program for better results often

//For fast reading you can configure the GPS module in Ublox u-center to only send GNGLL messages at 5Hz for higher accuracy and estimation

#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <stdint.h>
#include <stdlib.h>
#include <getopt.h>

#include "arduino-serial-lib.h" //header of arduino-serial-lib library

int GPS_fd;
char testChar = 'P';
char stringRead[100];
int stringIndex;
int stopInt;
char tag[10];

int main()
{
  // arm-none-linux-gnueabi-gcc -Wall gps_openC.c c:/Users/vsevolod/arduino-serial-lib.c -o gpsOpenC.elf
  // chmod 777 gpsOpenC.elf
  // ./gpsOpenC.elf
  //int main(int argc, char **argv)

  GPS_fd = serialport_init("/dev/ttyUSB0", 9600); //using arduino-serial-lib library  . set to 4800 or 9600 BAUD dependant onyour GPS module setting


    printf("open errno code is  = %d \n", errno);
   printf("GPS_fd file descriptor is  = %d \n", GPS_fd);
```

```
56
57  char tmp;
58  // char tagEnd;
59  int tmp1;
60
61   int i =0;
62  size_t nbytes;
63  nbytes = sizeof(tmp);
64  printf("nbytes to read is  = %d \n", nbytes);
65
66  //do { //do executes at least one, before checking while, thus read is executed at least one
      regardless of datas initialization
67  //ssize_t datas = read (GPS_fd, &tmp, 1); // read from file descriptor GPS_fd and put in buffer
      tmp (with length 'sizeof tmp') datas returns size of read bytes?
68
69  ssize_t datas = read (GPS_fd, &tmp, 1);
70
71  printf("          the read char tmp is = %c \n", tmp);
72
73  stringIndex = 0;
74  stopInt = 0;
75  strcpy(stringRead, "");
76  int lenst = 0;
77  char endst;
78  int stopMark = 0;
79
80  //while( tmp1 != '$' && stopInt < 400) {
81
82    //printf("$ sign not found, found =  %c , %d \n", tmp, stopInt);
83
84    //ssize_t datas = read (GPS_fd, &tmp, 1);
85
86    //tmp1 = fgetc(GPS_fd);
87
88  do{
89
90    while(tmp != '$')
91      datas = read (GPS_fd, &tmp, 1);
92
93
94
95    do{ //Carrige return CR is 0x0d
96      datas = read (GPS_fd, tag, 5);
97
98      // if (datas != 0){
99        // tagEnd = tag[datas - 1];
100       // printf("$tagEnd =%c \n", tagEnd);
101     // }
102
103     tag[datas] = '\0';
104
105     if (datas != 0){
106         //printf("$tag found =%s= number=%d \n", tag, datas);
107
108       strcat(stringRead, tag);
109       strcpy(tag, "");
110
111       lenst =  strlen(stringRead);
112       endst = stringRead[lenst - 1];
113
114     }
115
116
117     stopInt++;
118
119
120 }while(endst != '*' && lenst < 100);
121
122 //printf("%s\n", stringRead);
123
124 double longiD;
```

```
125   double longiM;
126   double latiD;
127   double latiM;
128
129   char longistD[3];
130   char longistM[9];
131   double longiNum;
132
133   char latistD[4];
134   char latistM[9];
135   double latiNum;
136
137   char longist[11];
138   char latist[12];
139
140   if( stringRead[0] == 'G' && stringRead[1] == 'N' && stringRead[2] == 'G' && stringRead[3] == 'L'
        && stringRead[4] == 'L'){
141
142     printf("%s\n", stringRead);
143
144     printf("GNGLL string found \n");
145
146     /*extract longitude from GNGLL*/
147     strncpy(longist, stringRead+6, 10);
148
149     longist[11] = '\0';
150
151     printf("%s\n", longist);
152
153     /* get degrees of longitude*/
154     strncpy(longistD, longist, 2);
155
156     longistD[3] = '\0';
157
158     printf("%s\n", longistD);
159
160     longiD = atof(longistD);
161
162     printf("longi Degree %f \n", longiD);
163
164     /* get minutes of longitude*/
165     strncpy(longistM, longist+2, 8);
166
167     longistM[9] = '\0';
168
169     printf("%s\n", longistM);
170
171     longiM = atof(longistM);
172
173     printf("longi Minute %f \n", longiM);
174
175     /* Longitude total degrees*/
176     longiNum = longiD + longiM/60;
177     printf("GPS longitude = %f \n", longiNum);
178
179
180
181     /*extract latitude from GNGLL*/
182     strncpy(latist, stringRead+19, 11);
183
184     latist[12] = '\0';
185
186     printf("%s\n", latist);
187
188     /* get degrees of latitude*/
189     strncpy(latistD, latist, 3);
190
191     latistD[4] = '\0';
192
193     printf("%s\n", latistD);
194
```

```
    latiD = atof(latistD);

    printf("lati Degree %f \n", latiD);

    /* get minutes of latitude*/
    strncpy(latistM, latist+3, 8);

    latistM[9] = '\0';

    printf("%s\n", latistM);

    latiM = atof(latistM);

    printf("lati Minute %f \n", latiM);


    /* Latitude total degrees*/
    latiNum = latiD + latiM/60;
    printf("GPS latitude = %f \n", latiNum);



    stopMark = 1;
  }


  strcpy(stringRead, "");
  lenst = 0;
  usleep(10000);


  }while(stopMark == 0);

  stopInt = 0;

  return 0;
}
```

0.4.2 arduino-serial-lib

The 'arduino-serial-lib.c' from `https://todbot.com/blog/2013/04/29/arduino-serial-updated/`, courtesy of Tod Kurt [3]. (this is an external library, it handles the options and opening of the ttyUSB0 port for reading. It is not necessary to use this specific library as the options and UNIX port opening commands can be written into the code manually, however it offers an organised way of handling the opening).

Note the Baud rate, which is given in the GPS C driver in the previous subsection.

arduino-serial-lib.c :

```
//
// arduino-serial-lib -- simple library for reading/writing serial ports
//
// 2006-2013, Tod E. Kurt, http://todbot.com/blog/
//

#include "arduino-serial-lib.h"

#include <stdio.h>    // Standard input/output definitions
#include <unistd.h>   // UNIX standard function definitions
#include <fcntl.h>    // File control definitions
#include <errno.h>    // Error number definitions
#include <termios.h>  // POSIX terminal control definitions
#include <string.h>   // String function definitions
#include <sys/ioctl.h>

// uncomment this to debug reads
//#define SERIALPORTDEBUG

// takes the string name of the serial port (e.g. "/dev/tty.usbserial","COM1")
// and a baud rate (bps) and connects to that port at that speed and 8N1.
// opens the port in fully raw mode so you can send binary data.
```

```
// returns valid fd, or -1 on error
int serialport_init(const char* serialport, int baud)
{
    struct termios toptions;
    int fd;

    //fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
    //fd = open(serialport, O_RDWR | O_NONBLOCK );
    fd = open(serialport, O_RDWR | O_NOCTTY | O_NONBLOCK | O_NDELAY );

    if (fd == -1)  {
        perror("serialport_init: Unable to open port ");
        return -1;
    }

    //int iflags = TIOCM_DTR;
    //ioctl(fd, TIOCMBIS, &iflags);     // turn on DTR
    //ioctl(fd, TIOCMBIC, &iflags);    // turn off DTR

    if (tcgetattr(fd, &toptions) < 0) {
        perror("serialport_init: Couldn't get term attributes");
        return -1;
    }
    speed_t brate = baud; // let you override switch below if needed
    switch(baud) {
    case 4800:   brate=B4800;   break;
    case 9600:   brate=B9600;   break;
#ifdef B14400
    case 14400:  brate=B14400;  break;
#endif
    case 19200:  brate=B19200;  break;
#ifdef B28800
    case 28800:  brate=B28800;  break;
#endif
    case 38400:  brate=B38400;  break;
    case 57600:  brate=B57600;  break;
    case 115200: brate=B115200; break;
    }
    cfsetispeed(&toptions, brate);
    cfsetospeed(&toptions, brate);

    // 8N1
    toptions.c_cflag &= ~PARENB;
    toptions.c_cflag &= ~CSTOPB;
    toptions.c_cflag &= ~CSIZE;
    toptions.c_cflag |= CS8;
    // no flow control
    toptions.c_cflag &= ~CRTSCTS;

    //toptions.c_cflag &= ~HUPCL; // disable hang-up-on-close to avoid reset

    toptions.c_cflag |= CREAD | CLOCAL;  // turn on READ & ignore ctrl lines
    toptions.c_iflag &= ~(IXON | IXOFF | IXANY); // turn off s/w flow ctrl

    toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); // make raw
    toptions.c_oflag &= ~OPOST; // make raw

    // see: http://unixwiz.net/techtips/termios-vmin-vtime.html
    toptions.c_cc[VMIN]  = 0;
    toptions.c_cc[VTIME] = 0;
    //toptions.c_cc[VTIME] = 20;

    tcsetattr(fd, TCSANOW, &toptions);
    if( tcsetattr(fd, TCSAFLUSH, &toptions) < 0) {
        perror("init_serialport: Couldn't set term attributes");
        return -1;
    }

    return fd;
}
```

```
//
int serialport_close( int fd )
{
    return close( fd );
}

//
int serialport_writebyte( int fd, uint8_t b)
{
    int n = write(fd,&b,1);
    if( n!=1)
        return -1;
    return 0;
}

//
int serialport_write(int fd, const char* str)
{
    int len = strlen(str);
    int n = write(fd, str, len);
    if( n!=len ) {
        perror("serialport_write: couldn't write whole string\n");
        return -1;
    }
    return 0;
}

//
int serialport_read_until(int fd, char* buf, char until, int buf_max, int timeout)
{
    char b[1];  // read expects an array, so we give it a 1-byte array
    int i=0;
    do {
        int n = read(fd, b, 1);  // read a char at a time
        if( n==-1) return -1;    // couldn't read
        if( n==0 ) {
            usleep( 1 * 1000 );  // wait 1 msec try again
            timeout--;
            if( timeout==0 ) return -2;
            continue;
        }
#ifdef SERIALPORTDEBUG
        printf("serialport_read_until: i=%d, n=%d b='%c'\n",i,n,b[0]); // debug
#endif
        buf[i] = b[0];
        i++;
    } while( b[0] != until && i < buf_max && timeout>0 );

    buf[i] = 0;  // null terminate the string
    return 0;
}

//
int serialport_flush(int fd)
{
    sleep(2); //required to make flush work, for some reason
    return tcflush(fd, TCIOFLUSH);
}
```

arduino-serial-lib.h :

```
//
// arduino-serial-lib -- simple library for reading/writing serial ports
//
// 2006-2013, Tod E. Kurt, http://todbot.com/blog/
//


#ifndef __ARDUINO_SERIAL_LIB_H__
#define __ARDUINO_SERIAL_LIB_H__

#include <stdint.h>   // Standard types
```

```
12
13 int serialport_init(const char* serialport, int baud);
14 int serialport_close(int fd);
15 int serialport_writebyte( int fd, uint8_t b);
16 int serialport_write(int fd, const char* str);
17 int serialport_read_until(int fd, char* buf, char until, int buf_max,int timeout);
18 int serialport_flush(int fd);
19
20 #endif
```

0.4.3 FTP via Matlab

The m-file of using FTP through Matlab to transfer the standalone driver to the AR Drone (alternatively WinSCP (`https://winscp.net/eng/download.php`) or another FTP program can be used) , Note that the transferred file `gpsOpenC.elf` originated from the same folder as is the working folder in Matlab, named `ftp_matlab.m`:

```
1 % to compile a c file test.c into an elf file to rn in linux on drone:
2 % use cmd, go to folfer of test.c, and run the following command:
3 % arm-none-linux-gnueabi-gcc test.c -o testOutput.elf
4 % then testOutput.elf will show up
5
6 %to link libraries, such as , use -Wall cfile.c c:/pathtolibrary/libraryname.c between
7 %gcc and -o , for example :
8 % arm-none-linux-gnueabi-gcc -Wall gps_openC.c c:/Users/vsevolod/arduino-serial-lib.c -o gpsOpenC.
      elf
9
10 % run the following in matlab to make connection with quad-copter via ftp
11 % (also possible through winSCP with ftp mode), (you  have to be in the
12 % folder in which testOutput.elf is in when running the code below):
13
14 IP = '192.168.1.1' ;
15 droneFtp = ftp([IP ':5551'], 'root' ,'root');
16
17 fileName = 'gpsOpenC.elf';
18 %fileName = 'testOutput.elf';
19
20 try
21     delete(droneFtp,fileName);
22 catch
23 end
24 if ~isempty(which(fileName))
25 mput(droneFtp,which(fileName));
26 else
27     disp([fileName ' not found'])
28 end
29
30 % droneTcpip =
31
32 % in putty look up with 'ls' what is in there
33 % go to update folder with: cd update
34 % ls again to see testOutput.elf
35 % to make testOutput.elf possible to run, use the following command line in
36 % putty:
37 % chmod 777 testOutput.elf
38 % or
39 % chmod 777 gpsOpenC.elf
40 % then to run the elf file, give in putty:
41 % ./testOutput.elf
42 % to remove the file, for putting a new one on:
43 % rm testOutput.elf
```

0.4.4 GPS driver Matlab

The code for `GPS_Read.c`.

It uses a better searching algorithm for a more accurate string detection, as well as the 4800 baudrate .

```
1 // GPS Driver by Vsevolod.I.Kiriouchine for AR_Drone toolbox by S.Marx and D.Lee
2 // at https://gitlab.com/SanneMarx/AR_Drone_2.0_Target
```

```
//This file should be in: YOUR_PROJECT_FOLDER_NAME\AR_Drone_Target\blocks
//Also include GPD_Read.h , arduino-serial-lib.c and .h in same folder
//arduino-serial-lib by T.Kurt from https://github.com/todbot/arduino-serial/

#ifndef MATLAB_MEX_FILE

#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <stdint.h>
#include <stdlib.h>
#include <getopt.h>

#include "arduino-serial-lib.h" //header of arduino-serial-lib.c library

#endif //MATLAB_MEX_FILE

int GPS_fd;
double testNum = 5.123;
double testNum2 = 123.5;
char stringRead[100];
int stringIndex;
int stopInt;
char tag[10];

// Don't Forget tO Adapt Header File for uotputs of update function( )
// GPS_def.OutputFcnSpec = 'GPS_Update(int32 y1[1])'; // in Generate_AR_Drone_S_Functions.m to
    define outputs in GPS part
// GPS_def.OutputFcnSpec = 'GPS_Update(double y1[1])'; % use the type 'single' for simple float
//long lSize;
//this function file is called by Generate_AR_Drone_S_Functions.m that generates the simulink
    blocks
//Generated block is found in Utitlties Library, named GPSRead
//Block output y1 is lingitude[degrees], y2 is latitude[degrees]

//For fast reading you can configure the GPS module in Ublox u-center to only send GNGLL messages
    at 5Hz for higher accuracy and estimation

void GPS_Init(void)
{
#ifndef MATLAB_MEX_FILE

    GPS_fd = serialport_init("/dev/ttyUSB0", 4800); //using arduino-serial-lib library. set to
    4800 or 9600 BAUD dependant onyour GPS module setting

    //GPS_fd = open ("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NONBLOCK | O_NDELAY ); // "/dev/ttyUSB0
    ","/dev/usbdev1.3"

    // printf("Setting IMU Serial Port properties \n");
    //int flags = fcntl(GPS_fd, F_GETFL) ;

  //fcntl(GPS_fd, F_SETFL, flags | O_NONBLOCK); //read calls are non blocking
   // printf("is it open = %d\n", GPS_fd);
#endif //MATLAB_MEX_FILE
}

void GPS_Close(void)
{
#ifndef MATLAB_MEX_FILE
    //close(GPS_fd);
#endif //MATLAB_MEX_FILE
}

void GPS_Update(double *gps_open, double *gps_open2) {

#ifndef MATLAB_MEX_FILE

    char tmp;
```

```
69
70      ssize_t datas = read (GPS_fd, &tmp, 1);
71
72      stringIndex = 0;
73    stopInt = 0;
74    strcpy(stringRead, "");
75    int lenst = 0;
76    char endst;
77    int stopMark = 0;
78    int stopChar = 0;
79
80
81      //printf("the read char tmp is = %c \n", tmp);
82
83      //Once the start of the string is found with $
84      if(tmp == '$'){
85
86          do{ //Carrige return CR is 0x0d
87        datas = read (GPS_fd, tag, 5);
88
89        //usleep(1);
90
91        tag[datas] = '\0';
92
93        if (datas != 0){
94          strcat(stringRead, tag);
95          strcpy(tag, "");
96
97          lenst =  strlen(stringRead);
98          endst = stringRead[lenst - 1];
99        }
100
101
102       stopInt++;
103
104       if (stopInt > 100000 -2){
105         //printf("timeout stopInt \n");
106         stopMark = 1;
107       }
108
109         }while(endst != '*' && lenst < 34 && stopInt < 100000); //stopChar == 0
110
111     }
112     //printf("%s\n", stringRead);
113
114   double longiD;
115   double longiM;
116   double latiD;
117   double latiM;
118
119   char longistD[3];
120   char longistM[9];
121   double longiNum;
122
123   char latistD[4];
124   char latistM[9];
125   double latiNum;
126
127   char longist[11];
128   char latist[12];
129
130     //Once the string is read, chack if it is the GNGLL string
131     if( stringRead[0] == 'G' && stringRead[1] == 'N' && stringRead[2] == 'G' && stringRead[3] == '
      L' && stringRead[4] == 'L' && stringRead[5] == ',' && stringRead[10] == '.' && stringRead[16]
      == ',' && stringRead[18] == ',' && stringRead[24] == '.' && stringRead[30] == ','){
132
133     //printf("%s\n", stringRead);
134
135     //printf("GNGLL string found \n");
136
137     /*extract longitude from GNGLL*/
```

```
138     strncpy(longist, stringRead+6, 10);
139
140     longist[11] = '\0';
141
142     //printf("%s\n", longist);
143
144     /* get degrees of longitude*/
145     strncpy(longistD, longist, 2);
146
147     longistD[3] = '\0';
148
149     //printf("%s\n", longistD);
150
151     longiD = atof(longistD);
152
153     //printf("longi Degree %f \n", longiD);
154
155     /* get minutes of longitude*/
156     strncpy(longistM, longist+2, 8);
157
158     longistM[9] = '\0';
159
160     //printf("%s\n", longistM);
161
162     longiM = atof(longistM);
163
164     //printf("longi Minute %f \n", longiM);
165
166     /* Longitude total degrees*/
167     longiNum = longiD + longiM/60;
168     //printf("GPS longitude = %f \n", longiNum);
169
170
171
172     /*extract latitude from GNGLL*/
173     strncpy(latist, stringRead+19, 11);
174
175     latist[12] = '\0';
176
177     //printf("%s\n", latist);
178
179     /* get degrees of latitude*/
180     strncpy(latistD, latist, 3);
181
182     latistD[4] = '\0';
183
184     //printf("%s\n", latistD);
185
186     latiD = atof(latistD);
187
188     //printf("lati Degree %f \n", latiD);
189
190     /* get minutes of latitude*/
191     strncpy(latistM, latist+3, 8);
192
193     latistM[9] = '\0';
194
195     //printf("%s\n", latistM);
196
197     latiM = atof(latistM);
198
199     //printf("lati Minute %f \n", latiM);
200
201
202     /* Latitude total degrees*/
203     latiNum = latiD + latiM/60;
204     //printf("GPS latitude = %f \n", latiNum);
205
206
207
208     stopMark = 1;
```

```
  }

    //Send the longitude and latitude to simulink block for output
    *gps_open = longiNum;
    *gps_open2 = latiNum;

     //*gps_open = testNum;
     //*gps_open2 = testNum2;

     //GPS_fd = open ("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NONBLOCK | O_NDELAY );

    //uint16_t dirt = read (IMU_fd, tmp, sizeof tmp);
    //int GPS_sig = read (GPS_fd);

  // obtain file size:
  //fseek (GPS_fd , 0 , SEEK_END);
  //lSize = ftell (GPS_fd);
  //rewind (GPS_fd);

//    char tmp[1]; //maybe uint8_t[1] instead of char[1] or non-array ?
//    int datas = read (GPS_fd, &tmp, sizeof tmp); // read from file descriptor GPS_fd and put in
    buffer tmp? datas returns size of read bytes?

   // char tmp2 = *tmp[ 1 ]; //maybe uint8_t
   // int readInt = (int)tmp2; // cast the read byte in the tmp buffer to int to display?

//     printf("read funct return is  = %c \n", tmp); //printf("read funct return is  = %d \n",
    GPS_fd);

    //*gps_open = readInt; //*gps_open = GPS_fd;

     // printf("errno code is  = %d \n", errno);
    // *gps_open = errno;
  //printf("char2print = %f\n", char2print);
#endif //MATLAB_MEX_FILE
}
```

The code for `GPS_Read.h`.

```
#ifndef _GPS_READ_H
#define _GPS_READ_H

#ifndef MATLAB_MEX_FILE
#endif //MATLAB_MEX_FILE

void GPS_Init(void);
void GPS_Close(void);
void GPS_Update(double *gps_open, double *gps_open2);
#endif
```

0.4.5 GPS driver Matlab

The code for `Generate_AR_Drone_S_Functions.m`.

The new part is marked with %% GPS

```
%% Make sure this is done in the correct folder

disp('=============================================================================')
disp('                Generating AR Drone Library Blocks                           ')
disp('=============================================================================')

originalDir = pwd;
targetDir = which('Generate_AR_Drone_S_Functions');
cd(targetDir(1:end-32));

%% Actuator Initialization

act_init_def = legacy_code('initialize');
act_init_def.SourceFiles = {'act_init.c'};
act_init_def.HeaderFiles = {'act_init.h'};
act_init_def.IncPaths = {''};
```

```
act_init_def.SFunctionName = 'Init_Actuator';
act_init_def.Options.language = 'C';
act_init_def.StartFcnSpec = 'Actuator_Initialization(void)';
act_init_def.TerminateFcnSpec = 'Actuator_Stop(void)';

if exist('Init_Actuator.c ','file')~=0
    delete('Init_Actuator.c ')
end

% Generate s function
legacy_code('sfcn_cmex_generate', act_init_def);
legacy_code('compile', act_init_def);
act_init_def.SourceFiles = {'act_init.c','Actuators.c','GPIO.c'};
% Add dependancy for target code generation
act_init_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', act_init_def);

%% create lib for utilities

lib_name = 'AR_Drone_Utilities';
frontCamBlockName = 'AR_Drone_Front_Camera';
botCamBlockName = 'AR_Drone_Bottom_Camera';
if exist(lib_name, 'file')
  load_system(lib_name);
  set_param(lib_name, 'Lock', 'off')
  try
  delete_block([lib_name, '/Print On Drone']);
  delete_block([lib_name, '/Get Drone Clock']);
  catch
  end
else
  new_system(lib_name, 'Library');
  load_system(lib_name);
end

%% printf

print_def = legacy_code('initialize');
print_def.SourceFiles = {'ARprintf.c'};
print_def.HeaderFiles = {'ARprintf.h'};
print_def.IncPaths = {''};
print_def.SFunctionName = 'ArPrintOnDrone';
print_def.Options.language = 'C';
print_def.StartFcnSpec = 'ARpintf_Init(void)';
print_def.OutputFcnSpec = 'ARpintf_Update(double u1)';
print_def.TerminateFcnSpec = 'ARpintf_Close(void)';

if exist('ARprint.c ','file')~=0
    delete('ARprint.c ')
end

% Generate s function
legacy_code('sfcn_cmex_generate', print_def);
legacy_code('compile', print_def);
lib_def = print_def;

% Add dependancy for target code generation
print_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', print_def);

%% Clock

clock_def = legacy_code('initialize');
clock_def.SourceFiles = {'ArClock.c'};
clock_def.HeaderFiles = {'ArClock.h'};
clock_def.IncPaths = {''};
clock_def.SFunctionName = 'ArGetDroneClock';
clock_def.Options.language = 'C';
clock_def.StartFcnSpec = 'ArClockInit(void)';
clock_def.OutputFcnSpec = 'ArClockStep(double y1[1])';
clock_def.TerminateFcnSpec = 'ArClockClose(void)';
```

```
clock_def.SampleTime    = 'parameterized';

if exist('ARclock_Sfcn.c ','file')~=0
    delete('ARclock_Sfcn.c ')
end

% Generate s function
legacy_code('sfcn_cmex_generate', clock_def);
legacy_code('compile', clock_def);
lib_def = [lib_def clock_def];

% Add dependancy for target code generation
clock_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', clock_def);

%% GPS
GPS_def = legacy_code('initialize');
GPS_def.SourceFiles = {'GPS_Read.c'};
GPS_def.HeaderFiles = {'GPS_Read.h'};
GPS_def.IncPaths = {''};
GPS_def.SFunctionName = 'GPSRead';
GPS_def.Options.language = 'C';
GPS_def.StartFcnSpec = 'GPS_Init(void)';
GPS_def.OutputFcnSpec = 'GPS_Update(double y1[1], double y2[1])';
GPS_def.TerminateFcnSpec = 'GPS_Close(void)';

if exist('GPSRead.c ','file')~=0
    delete('GPSRead.c ')
end

% Generate s function
legacy_code('sfcn_cmex_generate', GPS_def);
legacy_code('compile', GPS_def);
lib_def = [lib_def GPS_def];

% Add dependancy for target code generation
GPS_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', GPS_def);

GPS_def.SourceFiles  = {'arduino-serial-lib.c','GPS_Read.c'};
GPS_def.HeaderFiles  = {'arduino-serial-lib.h'};
GPS_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', GPS_def);

%% create blocks lib

legacy_code('slblock_generate',lib_def,lib_name);
set_param(lib_name, 'EnableLBRepository','on');
set_param(lib_name, 'Lock', 'on');
save_system(lib_name);
bdclose;

%% IMU Block

% load the IMU Packet bus
load('IMU_Packets_Bus.mat')

IMU_Block_def = legacy_code('initialize');
IMU_Block_def.SFunctionName = 'IMU_Sfcn_mex';
IMU_Block_def.InitializeConditionsFcnSpec  = 'void MDL_IMU_start()';
IMU_Block_def.TerminateFcnSpec = 'void MDL_IMU_term()';
IMU_Block_def.OutputFcnSpec                = 'void MDL_IMU_step(IMU_Packets y1[1], int32 y2[1],
    int32 y3[1], int32 u1)';
IMU_Block_def.SourceFiles                  = {'IMU_Navdata_Wrapper.c'};
IMU_Block_def.IncPaths                     = {''};
IMU_Block_def.Options.useTlcWithAccel      = false;
IMU_Block_def.Options.language =  'C';
IMU_Block_def.SampleTime    = 'parameterized';

if exist('IMU_Sfcn_mex.c ','file')~=0
    delete('IMU_Sfcn_mex.c ')
```

```
158 end
159
160 % Generate S-function source file
161 legacy_code('sfcn_cmex_generate', IMU_Block_def);
162
163 % Generate for simulation only - the S-function does nothing in normal mode
164 legacy_code('generate_for_sim', IMU_Block_def);
165
166 % When generating TLC, we want to include additional source files which are
167 % device driver / HW specific
168 IMU_Block_def.SourceFiles  = {'IMU_Navdata.c','IMU_Navdata_wrapper.c'};
169 IMU_Block_def.HeaderFiles  = {'IMU_Navdata.h'};
170 IMU_Block_def.SrcPaths = {'.'};
171 legacy_code('sfcn_tlc_generate', IMU_Block_def);
172
173 %% LED Block
174
175 LED_Block_def = legacy_code('initialize');
176 LED_Block_def.SourceFiles = {'led.c'};
177 LED_Block_def.HeaderFiles = {'led.h'};
178 LED_Block_def.IncPaths = {''};
179 LED_Block_def.SFunctionName = 'ARDrone_LED';
180 LED_Block_def.Options.language = 'C';
181 LED_Block_def.OutputFcnSpec = 'LED_set(uint8 u1[4])';
182
183 if exist('ARDrone_LED.c ','file')~=0
184     delete('ARDrone_LED.c ')
185 end
186
187 % Generate s function
188 legacy_code('sfcn_cmex_generate', LED_Block_def);
189 legacy_code('compile', LED_Block_def);
190
191 % Add dependancy for target code generation
192 LED_Block_def.SourceFiles = {'led.c','Actuators.c','GPIO.c'};
193 LED_Block_def.SrcPaths = {'.'};
194 legacy_code('sfcn_tlc_generate', LED_Block_def);
195
196 %% Motor Block
197
198 Motor_def = legacy_code('initialize');
199 Motor_def.SourceFiles = {'motor.c'};
200 Motor_def.HeaderFiles = {'motor.h'};
201 Motor_def.IncPaths = {''};
202 Motor_def.SFunctionName = 'ARDrone_Motor';
203 Motor_def.Options.language = 'C';
204 Motor_def.OutputFcnSpec = 'uint16 y1 = Motor_Set(uint8 u1[5])';
205
206 if exist('ARDrone_Motor.c ','file')~=0
207     delete('ARDrone_Motor.c ')
208 end
209
210 % Generate s function
211 legacy_code('sfcn_cmex_generate', Motor_def);
212 legacy_code('compile', Motor_def);
213
214 % Add dependancy for target code generation
215 Motor_def.SourceFiles = {'motor.c','Actuators.c','GPIO.c'};
216 legacy_code('sfcn_tlc_generate', Motor_def);
217
218 %% Version Check Block
219 Version_Check_def = legacy_code('initialize');
220 Version_Check_def.SourceFiles = {'versionCheck.c'};
221 Version_Check_def.HeaderFiles = {'versionCheck.h'};
222 Version_Check_def.IncPaths = {''};
223 Version_Check_def.SFunctionName = 'Version_Check';
224 Version_Check_def.Options.language = 'C';
225 Version_Check_def.StartFcnSpec = 'void versionCheckInit(void)';
226 Version_Check_def.OutputFcnSpec = 'void versionCheckStep(uint8 y1[1])';
227 Version_Check_def.TerminateFcnSpec = 'void versionCheckClose(void)';
228
```

```
if exist('Version_Check.c ','file')~=0
    delete('Version_Check.c ')
end

% Generate s function
legacy_code('sfcn_cmex_generate', Version_Check_def);
legacy_code('compile', Version_Check_def);
Version_Check_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', Version_Check_def);

%% Battery block
Battery_def = legacy_code('initialize');
Battery_def.SFunctionName = 'Battery_Sfcn_mex';
Battery_def.InitializeConditionsFcnSpec  = 'void BatteryMeasure_start()';
Battery_def.TerminateFcnSpec = 'void BatteryMeasure_term()';
Battery_def.OutputFcnSpec                = 'void BatteryMeasure_step(single y1[1])';
Battery_def.SourceFiles                  = {'BatteryMeasure_Wrapper.c'};
Battery_def.IncPaths                     = {''};
Battery_def.Options.useTlcWithAccel      = false;
Battery_def.Options.language =  'C';
Battery_def.SampleTime    = 'parameterized';

if exist('Battery_Sfcn_mex.c ','file')~=0
    delete('Battery_Sfcn_mex.c ')
end

% Generate S-function source file
legacy_code('sfcn_cmex_generate', Battery_def);

% Generate for simulation only - the S-function does nothing in normal mode
legacy_code('generate_for_sim', Battery_def);

% When generating TLC, we want to include additional source files which are
% device driver / HW specific
Battery_def.HeaderFiles                  = {'i2c-dev.h'};
Battery_def.SourceFiles  = {'BatteryMeasure.c','BatteryMeasure_Wrapper.c'};
Battery_def.SrcPaths = {'.'};
legacy_code('sfcn_tlc_generate', Battery_def);

%% Generate the rtwmakecfg file for all s functions

legacy_code('rtwmakecfg_generate', [print_def; act_init_def;IMU_Block_def;LED_Block_def;Motor_def;
    Version_Check_def;Battery_def; clock_def]);

%% clear the loaded bus object

clear('IMU_Packets');

%% return to original path
cd(originalDir);
```

Bibliography

[1] Vsevolod I. Kiriouchine "*Sensing and control design for high speed autonomous agile manoeuvring with quadrotors.*" Eindhoven, Eindhoven University of Technology, November 2017

[2] ArDrone2Target Toolbox by; Daren Lee, Sanne Marx. `https://github.com/darenlee/SimulinkARDroneTarget` , `https://gitlab.com/ARDrone/ArDrone2Target`, 2014, 2017.

[3] The arduino-serial-lib, courtesy of; Tod Kurt. `https://github.com/todbot/arduino-serial/` , `https://todbot.com/blog/2013/04/29/arduino-serial-updated/`, 2013, 2006.