Sendata Strcture: Instructions for Use and Intergration into AR.Drone API

Jesse Weeks
SPARTN Lab - Department of Electrical and Computer Engineering
Tufts University
jesse.weeks@tufts.edu

CONTENTS

I. Introduction

The Sendata structure was created to allow the use of third-party hardware with the Parrot AR.Drone (ARD) Application Programming Interface (API). This was done by replicating the Navdata communications protocol developed by Parrot. This document is meant to serve as instructions on how to install and utilize the Sendata protocol in the ARD's Software Development Kit (SDK). Sendata was developed for a Linux operating system and has been tested on Ubuntu 10.04-LTSx64.

A. Key Terms

The following are key terms or abbreviations that will be used throughout this document:

ARD
: The Parrot AR.Drone

API
: The Application Programming Interface supplied by Parrot

SDK
: The Software Development Kit supplied by Parrot that is used to build the API

Arduino
: An Arduino Mega2560, with a Wi-Fi transceiver; the server for which the Sendata was developed

Transceiver
: A Wi-Fly RN-XV Wi-Fi transceiver that was used as a wireless interface for the Arduino

Terminal
: The computer compiling and running the SDK and API

Client
: The device receiving the data streams (in most cases this is the Terminal)

Server
: The device creating and sending the data streams (in most cases this is the ARD or the Arduino)

SDKDIR
: The extracted base folder of the SDK (ie Ardrone_SDK_Version_1_8_20110726)

II. Hardware Specification & Configuration

The server side of Sendata was designed to work on an Arduino Mega2560 using a Wi-Fly RN-XV Wi-Fi transceiver. The power and ground pins of the RN-XV were connected to the 3.3V and GND pins of the Arduino respectively. The DOUT pin was connected to the Serial3 RX pin on the Arduino, and the DIN pin was connected to the Serial3 TX pin. Additionally the RST pin of the RN-XV was connected to I/O pin 40 on the Arduino. If a different layout is used, the necessary pin mappings can be changed by adjusting the `#define` statements in *arduino_server.cpp*, a copy of which can be found in Section B of the Appendix.

A. RN-XV Configuration

The RN-XV must be configured to connect to the client and to properly relay the data from the server. The configuration settings are as follows:

ssid=ardx
: Set to the name of the Wi-Fi network generated by the ARD

protocol=1
: Set IP protocol to UDP

dhcp=0
: Disable DHCP address assignment

address=192.168.1.3
: IP address of chip; must match `WIFI_ARDUINO_IP` in *sendata_config.h*.

host=192.168.1.2
: IP address of the client

remote=5574
: Port on client for receipt of UDP packet stream; must match `SENDATA_PORT` in *sendata_config.h*

local=5574
: Port UDP packet stream will be sent on; must match the remote port

mtu=1500
: MTU of Wi-Fi connection; should match that of the client

comm time=0
: Disable send packet at fixed time intervals

comm size=1420
: Set data buffer to maximum

comm match=35
: Send packet when ASCII symbol 35 (#) is received; should match `MATCHCHAR` in *arduino_server.cpp*

The Arduino sketch *serialfor* was written to allow easy serial communication with the RN-XV so that it may be configured using the Arduino Serial Monitor. A copy of the *serialfor* sketch and the necessary command strings to configure the RN-XV can be found in Sections A and C of the Appendix.

III. Installation of Sendata

The Sendata protocol consists of four distinct sections:

1) The Sendata core library
2) The Control library
3) The Server library
4) The Client library

The files that comprise each of the sections and their installation directories on the terminal are as follows: (some of these directories must be created by the user)

A. Core

Files:

- `ardrone_general_sendata.c`
- `ardrone_general_sendata.h`
- `ardrone_sendata_client.c`
- `ardrone_sendata_client.h`
- `sendata.c`
- `sendata.h`
- `sendata_common.h`
- `sendata_config.h`
- `sendata_keys.h`
- `sendata_server.c`
- `sendata_server.h`

Installation Directory: `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Sendata`

B. Control

Files:

- `ardrone_sendata_control.c`
- `ardrone_sendata_control.h`

Installation Directory: `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Control`

C. Server

Files:

- `arduino_server.cpp`
- `arduino_server.h`

Installation Directory: `SDKDIR/Arduino/source`

D. Client

Files:

- `sendata.c`
- `sendata.h`

Installation Directory: `SDKDIR/Examples/Linux/sdk_demo/Sources/Sendata`

E. Modifications to SDK Files

Once the Sendata libraries are installed they must be integrated into the API. The following are the modifications that need to be made to the existing SDK files to achieve Sendata functionality. Line numbers mark the line of the original file where the change occurred based on the output of the GNU Linux *diff* command.

SDKDIR/ARDroneLib/Soft/Build/custom.makefile

- 17: Change `USE_LINUX = no` to `yes`

SDKDIR/ARDroneLib/Soft/Lib/Build/Makefile

- 59: Append `$(ARDRONE_TOOL_DIR)/Sendata/sendata.c` to list of `GENERIC_LIBRARY_SOURCE_FILES`
- 82: Append `$(ARDRONE_TOOL_DIR)/Sendata/ardrone_general_sendata.c` to list of `GENERIC_LIBRARY_SOURCE_FILES`
- 90: Append `$(ARDRONE_TOOL_DIR)/Sendata/ardrone_sendata_client.c` and `$(ARDORNE_TOOL_DIR)/Control/ardrone_sendata_control.c` to list of `GENERIC_LIBRARY_SOURCE_FILES`

SDKDIR/Examples/Linux/sdk_demo/Build/Makefile

- 30: Append `Sendata/sendata.c` to list of `GENERIC_BINARIES_COMMON_SOURCE_FILES`

SDKDIR/Examples/Linux/sdk_demo/Sources/ardrone_testing_tool.c

- 10: Add `#include <ardrone_tool/Sendata/ardrone_sendata_client.h>` to header
- 36: Change `gamepad` to `ps3pad`
- 40: Add

```
ardrone_sendata_client_init();
START_THREAD( sendata_update, NULL );
```

- 51: Change `gamepad` to `ps3pad`
- 52: Add

```
ardrone_sendata_client_shutdown();
JOIN_THREAD( sendata_update );
```

- 73: Add `THREAD_TABLE_ENTRY( sendata_update, 20 )`

SDKDIR/Examples/Linux/sdk_demo/Sources/Navdata/navdata.h

- 5: Add `const navdata_unpacked_t* snav;`

SDKDIR/Examples/Linux/sdk_demo/Sources/Navdata/navdata.c

- 13: Replace function definition with:

```
inline C_RESULT demo_navdata_client_process( const navdata_unpacked_t* const navdata )
{
    const navdata_demo_t*nd = &navdata->navdata_demo;
    snav=navdata;//make navdata accessible to sendata thread

    printf("                                                                    \n");
    printf("===================== Navdata for flight demonstrations =====================\n");

    printf("Control state : %i  Battery level : %i mV  Altitude : %i          \n", \
           nd->ctrl_state,nd->vbat_flying_percentage,nd->altitude);
    printf("Orientation   : [Theta] %4.3f  [Phi] %4.3f  [Psi] %4.3f            \n", \
           nd->theta,nd->phi,nd->psi);
    printf("Speed         : [vX] %4.3f  [vY] %4.3f  [vZ] %4.3f                 \n", \
           nd->vx,nd->vy,nd->vz);

    printf("\n\n\n\n");
    printf("\033[9A");

   return C_OK;
```

IV. Running the Sendata Server

The C based platform independent nature of the SDK requires the extensive use of preprocessor directives. Unfortunately, the software environment and compiler supplied with the Arduino do not function properly when certain preprocessor directives are used in the source files. Thus it is necessary to manually compile and upload the Sendata server program to the Arduino.

A. Installation of Arduino Software

1) Download the Arduino environment from arduino.cc/en/Main/Software[1]
2) Extract the files from the archive to `EXDIR`, where `EXDIR` is a directory of your choosing
3) Copy the contents of the `EXDIR/hardware/arduino` directory to `SDKDIR/Arduino/src`
4) Copy the `EXDIR/libraries` directory to `SDKDIR/Arduino/src`

Note: The Arduino environment can be run via command line by calling `./EXDIR/arduino`.

B. Compiling & Uploading the Sendata Server

Before uploading the Sendata server, first configure the transceiver as defined in Section II-A.
The following instructions are to be executed via command line:

1) Install the Server Makefile in `SDKDIR/Arduino`
2) Go to the `SDKDIR/Arduino` directory
3) Call `make`, the server software will compile
4) Connect the Arduino to the Terminal
5) Call `make upload`, the server program will upload to the Arduino
6) Call `make clean` to remove the object files

The Sendata server will now automatically run when the Arduino is powered.
Please refer to the annotations in the Arduino Makefile if the `make` commands do not execute properly.

[1]Sendata was developed using arduino-1.0.1-linux64

V. Running Sendata in the API

Sendata was originally integrated into the *Linux sdk_demo* for simplicity. Once the client libraries are installed and the necessary SDK files are modified, the demo can be easily built to include Sendata by simply following the instructions in the ARD API Reference manual.

sendata.c located in `SDKDIR/Examples/Linux/sdk_demo/Sources/Sendata` is where all of the data relayed through Sendata should be processed. This file obeys the threading structure of the API and all operations should occur or be called from `demo_sendata_client_process`. Of special note in this function is the display portion. This and its counterpart in `demo_navdata_client_process` in *navdata.c* have been formatted such that the information from the two independent process threads is displayed simultaneously. Thus, the total number of display lines in each function must remain constant and the number of blank lines at the top or bottom of each must match the number of non-blank lines at the bottom or top of the other.

A. Demo Makefile

To ease the process of compiling and running the various Linux Demo programs and management of the SDK, a special Makefile located in Section G of the Appendix was written. When installed in the `SDKDIR/Examples/Linux/Build` directory one can use this Makefile by calling `make <COMMAND>` from that directory. Calling `make help` will display the list of available commands.

B. PS3 Controller

One reason the *sdk_demo* was chosen was the ability to use a PlayStation 3 controller to override the actions of the ARD (Note: the Linux examples do not support keyboard control of the ARD). For this to work a genuine PS3 controller must be used.

VI. Changing Sendata

Sendata was designed to allow the user to relay any data from the server to the client. To change what Sendata contains one must alter the following files:

- `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Sendata/sendata_common.h`
- `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Sendata/sendata_keys.h`
- `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Sendata/sendata_server.h`
- `SDKDIR/ARDroneLib/Soft/Lib/ardrone_tool/Sendata/sendata_server.c`
- `SDKDIR/Arduino/source/arduino_server.cpp`

The following are the details of the necessary changes that need to be made to each of these files.
To ensure proper operation, all variables in these files must use the data types found in the `stdint` library.

A. sendata_common.h

sendata_common.h is where the Sendata structures (referred to as options) and data types are declared; options can be added, removed, or modified. The format for a structure is as follows:

```
typedef struct _<STRUCTNAME>_t {
 uint16_t tag;
 uint16_t size;

 <DATAMEMBERS>

} _ATTRIBUTE_PACKED_ <STRUCTNAME>_t;
```

A copy of the `ir_measures_t` structure is located in Section D of the Appendix for reference.

B. sendata_keys.h

Each of the options in *sendata_common.h* must be listed in *sendata_keys.h* in the following format:

```
SENDATA_OPTION( <OPTIONNAME>_t,          <OPTIONNAME>          , <OPTIONNAMEALLCAPS>_TAG          )
```

This produces the tags for the options and is used by the client and server APIs to pack and unpack the data streams.
For reference, a copy of *sendata_keys.h* is located in Section F of the Appendix.

C. sendata_server.h

sendata_server.h is used by the server to manage the Sendata structure and its transmission. In *sendata_server.h* the structure `sendatas_t` is declared. This structure is used as a wrapper for the `sendata_t` structure and all of its options. If a option is added or removed, the same must be done in `sendatas_t`. For reference, a copy of `sendatas_t` can be found in Section E of the Appendix.

D. sendata_server.c

Like *sendata_server.h*, *sendata_server.c* is used by the server to manage the Sendata structure and its transmission. It contains two functions: `initialize` and `pack`. If an option is removed, the corresponding lines in the functions must be removed, and if one is added the following must be added to the respective functions:

- initialize

```
sendatas-><OPTIONNAME>.tag=<OPTIONTAG>;
sendatas-><OPTIONNAME>.size=sizeof(<OPTIONNAME>_t);
```

- pack

```
traveler=(uint8_t*) ardrone_sendata_pack(traveler, sendatas-><OPTIONNAME>);
```

For reference, a copy of *sendata_server.c* is located in Section H of the Appendix.

E. arduino_server.cpp

arduino_server.cpp is where the Sendata protocol and the Arduino API meet. It is in this file where the variables in the Sendata options are assigned values from the Arduino's systems. Additional functions and libraries can be added to this file to retrieve the desired data from the Arduino (be it an internal reading such as the CPU clock or the reading from an external device via the Arduino's I/O). This data must then be passed to the relevant Sendata option for it to be received by the client. For reference, a copy of *arduino_server.cpp* is located in Section B of the Appendix.

APPENDIX

A. serialfor

The following is the Arduino code for the *serialfor* sketch. This sketch was written assuming that the RN-XV is using the default baud rate and command character.

```
//constants declarations

#define WIFI_SERIAL Serial3
#define WIFI_BAUD 9600
#define WIFI_RST_PIN 40

void setup(void)
{
  Serial.begin(9600);
  WIFI_SERIAL.begin(WIFI_BAUD);
  pinMode(WIFI_RST_PIN, OUTPUT);

  //reboot wifi chip
  digitalWrite(WIFI_RST_PIN, LOW);
  delay(1000);
  digitalWrite(WIFI_RST_PIN, HIGH);
  delay(500);
}

void loop ()
{
  if(Serial.available()>0)
   {
     if(Serial.peek()=='$')  //if comm char don't send the rest of the inbound buffer
     {
       Serial3.write("$$$");
       Serial3.flush();
       delay(250);
     }
     Serial3.write(Serial.read());
   }
  if(Serial3.available()>0)
    {
      Serial.write(Serial3.read());
      Serial.flush();
    }
}
```

B. arduino_server.cpp

```
/*********************************************************
 ** arduino_server.cpp
 **
 ** Server core program to be loaded on to Arduino.
 ** This is the primary file to be compiled by the
 ** Arduino compiler and used to run the outboard sensor
 ** array.
 **
 ** (C) J.Weeks Tufts University 07/26/2012
 **
 *********************************************************/

#define _ARDUINO_
#include <arduino_server.h>
#include <sendata_server.h>
#include <sendata_common.h>
#include <sendata_config.h>
#include <Arduino.h>
#include <stdint.h>

//constant declarations
#define WIFI_SERIAL Serial3
#define WIFI_BAUD 9600
#define WIFI_RST_PIN 40
#define MATCHCHAR '#'
#define S0_0_PIN A0
#define S90_0_PIN A1
#define S180_0_PIN A2
#define S270_0_PIN A3


//variable declarations
sendatas_t sendatas;
static uint8_t buff_head[SENDATA_MAX_SIZE];
uint8_t* buff_tail;
uint16_t* s0_0p;
uint16_t* s90_0p;
uint16_t* s180_0p;
uint16_t* s270_0p;

void setup(void)
{
  delay(30000); //wait for drone to come online
  //Serial.begin(9600);  //Debug
  WIFI_SERIAL.begin(WIFI_BAUD);
  pinMode(WIFI_RST_PIN, OUTPUT);
  initialize(&sendatas);

  //reboot wifi chip
  digitalWrite(WIFI_RST_PIN, LOW);
  delay(1000);
  digitalWrite(WIFI_RST_PIN, HIGH);
  delay(500);

  //config_wifi();
  //initilize sensor vars
  s0_0p=&(sendatas.ir_measures.s0_0);
  *s0_0p=0;
  s90_0p=&(sendatas.ir_measures.s90_0);
  *s90_0p=0;
  s180_0p=&(sendatas.ir_measures.s180_0);
  *s180_0p=0;
  s270_0p=&(sendatas.ir_measures.s270_0);
  *s270_0p=0;
}

void loop(void)
{
  //read sensor values
  *s0_0p=analogRead(S0_0_PIN);
  *s90_0p=analogRead(S90_0_PIN);
  *s180_0p=analogRead(S180_0_PIN);
  *s270_0p=analogRead(S270_0_PIN);
  //always last part of loop
  sendatas.sendata_time.time=micros();
```

```
  buff_tail=pack(buff_head, &sendatas);
  WIFI_SERIAL.flush();
  WIFI_SERIAL.write(buff_head, buff_tail-buff_head);
  WIFI_SERIAL.flush();
  WIFI_SERIAL.print(MATCHCHAR);
  //Debug
  //Serial.write(buff_head, buff_tail-buff_head);
  //Serial.println();
}


// /**
//  * @fn serialEvent3
//  * @brief debugging for wifi chip, forwards data to terminal
//  */
// void serialEvent3()
// {
//   while(Serial3.available()>0)
//     {
//       Serial.write(Serial3.read());
//     }
// }
```

C. RN-XV Configuration Commands

Before any settings can be configured, the RN-XV must be put into command mode. This is done by sending `$$$` with no line ending, where '$' is the command mode access character if it has been changed from the default. After this string is entered one must wait at least 250ms before sending another command. The RN-XV should respond by printing `CMD` once it enters command mode.

The following are the exact configuration commands. When a command is properly received the RN-XV will respond by sending back `AOK`. The '\r' denote carriage returns, the Arduino Serial Monitor must be set to use them as line endings.

```
set wlan ssid ard\r
set ip proto 1\r
set ip dhcp 0\r
set ip address 192.168.1.3\r
set ip host 192.168.1.2\r
set ip remote 5574\r
set ip local 5574\r
set ip mtu 1500\r
set comm time 0\r
set comm size 1420\r
set comm match 35\r
save\r
reboot\r
```

Once these settings are entered they can be verified by returning to command mode and sending `get e\r`. Command mode is exited by sending `exit\r`.

D. ir_measures_t

```
/**
 * @brief IR sensor measurements
 * @nominclature x_y are horizontal cw and vertical "up first" degrees
 * ie 0_90 is straight up, 90_0 is dead starboard
 */
typedef struct _ir_measures_t {
  uint16_t  tag;
  uint16_t  size;

  uint16_t  s0_0;
  uint16_t  s90_0;
  uint16_t  s180_0;
  uint16_t  s270_0;

} _ATTRIBUTE_PACKED_ ir_measures_t;
```

E. sendatas_t

```
/**
 * @struct _sendatas_t
 * @brief Wrapper struct for sendata structures
 */
typedef struct _sendatas_t
{
  sendata_t sendata;
  //include all structures to be used/sent to client
  ir_measures_t ir_measures;
  sendata_time_t sendata_time;
  sendata_cks_t sendata_cks;
} sendatas_t;
```

F. sendata_keys.h

```
/*****************************************************
 **  sendata_keys.h
 **
 **  This is a modified version of navdata_keys.h rewritten
 **  to accomidate data input from an outboard sensor array.
 **
 **  (C) J.Weeks Tufts University 07/18/2012
 **
 **  navdata_keys.h (C) PARROT SA 2007-2011
 **
 *****************************************************/

#ifndef SENDATA_OPTION
   #define SENDATA_OPTION(x,y,z)
#endif
#ifndef SENDATA_OPTION_CKS
   #define SENDATA_OPTION_CKS(x,y,z)
#endif

SENDATA_OPTION( ir_measures_t,          ir_measures           , IR_MEASURES_TAG          )
SENDATA_OPTION( sendata_time_t,         sendata_time          , SENDATA_TIME_TAG         )
SENDATA_OPTION( sendata_watchdog_t,     sendata_watchdog      , SENDATA_WATCHDOG_TAG     )

SENDATA_OPTION_CKS( sendata_cks_t,      sendata_cks           , SENDATA_CKS_TAG          )


#undef SENDATA_OPTION
#undef SENDATA_OPTION_CKS
```

G. Demo Makefile

```
#shortcut for building SDK demos
#(C) J.Weeks Tufts University 2011/08/12

#local sdkdir/Examples/Linux/Build/
#navtarg sdkdir/Examples/Linux/Navigation/Build
#demtarg sdkdir/Examples/Linux/sdk_demo/Build

#Release versions

nav:
    @cd -P ../Navigation/Build && \
    make

cnav:
    @cd -P ../Navigation/Build && \
    make clean && \
    cd -P ../../../../ARDroneLib/Soft/Build && \
    rm -frv targets_versions

dem:
    @cd -P ../sdk_demo/Build && \
    make

cdem:
    @cd -P ../sdk_demo/Build && \
    make clean && \
    cd -P ../../../../ARDroneLib/Soft/Build && \
    rm -frv targets_versions

#Debug Versions

dnav:
    @cd -P ../Navigation/Build && \
    make RELEASE_BUILD=no

cdnav:
    @cd -P ../Navigation/Build && \
    make clean RELEASE_BUILD=no && \
    cd -P ../../../../ARDroneLib/Soft/Build && \
    rm -frv targets_versions

ddem:
    @cd -P ../sdk_demo/Build && \
    make RELEASE_BUILD=no

cddem:
    @cd -P ../sdk_demo/Build && \
    make clean RELEASE_BUILD=no && \
    cd -P ../../../../ARDroneLib/Soft/Build && \
    rm -frv targets_versions

#Running

cls:
    clear

rst:
    reset

rdd:    rst ddem cls
    ./Debug/linux_sdk_demo

rd:   rst dem cls
    ./Release/linux_sdk_demo

#Cleaning

clean:  cnav cdem cdnav cddem
    @rm -frv Release/common Debug/common
    @find . -type f \! -name *.xml \! -name Makefile \
    -print0 | xargs -0 /bin/rm -fv

rmtmp:
    @cd ../ && \
    find . -type f -name *~ -print0 | xargs -0 /bin/rm -v
```

```
help:
    @echo Release Builds: nav, cnav, dem, cdem
    @echo Debug Builds: dnav, cdnav, ddem, cddem
    @echo Quick Run: rdd \(ddem\), rd \(dem\)
    @echo Remove all non-config files: clean
    @echo Remove all tmp files: rmtmp
```

H. sendata_server.c

```
/*****************************************************
 **  sendata_server.c
 **
 **  Server for Arduino responsible for initialization and
 **  packing of sendata for transmission to transceiver
 **
 **  (C) J.Weeks Tufts University 07/25/2012
 **
 *****************************************************/

#include <sendata.h>
#include <sendata_server.h>
#include <sendata_common.h>
#include <sendata_config.h>

/**
 * @fn initialize
 * @brief initializes sendata_t contents
 * @pram sendatas Sendatas to be initialized
 */
void initialize(sendatas_t* sendatas)
{
  sendatas->sendata.header=SENDATA_HEADER;
  sendatas->sendata.sequence=SENDATA_SEQUENCE_DEFAULT;
  sendatas->ir_measures.tag=IR_MEASURES_TAG;
  sendatas->ir_measures.size=sizeof(ir_measures_t);
  sendatas->sendata_time.tag=SENDATA_TIME_TAG;
  sendatas->sendata_time.size=sizeof(sendata_time_t);
  sendatas->sendata_cks.tag=SENDATA_CKS_TAG;
  sendatas->sendata_cks.size=sizeof(sendata_cks_t);
}

/**
 * @fn pack
 * @brief packs sendatas for UDP transmission
 * @pram sendata_ptr Pointer to start of transmission buffer
 * @param sendatas Sendatas to be transmitted
 */
uint8_t* pack(uint8_t* sendata_pointer, sendatas_t* sendatas)
{
  uint8_t* traveler=sendata_pointer;
  //load sendata into buffer
  sendatas->sendata.sequence++; //set sequence to next
  traveler=sendata_pack_option(traveler, (uint8_t*) &sendatas->sendata, sizeof(sendatas->sendata));
  traveler-=sizeof(sendata_option_t);
  //load rest of sendatas
  traveler=(uint8_t*) ardrone_sendata_pack(traveler, sendatas->sendata_time);
  traveler=(uint8_t*) ardrone_sendata_pack(traveler, sendatas->ir_measures);
  //compute and store checksum
  sendatas->sendata_cks.cks=ardrone_sendata_compute_cks(sendata_pointer, traveler-sendata_pointer);
  //bugfix add 255 to checksum
  sendatas->sendata_cks.cks+=255;
  //load cks_t to buffer
  traveler=(uint8_t*) ardrone_sendata_pack(traveler, sendatas->sendata_cks);
  return traveler;
}
```