LTM Receiver via LoRa (LTM Decoder)

DuPe (www.dupedup.cz)
Prototype Version: 2/2025

Purpose:

Receive Light Telemetry Protocol (LTM) from RC model on the Ground station via Long Range Radio (433 MHz) up to 3 km. The LTM includes flight data from Flight Controller. Decode LTM flight data to display it on TFT in ground station ando also save raw LTM data on SD Card.
Fligt data is particularly concerned of board voltage, GPS data (latitude, longitude, altitude, ground speed) and status of flight modes. Also is computes data from GPS (flight time , distance from home, flight and home heading).

Hardware components:

1) **MCU STM32F103C8T6 (Blue Pill)**
   - STM32F103C8T6 Development Board based on an ARM 32bit Cortex-M3 (maximum frequency of 72 MHz)
   - Memories: 64 Kbytes of Flash and 20 Kbytes of SRAM
   - GPIO Pins – 32 with external interrupt capability
   - Timers – 3 16-bit Timers, 1 16-bit PWM Timer
   - PWM Pins – 15
   - Analog – 10 Channels of 12-bit ADC
   - I2C – 2 I2C Peripherals
   - USART – 3 USART Peripherals with hardware control
   - SPI – 2 SPI Peripherals
   - USB 2.0 Full Speed, CAN 2.0B
   - A Reset Switch – to reset the Microcontroller
   - microUSB port – for serial communication and power
   - BOOT Selector Jumpers – BOOT0 and BOOT1 jumpers for selecting the booting memory
   - Two LEDs – User LED and Power LED
   - 8 MHz Crystal – Main Clock for MCU
   - 32.768KHz Oscillator – RTC Clock
   - SWD Interface – for programming and debugging using ST-Link.
   - 3.3V regulator – converts 5V to3.3V for poweringthe MCU.

2) **LoRa Radio Modul E32-TTL-100**
   - 100mW wireless transceiver module with LoRa spread-spectrum technology, operates at 410~441MHz (Default: 433MHz), based on original RFIC SX1278 from SEMTECH, transparent transmission is available, TTL level
   - 21* 36mm / 6.7g
   - Supply voltage : 2.3 ~ 5.5V DC
   - Operation Range 3000m (Clear and open area, 20dBm, antenna gain: 5dBi， height: 2m， air date rate: 2.4kbps)
   - Transmitting power 20dBm 4 optional level（20,17,1410 dBm)
   - Air data rate 2.4kbps6 optional level （0.3,1.2,2.4,4.8,9.6,19.2 kbps）
   - Transmitting current 120mA@20dBm
   - Receiving current 14mA
   - UART (8N1、8E1、8O1， Eight kinds of UART baud Rate, from 1200 to 115200 bps （Default: 9600）
   - 512 bytes buffer (58 bytes per package)
   - Antenna type SMA-K 50Ω impedance

3) **Display TFT 4.0" 480x320**
   - 4.0" 480x320 TFT (modul MSP4020) driver ST7796
   - colors: 65K

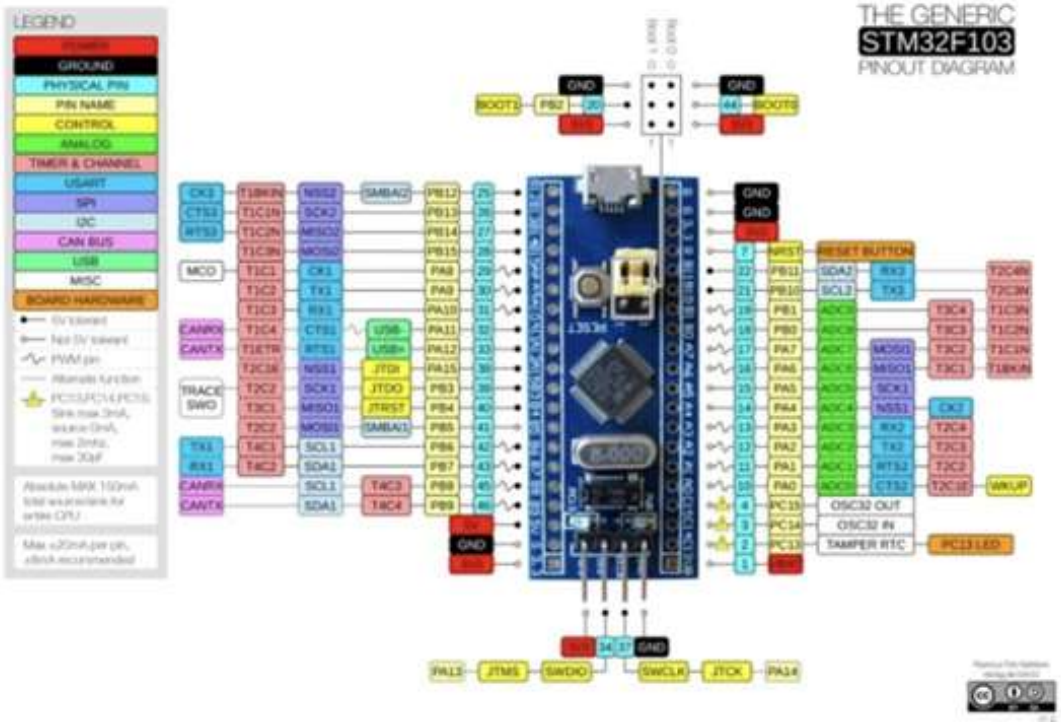
- SPI
- touch SPI, XPT2046
- 3,3-5V
- 61,74x108,04mm
- SD SPI

4) **DC-DC Step Down Modul (3.3V)**
   - INPUT: 5.3-26V, MP1584
   - OUTPUT: 3.3V / 1.8A - 3A , 22 x17,5 x 4,5mm

5) **LiPo 2-3S 1000 mAh**

Circuit

STM32 Modul	**TFT** (SPI)	**SD CARD** (SPI)	**LORA MODUL** (UART)	**STEP DOWN** (INPUT 6-23V)
PA0	RESET			
PA1	DC / A0 /RS			
PA4 SPI1	CS			
PA5	CLK	CLK		
PA7 SPI1	SDA / MOSI	MOSI		
PA6		MISO		
PA10 RX1			TXD	
PA 9 TX1			RXD	
PA3 RX2				
PA2 TX2				
GND	GND		GND, M0, M1	GND
3.3V	VCC , LED VCC		VCC	OUTPUT 3V





Current Consumption: 50mA (3S LiPo)

Software Components:

- VSCode + Platform IO + STM32 support
- GitHub - Bodmer/TFT_eSPI: Arduino and PlatformIO IDE compatible TFT library optimised for the Raspberry Pi Pico (RP2040), STM32, ESP8266 and ESP32 that supports different driver chips
- Home · KipK/Ghettostation Wiki · GitHub

Processing:

FLIGT DATA (FLIGHT CONTROLLER) -> LORA SENDER -> LORA RADIO (Up To eg. 3km) -> LORA RECEIVER -> SERIAL DATA BUFFER -> DECODE LTM-> DISPLAY

Code:

platformio.ini

```
[env:bluepill_f103c8]
platform = ststm32
board = bluepill_f103c8
framework = arduino
debug_tool = stlink
upload_protocol = stlink
lib_deps =
        bodmer/TFT_eSPI@^2.5.43
        arduino-libraries/SD@^1.3.0
```

LightTelemetry.h

```
#define LIGHTTELEMETRY_START1 0x24 //$ HEADER '$'
#define LIGHTTELEMETRY_START2 0x54 //T  HEADER 'T'
#define LIGHTTELEMETRY_GFRAME 0x47 //G GPS + Baro altitude data ( Lat, Lon, Speed, Alt, Sats, Sat fix)
#define LIGHTTELEMETRY_AFRAME 0x41 //A Attitude data ( Roll,Pitch, Heading )
#define LIGHTTELEMETRY_SFRAME 0x53 //S Sensors/Status data ( VBat, Consumed current, Rssi, Airspeed, Arm status, Failsafe status, Flight mode )
#define LIGHTTELEMETRY_OFRAME 0x4F //O  Origin data (home lon, home lat, homefix)
#define LIGHTTELEMETRY_NFRAME 0x4E //N  NAVIGATION data
#define LIGHTTELEMETRY_XFRAME 0x58 //X  EXTRA FRAME
```

```
#define LIGHTTELEMETRY_GFRAMELENGTH 18
#define LIGHTTELEMETRY_AFRAMELENGTH 10
#define LIGHTTELEMETRY_SFRAMELENGTH 11
#define LIGHTTELEMETRY_OFRAMELENGTH 18
#define LIGHTTELEMETRY_NFRAMELENGTH 10
#define LIGHTTELEMETRY_XFRAMELENGTH 10

void telemetry_off(void);
void ltm_read();
void ltm_check();

//Global variables
int32_t       LTM_pkt_ko = 0;   //wrong LTM packet counter
int32_t       LTM_pkt_ok = 0;   //good LTM packet counter
int32_t       uav_homelat = 0; //home latitude
int32_t       uav_homelon = 0; //home longitude
int32_t       uav_homealt = 0; //home altitude
int32_t       ground_course = 0; //course over ground
int32_t       ground_distance = 0; //distance bettween two consecutive GPS points in actual course (unused) // TODO
use this to calc SPEED
uint8_t       uav_osd_on = 0; // allways 1
int32_t      home_distance = 0; // distance to home
int32_t       home_heading = 0; // heading to home
uint8_t       uav_homefixstatus = 0; // 1=homefix ok
uint8_t       uav_gpsmode = 0;
uint8_t       uav_navmode = 0;
uint8_t       uav_navaction = 0;
uint8_t       uav_WPnumber = 0; // Waypoint number
uint8_t       ltm_naverror = 0;
uint8_t       ltm_flags = 0;
uint16_t      uav_HDOP = 0; // GPS HDOP
uint8_t       uav_HWstatus = 0; // hardware error
uint8_t       uav_spare1 = 0;
uint8_t       uav_spare2 = 0;
uint8_t       ltm_spare3 = 0;
int32_t      uav_lat = 0;                  // latitude
int32_t      uav_lon = 0;                   // longitude
float        lonScaleDown = 0.0;             // longitude scaling
uint8_t      uav_satellites_visible = 0;     // number of satellites
uint8_t      uav_fix_type = 0;              // GPS lock 0-1=no fix, 2=2D, 3=3D
int32_t      uav_alt = 0;                  // altitude (dm)
int32_t      rel_alt = 0;                 // relative altitude to home
uint16_t     uav_groundspeed = 0;            // ground speed in km/h
uint8_t      uav_groundspeedms = 0;          // ground speed in m/s
int16_t      uav_pitch = 0;                 // attitude pitch
int16_t      uav_roll = 0;                 // attitude roll
int16_t      uav_heading = 0;               // attitude heading
int16_t      uav_gpsheading = 0;             // gps heading
uint16_t     uav_bat = 0;                  // battery voltage (mv)
uint16_t     uav_amp = 0;                   // consumed mah.
uint16_t     uav_current = 0;               // actual current
uint8_t      uav_rssi = 0;                 // radio RSSI (%)
uint8_t      uav_linkquality = 0;           // radio link quality
uint8_t      uav_airspeed = 0;              // Airspeed sensor (m/s)
uint8_t      ltm_armfsmode = 0;
uint8_t      uav_arm = 0;                  // 0: disarmed, 1: armed
uint8_t      uav_failsafe = 0;              // 0: normal,   1: failsafe
uint8_t      uav_flightmode = 19;
//long lastpacketreceived;
boolean gps_fix      = false;
//boolean btholdstate  = false;
//boolean telemetry_ok = false;
//boolean home_pos     = false;
//boolean home_bear    = false;
```

MCU Source Code

```
/*-----------------------------------------------------
* LIGHT TELEMETRY (LTM) LORA RECEIVER ver 4.4
* Receive LTM Data via LoRa 433MHz - Write RAW LTM Data to SD Card - Display Decoded LTM Data on TFT
* DUPE 20220730, 20240722 (NEW TFT 480x320 ST7796S)
* CO: https:*github.com/KipK/Ghettostation/wiki
* HW MCU: STM32F103C (BLUEPILL)
* HW LORA: RECEIVER E32TTL100
* HW TFT/SD: 480x320ST7796(SPI)
* UART1: RX->PA9,TX->PA10  (LoRa)
* SPI1: SDA/MOSI-PA7, CLK/SCK-PA5, CS-PA4, RST-PA0, RS/DC-PA1 (TFT)
* SPI2: MOSI-PB15, MISO-PB14, SCK-PB13, CS-PB12 (SD)
* VCC 3.3V - step down from 3S (12V)
-----------------------------------------------------*/
#include <Arduino.h>
#ifdef membug
#include <MemoryFree.h>
#endif
#include <TFT_eSPI.h>
#include <SPI.h>
#include <SD.h>
#include "LightTelemetry.h"
#include <stdint.h>
#define SD_CS  PB0
#define LTM_BAUDS     9600
// HardwareSerial Serial1(PA10, PA9);
TFT_eSPI tft = TFT_eSPI();
//nutne editovat lib TFT_eSPI User Setup.h pro spravny driver !!
File dataFile;
uint8_t gps_minsatfix = 4;
int32_t gps_range = 10000000;
uint16_t max_alt = 1000;
uint16_t min_speed = 5;
uint16_t max_speed = 250;
uint8_t gps_ok = 0;
int32_t gps_lastlat = 0;
int32_t gps_lastlon = 0;
int32_t gps_minlon = 0;
int32_t gps_maxlon = 0;
int32_t gps_minlat = 0;
int32_t gps_maxlat = 0;
int32_t route_distance = 0;
uint32_t flight_start = 0;
uint32_t flight_time = 0;
uint8_t  existFile = 1;
uint8_t numberFile = 1;
String errMessage ="Start-LTM-OK";
String prefixnameFile = "/ltm";
String nameFile = "/ltm1.dat";
static uint8_t LTMBuffer[LIGHTTELEMETRY_GFRAMELENGTH - 4];
static uint8_t LTMreceiverIndex;
static uint8_t LTMcmd;
static uint8_t LTMrcvChecksum;
static uint8_t LTMreadIndex;
static uint8_t LTMframelength;
uint32_t row = 0;
uint32_t display_clear = 0;
String flightmode = "N";
String navmode ="N";
String gpsmode ="N";
String arm = "N";
String LTM_pkt = "N";
int32_t LTM_lastpkt_ok = 0;
```

```
uint8_t ltmread_u8()  {
  return LTMBuffer[LTMreadIndex++];
}
uint16_t ltmread_u16() {
  uint16_t t = ltmread_u8();
  t |= (uint16_t)ltmread_u8() << 8;
  return t;
}
uint32_t ltmread_u32() {
  uint32_t t = ltmread_u16();
  t |= (uint32_t)ltmread_u16() << 16;
  return t;
}
void ltm_read() {
    uint8_t c;
    static enum _Serial1_state {
    IDLE,
    HEADER_START1,
    HEADER_START2,
    HEADER_MSGTYPE,
    HEADER_DATA
  }
  c_state = IDLE;
  while (Serial1.available()) {
    c = char(Serial1.read());
   // Serial.printf("%02x ", c);
    if (dataFile) {
      dataFile.write(c);
      errMessage = "Writing " + nameFile;
      }
    else {errMessage = "ERROR write to file";}
    if (c_state == IDLE) {
      c_state = (c == '$') ? HEADER_START1 : IDLE;
    }
    else if (c_state == HEADER_START1) {
      c_state = (c == 'T') ? HEADER_START2 : IDLE;
    }
    else if (c_state == HEADER_START2) {
      switch (c) {
        case 'G':
          LTMframelength = LIGHTTELEMETRY_GFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
        case 'A':
          LTMframelength = LIGHTTELEMETRY_AFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
        case 'S':
          LTMframelength = LIGHTTELEMETRY_SFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
        case 'O':
          LTMframelength = LIGHTTELEMETRY_OFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
        case 'N':
          LTMframelength = LIGHTTELEMETRY_NFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
        case 'X':
          LTMframelength = LIGHTTELEMETRY_XFRAMELENGTH;
          c_state = HEADER_MSGTYPE;
          break;
```

```
        default:
          c_state = IDLE;
      }
      LTMcmd = c;
      LTMreceiverIndex = 0;
    }
    else if (c_state == HEADER_MSGTYPE) {
      if (LTMreceiverIndex == 0) {
        LTMrcvChecksum = c;
      }
      else {
        LTMrcvChecksum ^= c;
      }
      if (LTMreceiverIndex == LTMframelength - 4) {
        if (LTMrcvChecksum == 0) {
          LTM_pkt_ok++;
          ltm_check();
          c_state = IDLE;
        }
        else {

          LTM_pkt_ko++;
          c_state = IDLE;
        }
      }
      else LTMBuffer[LTMreceiverIndex++] = c;
    }
  }
}
// Decoded received commands
void ltm_check() {
  LTMreadIndex = 0;
  if (LTMcmd == LIGHTTELEMETRY_GFRAME)
  {
    uav_lat = (int32_t)ltmread_u32();
    uav_lon = (int32_t)ltmread_u32();
    uav_groundspeedms = ltmread_u8();
    uav_groundspeed = (uint16_t) round((float)(uav_groundspeedms * 3.6f)); // convert to kmh
    uav_alt = (int32_t)ltmread_u32()/100;//convert to m
    uint8_t ltm_satsfix = ltmread_u8();
    uav_satellites_visible = (ltm_satsfix >> 2) & 0xFF;
    uav_fix_type = ltm_satsfix & 0b00000011;
    memset(LTMBuffer, 0, LIGHTTELEMETRY_GFRAMELENGTH - 4);
  }
  if (LTMcmd == LIGHTTELEMETRY_AFRAME)
  {
    uav_pitch = (int16_t)ltmread_u16();
    uav_roll =  (int16_t)ltmread_u16();
    uav_heading = (int16_t)ltmread_u16();
    if (uav_heading < 0 ) uav_heading = uav_heading + 360; //convert from -180/180 to 0/360°
    memset(LTMBuffer, 0, LIGHTTELEMETRY_AFRAMELENGTH - 4);
  }
  if (LTMcmd == LIGHTTELEMETRY_SFRAME)
  {
    uav_bat = ltmread_u16();
    uav_amp = ltmread_u16();
    uav_rssi = (ltmread_u8()*100)/255;
    uav_airspeed = ltmread_u8()*100;
    ltm_armfsmode = ltmread_u8();
    uav_arm = ltm_armfsmode & 0b00000001;
    uav_failsafe = (ltm_armfsmode >> 1) & 0b00000001;
    uav_flightmode = (ltm_armfsmode >> 2) & 0b00111111;
    switch (uav_flightmode) {
    case 0: flightmode = "MANUAL";break;
```

```
    case 1: flightmode = "RATE";break;
    case 2: flightmode = "ANGLE";break;
    case 3: flightmode = "HORIZON";break;
    case 4: flightmode = "ACRO";break;
    case 8: flightmode = "ALTHOLD";break;
    case 9: flightmode = "GPSHOLD";break;
    case 10: flightmode = "WAYPOINT";break;
    case 12: flightmode = "CIRCLE";break;
    case 13: flightmode = "RTH";break;
    case 15: flightmode = "LAND";break;
    case 18: flightmode = "CRUISE";break;
    default: flightmode = String(uav_flightmode);
    }
    switch (uav_arm) {
    case 0: arm = "NOARM";break;
    case 1: arm = "ARMED";break;
    }
    memset(LTMBuffer, 0, LIGHTTELEMETRY_SFRAMELENGTH - 4);
  }
  if (LTMcmd == LIGHTTELEMETRY_OFRAME) // origin frame
  {
    uav_homelat = (int32_t)ltmread_u32();
    uav_homelon = (int32_t)ltmread_u32();
    uav_homealt = (int32_t)ltmread_u32();
    uav_osd_on = (int8_t)ltmread_u8(); // always 1
    uav_homefixstatus = (int8_t)ltmread_u8();
    memset(LTMBuffer, 0, LIGHTTELEMETRY_OFRAMELENGTH - 4);
  }
  if (LTMcmd == LIGHTTELEMETRY_NFRAME)
  {
    uav_gpsmode = ltmread_u8();
    uav_navmode = ltmread_u8();
    uav_navaction = ltmread_u8();
    uav_WPnumber = ltmread_u8();
    ltm_naverror = ltmread_u8();
    ltm_flags = ltmread_u8();
    switch (uav_navmode) {
    case 0: navmode = "NONE";break;
    case 1: navmode = "RTH-START";break;
    case 2: navmode = "RTH-ROUTE";break;
    case 5: navmode = "WP-ROUTE";break;
    case 13: navmode = "RTH-HOVER";break;
    default: navmode = String(uav_navmode);
    }
    switch (uav_gpsmode) {
    case 0: gpsmode = "NONE";break;
    case 1: gpsmode = "POSHOLD";break;
    case 2: gpsmode = "RTH";break;
    case 3: gpsmode = "MISSION";break;
    default: gpsmode = String(uav_gpsmode);
    }
    memset(LTMBuffer, 0, LIGHTTELEMETRY_NFRAMELENGTH - 4);
  }
  if (LTMcmd == LIGHTTELEMETRY_XFRAME)
  {
    uav_HDOP = ltmread_u16();
    uav_HWstatus = ltmread_u8();
    uav_spare1 = ltmread_u8();
    uav_spare2 = ltmread_u8();
    ltm_spare3 = ltmread_u8();
    memset(LTMBuffer, 0, LIGHTTELEMETRY_XFRAMELENGTH - 4);
  }
}
// distance and direction between two points
```

```
void GPS_dist(int32_t* lat1, int32_t* lon1, int32_t* lat2, int32_t* lon2, int32_t* dist, int32_t* bearing) {
  float rads = (abs((float) * lat2) / 10000000.0) * 0.0174532925; // latitude to radians
  double scaleLongDown = cos (rads); // calculate longitude scaling
  float distLat = *lat2 - *lat1;     // difference of latitude in 1/10 000 000 degrees
  float distLon = (float)(*lon2 - *lon1) * scaleLongDown;
  *dist = sqrt(sq(distLat) + sq(distLon)) * 1.113195 / 100;  // distance between two GPS points in m
  *bearing = (int) round ( 90 + atan2(-distLat, distLon) * 57.2957795);  //azimut, convert the output radians to deg
  if (*bearing < 0) *bearing += 360;
}
void display_tft_head() {
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setTextSize(3);    // 1 = 10 pixels, size 2 =20 pixels
  tft.setCursor(10, 10); tft.print("ALT:");
  tft.setCursor(10, 40); tft.print("SPEED: ");
  tft.setCursor(10, 70); tft.print("HEAD:");
  tft.setCursor(10, 100); tft.print("BAT:");
  tft.setCursor(10, 130); tft.print("DIST:");
  tft.drawFastHLine(5, 160,  tft.width(), TFT_WHITE);
  tft.setTextSize(2);
  tft.setCursor(10, 180); tft.print("SATELIT:");
  tft.setCursor(10, 200);tft.print("FMODE:");
  tft.setCursor(10, 220);tft.print("NAVMODE:");
  tft.setCursor(10, 240);tft.print("GPSMODE:");
  tft.setCursor(10, 260); tft.print("ARM-HFIX:");
  tft.setCursor(10, 280);tft.print("ROLL:");
  tft.setCursor(10, 300);tft.print("PITCH:");
  tft.setCursor(10, 320);tft.print("RSSI:");
  tft.setCursor(10, 340);tft.print("FTIME:");
  tft.setCursor(10, 360);tft.print("LAT:");
  tft.setCursor(10, 380);tft.print("LON:");
  tft.drawFastHLine(5, 400,  tft.width(), TFT_WHITE);
  }
void display_tft() {
  tft.setTextSize(3);
  tft.setTextColor(TFT_YELLOW, TFT_BLACK);
  tft.setCursor(120, 10); tft.print(uav_alt);
  tft.setCursor(120, 40); tft.print(uav_groundspeed);
  tft.setCursor(120, 70); tft.print(uav_heading);
  tft.setCursor(120, 100); tft.print(uav_bat/100);
  tft.setCursor(120, 130); tft.print(home_distance); tft.print("   "); tft.print(route_distance);
  tft.setTextSize(2);
  tft.setCursor(120, 180);tft.print(uav_satellites_visible);
  tft.setCursor(120, 200);tft.print(flightmode); if (uav_failsafe == 1) {tft.print(" - !!FS!!");}
  tft.setCursor(120, 220);tft.print(navmode);
  tft.setCursor(120, 240);tft.print(gpsmode); if (uav_gpsmode == 3) {tft.print(" - WP:");tft.print(uav_WPnumber);}
  tft.setCursor(120, 260);tft.print(arm);if (uav_homefixstatus == 1) {tft.print(" - HFIX");}
  tft.setCursor(120, 280);tft.print(uav_roll);
  tft.setCursor(120, 300);tft.print(uav_pitch);
  tft.setCursor(120, 320);tft.print(uav_rssi);
  tft.setCursor(120, 340);tft.print(flight_time);tft.print(" ");tft.print(LTM_pkt_ok);tft.print(" ");tft.print(row);;
  tft.setCursor(70, 360);tft.print(uav_lat);tft.print(" ");tft.print(uav_homelat);
  tft.setCursor(70, 380);tft.print(uav_lon);tft.print(" ");tft.print(uav_homelon);
  tft.setCursor(10, 420);tft.print(LTM_pkt);
  tft.setCursor(10, 440);tft.print(errMessage);
   }
void display_tft_clear(){
  tft.setTextSize(3);
  tft.setTextColor(TFT_BLACK, TFT_BLACK);
  tft.setCursor(120, 10); tft.print(uav_alt);
  tft.setCursor(120, 40); tft.print(uav_groundspeed);
  tft.setCursor(120, 70); tft.print(uav_heading);
  tft.setCursor(120, 100); tft.print(uav_bat/100);
  tft.setCursor(120, 130); tft.print(home_distance); tft.print("   "); tft.print(route_distance);
  tft.setTextSize(2);
```

```
  tft.setCursor(120, 180);tft.print(uav_satellites_visible);
  tft.setCursor(120, 200);tft.print(flightmode); if (uav_failsafe == 1) {tft.print(" - !!FS!!");}
  tft.setCursor(120, 220);tft.print(navmode);
  tft.setCursor(120, 240);tft.print(gpsmode); if (uav_gpsmode == 3) {tft.print(" - WP:");tft.print(uav_WPnumber);}
  tft.setCursor(120, 260);tft.print(arm);if (uav_homefixstatus == 1) {tft.print(" - HFIX");}
  tft.setCursor(120, 280);tft.print(uav_roll);
  tft.setCursor(120, 300);tft.print(uav_pitch);
  tft.setCursor(120, 320);tft.print(uav_rssi);
  tft.setCursor(120, 340);tft.print(flight_time);tft.print(" ");tft.print(LTM_pkt_ok);tft.print(" ");tft.print(row);;
  tft.setCursor(70, 360);tft.print(uav_lat);tft.print(" ");tft.print(uav_homelat);
  tft.setCursor(70, 380);tft.print(uav_lon);tft.print(" ");tft.print(uav_homelon);
  tft.setCursor(10, 420);tft.print(LTM_pkt);
  tft.setCursor(10, 440);tft.print(errMessage);
}
void vypocet(){
   if ((uav_satellites_visible >= gps_minsatfix) && (gps_fix == 0)) {
          gps_maxlat = uav_lat + gps_range;
          gps_minlat = uav_lat - gps_range;
          gps_maxlon = uav_lon + gps_range;
          gps_minlon = uav_lon - gps_range;
          gps_fix = 1;
      }
      if ((gps_fix) && (uav_lat > gps_minlat) && (uav_lat < gps_maxlat) && (uav_lon > gps_minlon) && (uav_lon <
gps_maxlon)) {
          gps_ok = 1;
          if (gps_lastlat == 0) { gps_lastlat = uav_lat; }
          if (gps_lastlon == 0) { gps_lastlon = uav_lon; }
      }
      else {
          gps_ok = 0;
      }
      if ((gps_ok) && (uav_homefixstatus) && (uav_groundspeed > min_speed))
      {
          GPS_dist(&gps_lastlat, &gps_lastlon, &uav_lat, &uav_lon, &ground_distance, &ground_course);
          GPS_dist(&uav_lat, &uav_lon, &uav_homelat, &uav_homelon, &home_distance, &home_heading);
          route_distance += ground_distance;
          gps_lastlat = uav_lat;
          gps_lastlon = uav_lon;
          if (flight_start = 0) {flight_start = (int) millis()/1000;}
          flight_time = (int) millis()/1000-flight_start;
      }
   }
void display_terminal() {
  //Serial.println("LoRa Receiver");
  String dataMessage1;
  String dataMessage2;
  String dataMessage3;
  String dataMessage4;
  String dataMessage;
  Serial.println(errMessage);
  dataMessage1 = String(uav_bat/100) + ";" + flightmode + ";" + gpsmode + ";" + navmode + ";" + String(uav_alt) + ";"
+ String(uav_groundspeed) + ";" + String(uav_roll) + ";" + String(uav_pitch) + ";" + String(uav_heading) + ";" +
String(home_distance) + ";" + String(home_heading);
  dataMessage2 = String(uav_lat) + ";" + String(uav_lon) + ";" + String(uav_homelat) + ";" + String(uav_homelon) +  ";"
+ + ";" + String(home_heading) + ";" + String(home_distance);
  dataMessage3 = String(uav_amp) + ";" + String(uav_rssi) + ";" + String(uav_HDOP) + ";" +
String(uav_satellites_visible) + ";" + String(uav_fix_type) + ";" + String(uav_homefixstatus) + ";" + String(gps_ok) + ";"
+ String(uav_arm) + ";" + String(uav_failsafe);
  dataMessage3 = String(ltm_naverror) + ";" + String(uav_HWstatus) + ";" + String(uav_WPnumber) + ";" +
String(LTM_pkt_ok) + ";" + String(LTM_pkt_ko);
  dataMessage = dataMessage1 + ";" + dataMessage2 + ";" + dataMessage3 + ";" + dataMessage4;
  Serial.println(dataMessage);
}
```

```
void setup() {
  tft.init();
  tft.setRotation(2);
  tft.fillScreen(TFT_BLACK);
  display_tft_head();
  tft.setTextColor(TFT_RED, TFT_BLACK);
  tft.setTextSize(2);tft.setCursor(10, 460); tft.print(errMessage);
  Serial1.begin(9600);
  delay(500);
  SD.begin(SD_CS);
  delay(500);
  if(!SD.begin(SD_CS)) {
   errMessage = "Card Mount Failed!";
    tft.setCursor(10, 460);tft.print(errMessage);
   return;
  }
  if (!SD.begin(SD_CS)) {
   errMessage ="Card Init Failed!";
    tft.setCursor(10, 460);tft.print(errMessage);
   return;
   }
  while (existFile) {
   nameFile = prefixnameFile + String(numberFile) + ".dat";
   if (SD.exists(nameFile.c_str())) {
    numberFile++;
    } else {
     existFile = 0;
    }
  }
  dataFile = SD.open(nameFile.c_str(), FILE_WRITE);
  if (dataFile) {
  //dataFile.println(dataHeader);
    dataFile.seek(dataFile.size());
    errMessage = "Created " + nameFile ;
   } else {
    errMessage = "Created File Failed!";
    tft.setCursor(10, 460);tft.print(errMessage);
  }
 }

void loop() {
  row++;
  LTM_lastpkt_ok=LTM_pkt_ok;
  ltm_read();
  if ((LTM_pkt_ok - LTM_lastpkt_ok) > 0 ){LTM_pkt="LTM-RECEIVE-OK";vypocet();}
  else {LTM_pkt="!! LTM-NORECEIVE !!";}
  // display_terminal();
  if (uav_alt < max_alt && uav_groundspeed < max_speed) {display_tft();}
  /*
  if (dataFile) {
   dataFile.println(dataMessage);
   dataFile.flush();
   errMessage = "Writing " + nameFile;
  }
  */
  dataFile.flush();
  delay(500);
  display_tft_clear();
 }
```