

---

GEOSCAN

# GEOSCAN PIONEER

Operating manual





<b>1</b>	<b>Return to Pioneer webpage</b>	<b>1</b>
1.1	Pioneer assembly	2
1.1.1	Frame assembly	4
1.1.2	Protection assembly	8
1.1.3	First switch-on	10
1.2	Settings	12
1.2.1	Firmware update	12
1.2.2	RC transmitter setting	12
1.2.3	Receiver connection	13
1.2.4	Autopilot parameters setting	14
1.2.5	Autopilot logs	15
1.3	Flight	17
1.3.1	Simulator	18
1.3.2	LiPo battery	20
1.3.3	Flight safety	21
1.3.4	RC transmitter	21
1.3.5	Controls	23
1.3.6	Flight preparation - first start	24
1.4	Extension modules	25
1.4.1	Extension modules board	26
1.4.2	Cargo module	28
1.4.3	GPS/GLONASS module	30
1.4.4	OpenMV camera	33
1.4.5	Optical flow module	36
1.4.6	LED module	39
1.4.7	GoPro camera gimbal	42
1.4.8	Photo and video camera	43
1.4.9	FPV kit	43
1.4.10	Onboard indoor navigation module	44
1.5	Programming	45
1.5.1	Visual programming TRIK	45
1.5.2	Pioneer Station	49
1.5.3	Lua programming language	54
1.6	Geoscan Locus indoor navigation system	62
1.6.1	Flyzone setting	63
1.6.2	Indoor flight	65
1.6.3	Navigation system firmware update	65
1.7	FAQ	66
1.8	Changelog	66
1.8.1	20/06/2019	66

1.8.2	11/02/2019 . . . . .	67
1.8.3	1/02/2019 . . . . .	67

# CHAPTER 1

---

[Return to Pioneer webpage](#)

---

Geoscan Pioneer is a multifunctional educational complex. It was developed for schools, robotics labs and for individual use.



Use Pioneer to:

- study the robotics and avionics basics
- get to know the principles of quadcopter construction

- 
- master your flying skills
  - tailor the drone to suit your needs and tasks
  - program the drone

---

**Important:** Pioneer’s functional abilities can be extended by adding and programming extension modules.

---

Table 1: Technical specs

Parameter	Value
Class	quadcopter
Flight time	up to 17 minutes
Airspeed	up to 65 km/h
Weight	230 g
Max takeoff weight	230 g
Dimensions	290 x 290 x 120 mm
Motors	Brushless 1306 3100KV
Battery	LiPo 2S 1300 mAh 9.62 Wh
Max flight distance	500 m
Max wind speed	up to 5 m/s
Operation temperature	From 0 to +40 °C
























## 1.1 Pioneer assembly

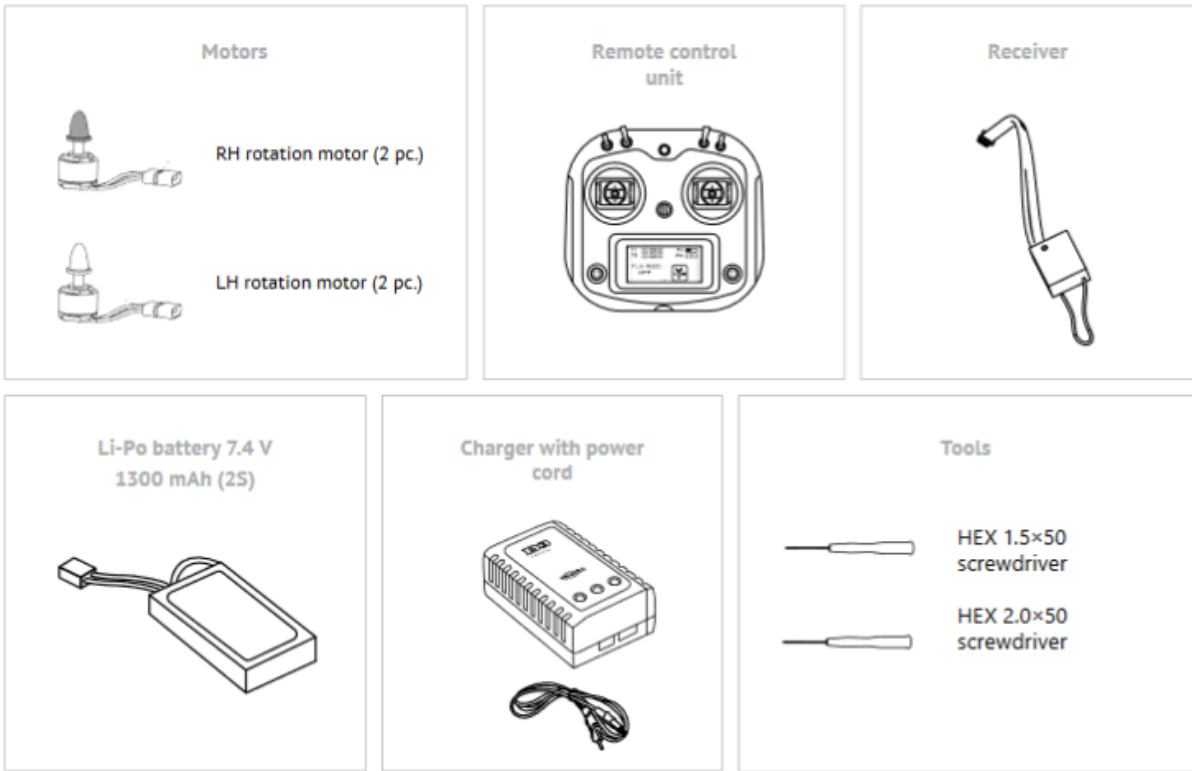


---

**Important:** The stock Pioneer kit contains all necessary parts and tools to assemble and launch it.

---

<p style="text-align: center;"><b>Propellers</b></p>  <p style="margin-left: 100px;">RH rotation propeller (4 pc.)</p>  <p style="margin-left: 100px;">LH rotation propeller (4 pc.)</p>	<p style="text-align: center;"><b>Main board</b></p> 
<p style="text-align: center;"><b>Protection</b></p>  <p style="margin-left: 100px;">Protection arch (8 pc.)</p>  <p style="margin-left: 100px;">Protection bar (8 pc.)</p>  <p style="margin-left: 100px;">Protection base (4 pc.)</p>	<p style="text-align: center;"><b>USB cable</b></p> 
<p style="text-align: center;"><b>Frame</b></p>  <p style="margin-left: 100px;">Battery compartment back plate</p>  <p style="margin-left: 100px;">Frame base</p>  <p style="margin-left: 100px;">Battery compartment cover</p>  <p style="margin-left: 100px;">Landing gear (upper part)</p>  <p style="margin-left: 100px;">Landing gear (lower part)</p>	<p style="text-align: center;"><b>Frame fixings</b></p>  <p style="margin-left: 100px;">Studding nut (metal, short) (6 pc.)</p>  <p style="margin-left: 100px;">Studding nut (long) (6 pc.)</p>  <p style="margin-left: 100px;">Studding nut (short) (6 pc.)</p>  <p style="margin-left: 100px;">Screw M2x4 (12 pc.)</p>  <p style="margin-left: 100px;">Screw M3x5 (6 pc.)</p>  <p style="margin-left: 100px;">Screw M3x10 (12 pc.)</p>  <p style="margin-left: 100px;">Screw M3x4 (6 pc.)</p>  <p style="margin-left: 100px;">Damper (4 pc.)</p> <p style="text-align: center;"><b>Protection fixings</b></p>  <p style="margin-left: 100px;">Studding nut (long) (12 pc.)</p>  <p style="margin-left: 100px;">Screw M2x6 (12 pc.)</p>  <p style="margin-left: 100px;">Screw M3x10 (24 pc.)</p>



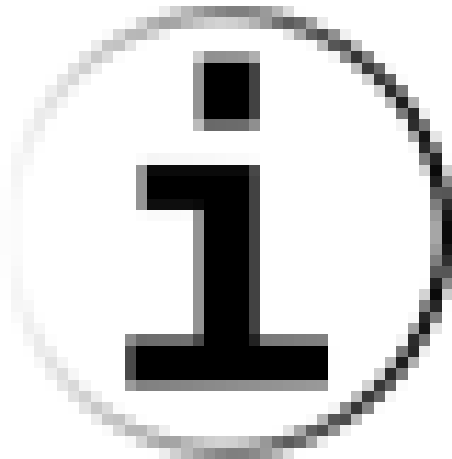
Pioneer assembly is executed in two stages:

Frame assembly

Protection assembly

Modules connection is described in [Extension modules](#)

**1.1.1 Frame assembly**



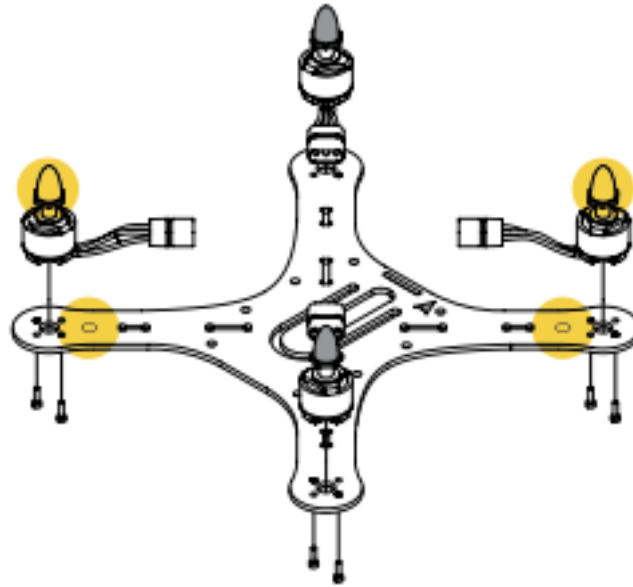
Pioneer kit contains all necessary elements and tools

**Step 1**



---

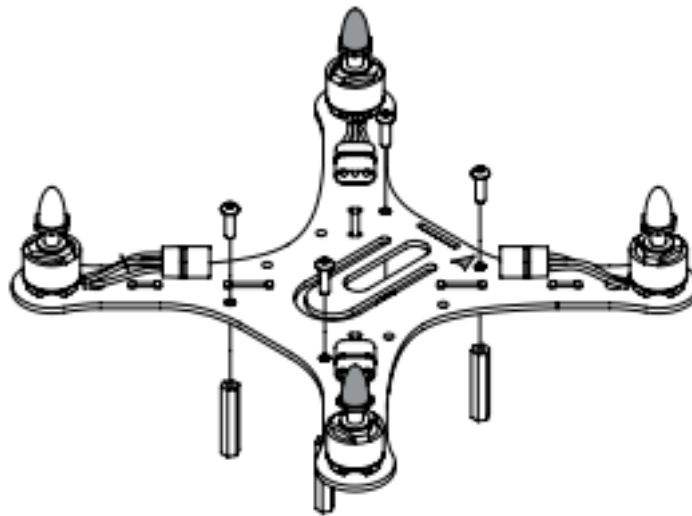
Parts: frame base, clockwise motor - 2, counterclockwise motor - 2, M2x4 screw - 8.



The motors with silver caps should be installed according to white marks on the frame. Place and secure each motor with 4 screws.

### Step 2

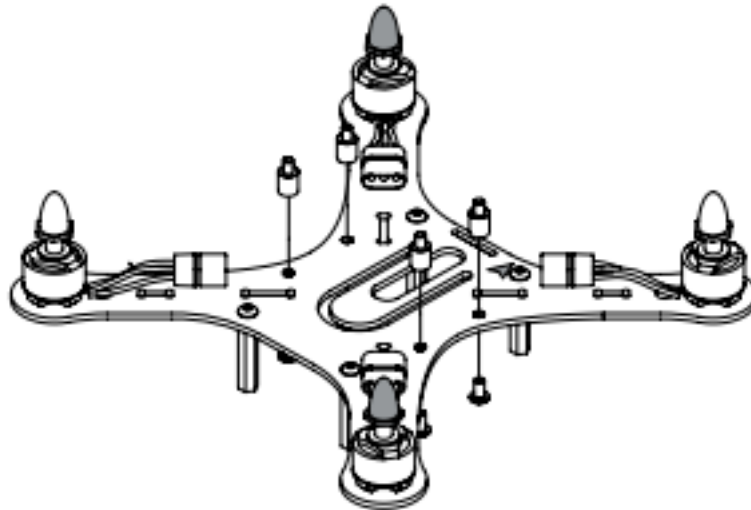
Parts: unit assembled at step 1, studding nut (long) - 4, m3x10 screw - 4.



Install the stands on the bottom side of the frame and secure them with M3x10 screws like shown in the picture.

### Step 3

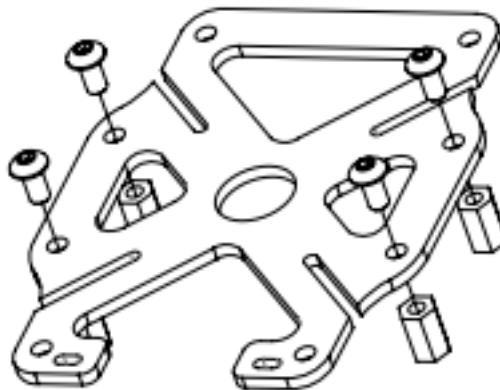
Parts: unit assembled at step 2, dampers - 4, M3x4 screws - 4.



Mount 4 dampers on the board and secure them with M3x4 screws as shown on the picture.

#### Step 4

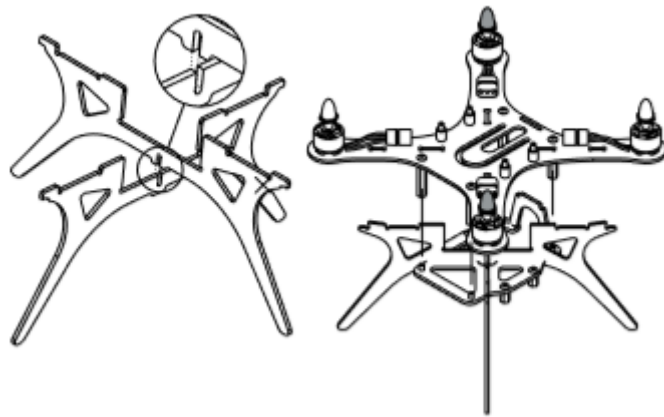
Parts: battery compartment cover, 8 mm stubbing nut - 4 M3x5 asrews - 4.



Using M3x5 screws thighten the stands on the battery bay cover like shown on the picture. These stands will be required for mounting extra modules.

#### Step 5

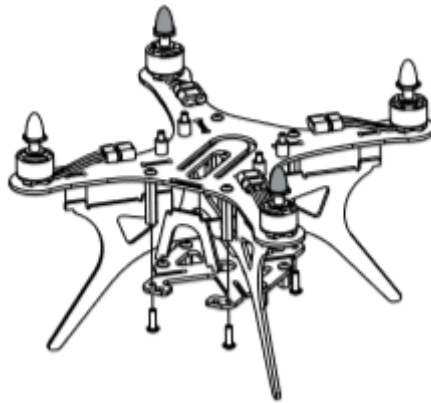
Parts: unit assembled at step 3, landing gear (upper and lower), battery compartment backplate.



Insert the LiPo bay backplate and landing gear in their slots at the bottom of the frame.

### Step 6

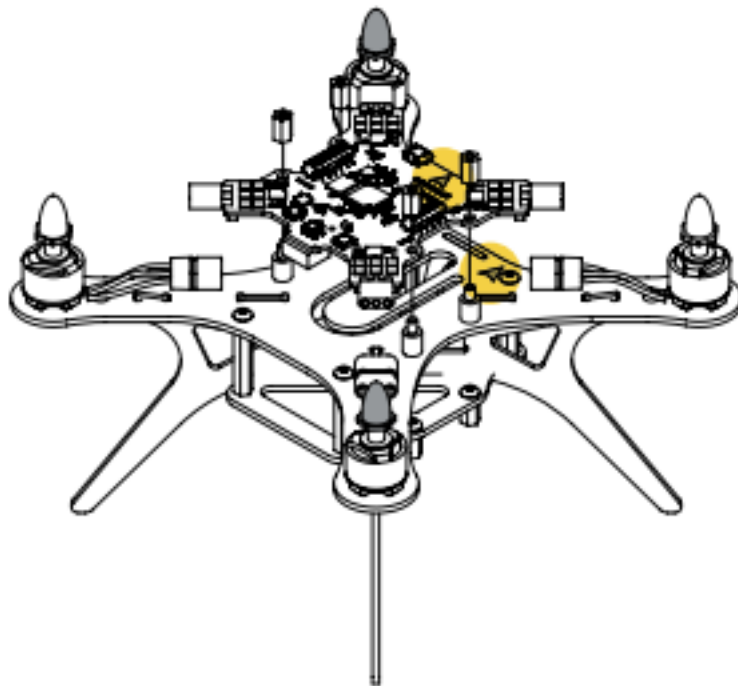
Parts: unit assembled at step 4, unit assembled at step 5, M3x10 screws.



Using M3x10 screws, connect the assembled frame and lipo bay cover together (steps 4 and 5)

### Step 7

Parts: unit assembled at step 6, studding nut(metal, short) - 4, main board.



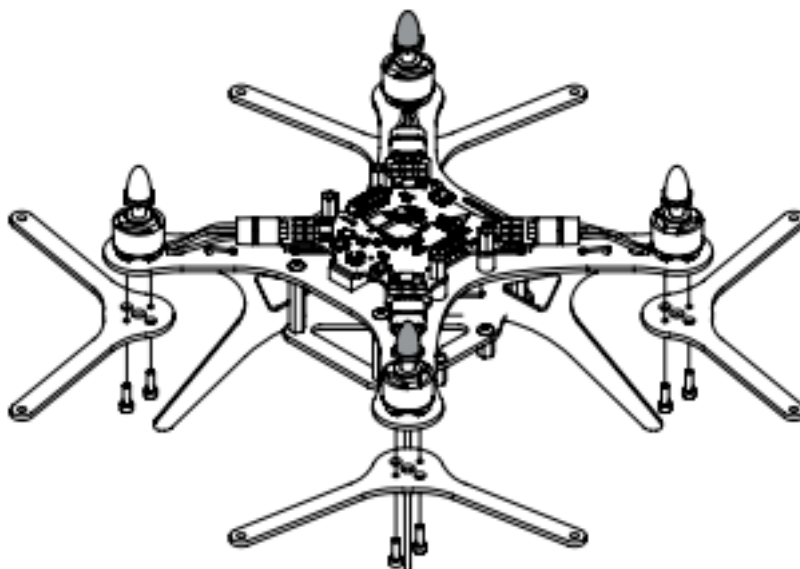
Using dampers, mount the board on the frame so that white arrows on top of them point in same direction. Plug in motors connectors applying no extra force.

Once finished with the frame, proceed to the [protection assembly](#)

## 1.1.2 Protection assembly

### Step 8

Parts: unit assembled at step 7, protection base - 4, M2x6 screw - 8.



Place protection mounts and secure them using M2x6 screws like shown on the picture

### Step 9

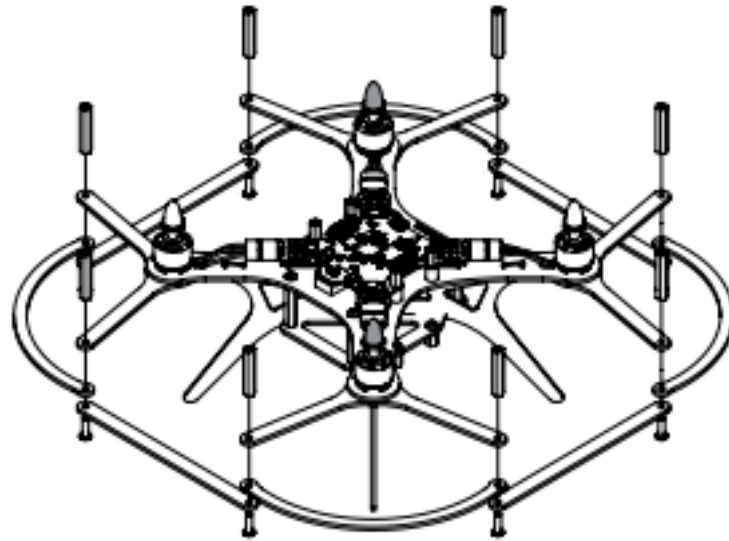
Parts: unit assembled at step 8, protection arch - 8, protection bar - 8, M3x10 screw - 8,  studding nut (long) - 8.

(continues on next page)

---

(continued from previous page)

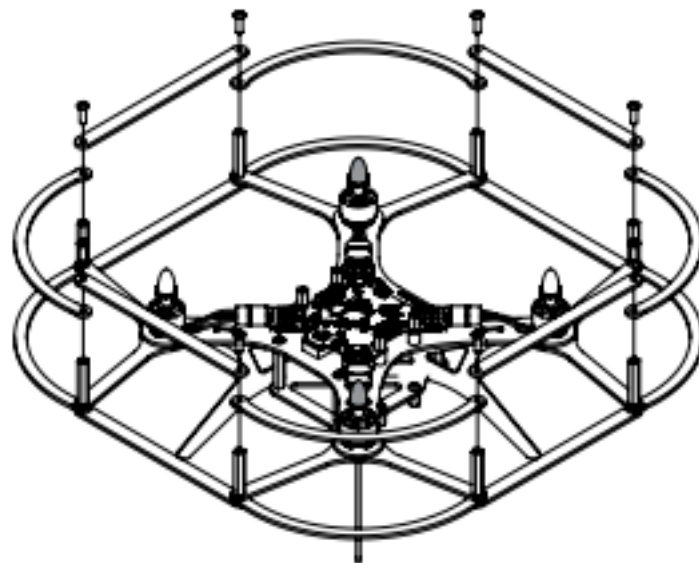
---



Secure the protection arcs, bulkhead and stands using M3x10 screws like shown in the picture.

**Step 10**

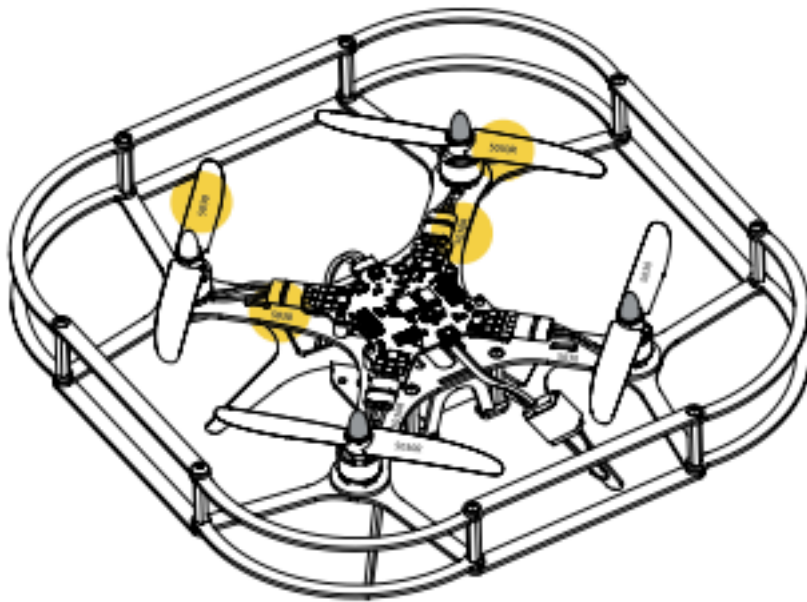
Parts: unit assembled at step 9, protection arch - 8, protection bar - 8, M3x10 screw - 10.



Connect the stands and protection arcs using M3x10 screws.

**Step 11**

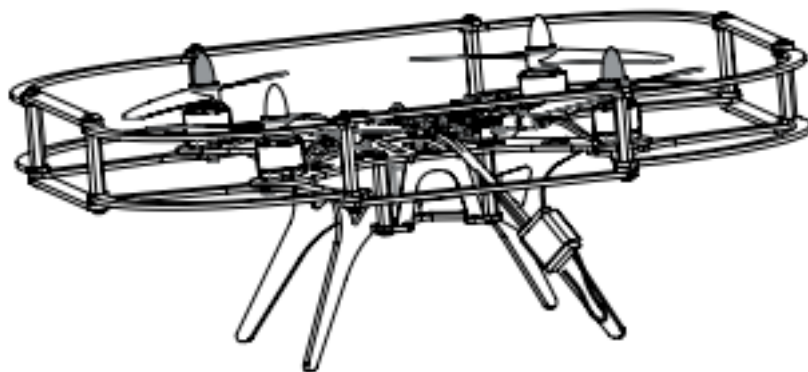
Parts: unit assembled at step 10, 5030 prop -2, 5030R prop - 2.



Note: black caps have right thread! The markers on props and board should match. Take off the caps, mount the props and twist them back tight.

### Step 12

Parts: unit assembled at step 11, receiver (packed with RC transmitter).

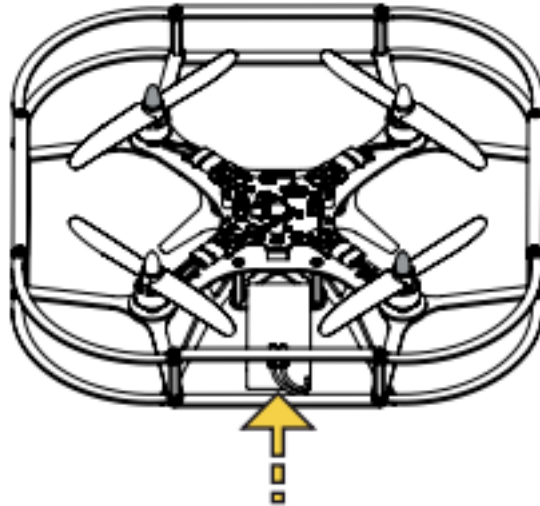


Install the receiver on the landing gear using elastic ring. Connect the PPM cable to the board under mUSB port.

For further instructions, proceed to [First switch-on](#)

### 1.1.3 First switch-on

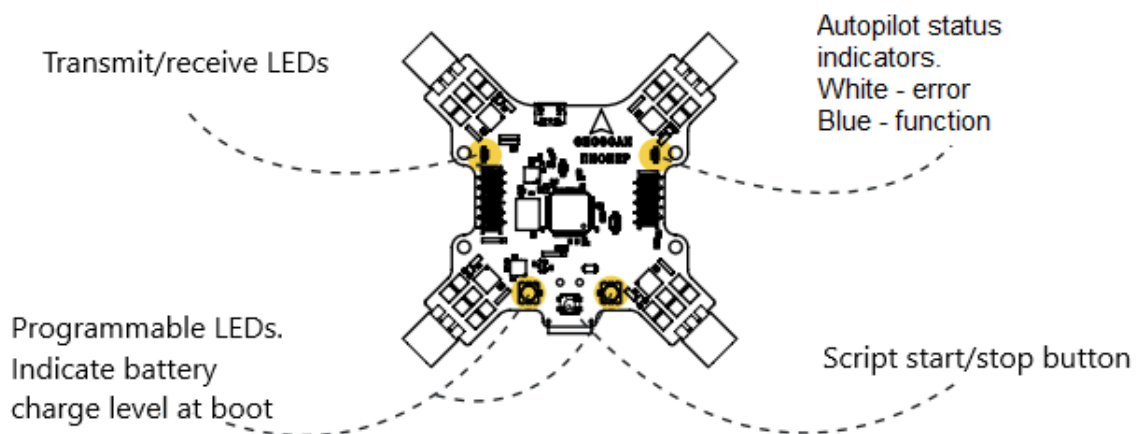
To turn Pioneer on, place the LiPo battery in the bay and plug in its connector to the board.



---

**Important:**

- When connected, the quadcopter makes a beeping sound and blinks with its leds to show the remaining charge level. Red lights - LiPo discharged, green - fully charged.
  - White led blinking fast - autopilot loading.
  - Blue led blinking slowly - the quadcopter is ready to fly.
  - Blue led blinking fast - the script is initialised
  - White led blinking fast - error
  - When the battery voltage drops down - quadcopter begins to beep every 5 seconds.
- 



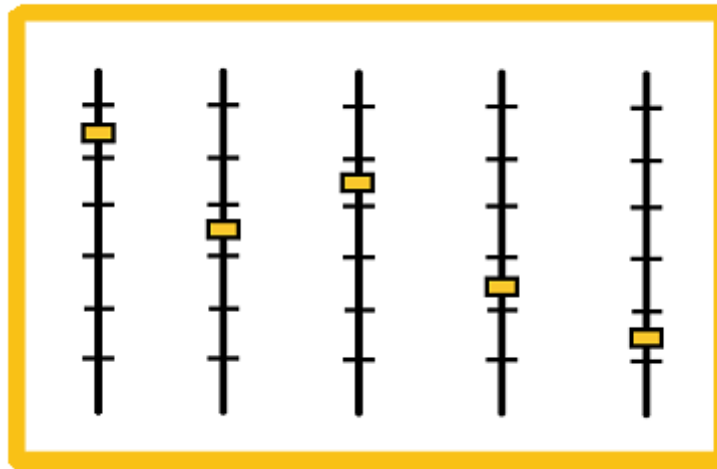
**Attention:**

**Pioneer is able to switch off its motors in case of:**

- pitch or roll angle more than 90°;
- strike or fall.

---

## 1.2 Settings



Pioneer's firmware can be upgraded. This section has necessary info on the update process.

If your RC transmitter or receiver were changed, proceed to [RC connection](#)

You can [set up the transmitter](#) in case of malfunction.

[Autopilot parameters](#) instruction will help you set parameters correctly.

### 1.2.1 Firmware update

For Pioneer to perform correctly, it is recommended to upload the last version of firmware. You need to have Pioneer Station installed on your PC. If you don't have it, click [download Pioneer Station](#) and run the acquired .exe file.

---

**Note:** If program doesn't run, try to [download 32bit version](#), unpack the archive and open run.bat file.

---

Turn on firmware update mode on Pioneer. Press "Start" button on it and connect the USB cable while holding the button pressed. All LEDs on the board should switch on.

Run Pioneer Station and click the quadcopter icon in the top left corner of the window. Select "firmware update" and follow the appearing instructions.

---

**Note:** If quadcopter is not shown in the window, try rebooting your PC and re-connect Pioneer. If this didn't help, [download and install port driver](#), then reload Pioneer Station and re-connect the drone.

---

It is recommended to select "inbuilt" source of firmware. That guarantees you install the latest version.

Wait for the firmware update to finish, this can take a few minutes. Click OK button afterwards to switch Pioneer in standard connection mode.

If any problem occurs during Pioneer update or usage, feel free to contact our helpdesk via [support@geoscan.aero](mailto:support@geoscan.aero)

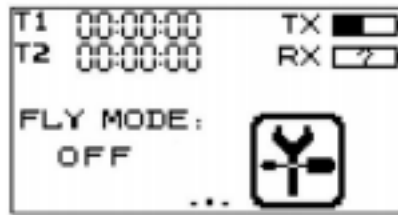
### 1.2.2 RC transmitter setting

To set up the controller, follow these steps:

1. Plug in the batteries in the battery compartment on the back of controller. To turn it on, press and hold two power buttons.



2. The default screen image looks like this:



“warning! Place all switches in their up position” means you have to check all flipswitches and place left stick to its lower position.

3. Use touchscreen to navigate the menu and press settings icon.



4. “Function” tab contains settings of the RC transmitter. “System” tab is used to change the model’s properties.

**Adjust the following settings in “system” tab:**

1. REVERSE → Ch2 и Ch4 –Rev
2. AUX. CHANNELS → Channel 5 → →select SWx switch type→ press SwA and select SwC in the opened menu.
3. AUX. CHANNELS → Channel 6 → → select switch type SWx→ press SwA and select SwD.
4. AUX. CHANNELS → Channel 7 → → select switch type SWx→ press SwA and select SwB.

**Select the SYSTEM tab:**

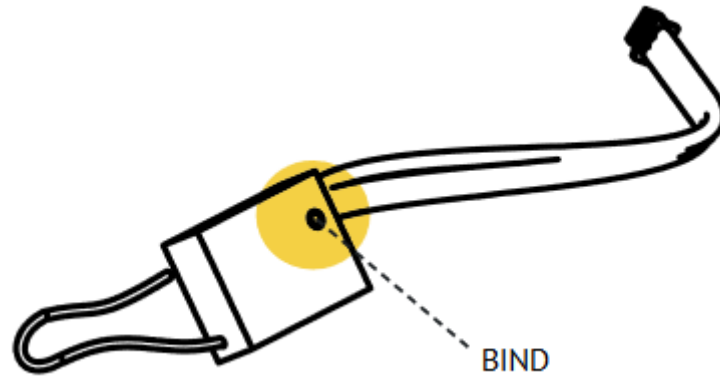
5. OUTPUT MODE → Output → PPM
6. STICKS MODE → M2 (Mode 2)

For further setting proceed to [Receiver connection](#)

### 1.2.3 Receiver connection

To control the drone with RC transmitter, a binding operation is required. Follow this manual to perform it.

**Attention:** Controlling several drones using one RC transmitter might cause crashes. Do not bind more than one receiver to one controller.



1. Turn the RC transmitter on
2. Use touchscreen to navigate the settings menu.
3. Select Rx Bind in “System” tab. “Binding to Rx..” should appear on the screen.
4. Press and hold “BIND” button on the receiver.
5. While holding the button, plug in the battery to turn quadcopter on.
6. Leave the Rx bind mode on the controller. The drone should beep once after this. A light on the receiver should not blink.

If the drone doesn't react on controller signals, try to re-bind its receiver once again. If this doesn't solve the problem, contact Geoscan helpdesk: [support@geoscan.aero](mailto:support@geoscan.aero)

---

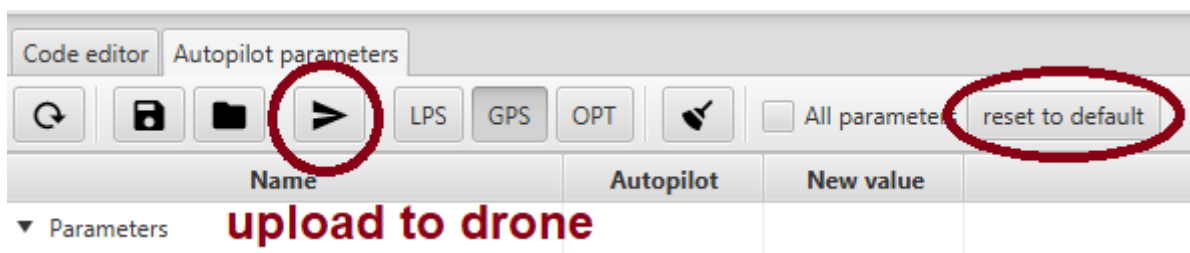
**Important:** Binding procedure works for FlySky-A8S receiver.

---

### 1.2.4 Autopilot parameters setting

Autopilot parameters can be set precisely using [Pioneer Station](#). Follow the [manual](#) to connect Pioneer to your PC. Switch to “autopilot parameters setting” tab.

To perform a setting, Pioneer should be connected to PC and its current parameters should be displayed in right part of Pioneer station window.



Switch to “autopilot parameters” tab. Left column represents parameters, their current values are displayed in the second column. To change any of them, click to the right of selected parameter and type the new value in, then press Enter. Don't forget to save the changes by clicking **save changes to autopilot**

**LPS**, **GPS** and **OPT** buttons are situated in the top part of the window. Each is responsible for loading a standard set of parameters for indoor positioning system, GPS and optical flow mode respectively.

---

**Note:** If you want to return to factory preset, click **return to default** button above the table and wait until Pioneer reboots.

---

---

## Parameters setting for motors change

Changing stock motors on Pioneer to more powerful (or any other at all) may result in different electrical specs. This, in turn, will lead to pre-flight checks fail, and Pioneer will not be able to take off. Adjust the following parameters to reset pre-flight checks:

- **Copter\_motorCheckTime** - duration (in seconds) of motors rpm check before takeoff. Set to zero to turn it off.
- **Copter\_startRpmMax** - Max rpm for pre-flight check pass.
- **Copter\_startRpmMin** - Min rpm for pre-flight check pass.
- **Copter\_startRpmSigma** - Max allowed inconsistency between motors rpm speeds.

There is another check, run for the first 5 seconds after takeoff command:

- **Copter\_stallRpm** - max rpm after which sync fail is declared and motors stop. For custom motors, use its max possible rpm value (calculated as motor's *kv* multiplied by *battery voltage*)

You may also need to customize PID parameters for different engines. If done incorrectly, this may result in noticeable vibrations and oscillations during flight.

To customize PIDs, adjust the following parameters:

- **Copter\_xyRate\_ki** - controller's integral component. Lower it if the controls are sluggish. Increase it if low-frequency oscillations are present.
- **Copter\_xyRate\_kp** - controller's proportional component. Lower it if the drone wobbles when hovering. Increase it if the controls feel sluggish.

---

**Note:** To evaluate necessary rpm values, look through **rpm** data chart in **motor-x** autopilot logs channel after manual-controlled flight. See [Autopilot logs](#) section.

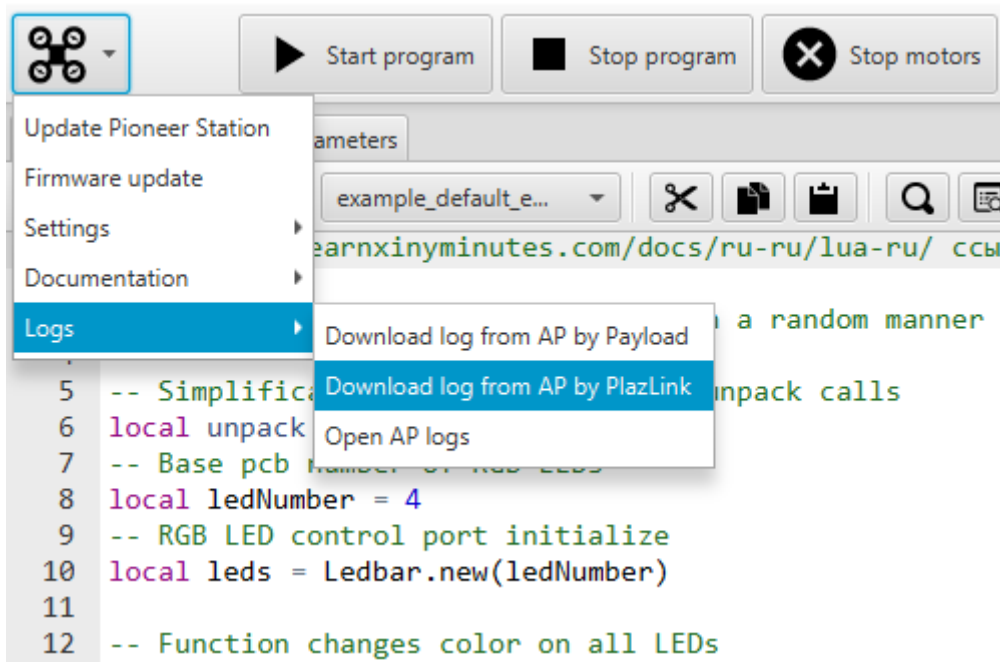
---

### 1.2.5 Autopilot logs

From the moment of motors start, Pioneer's autopilot begins to log flight parameters in multiple channels and saves them in downloadable file. Log writing stops when motors are turned off. In case they are started again, previous log is erased to start a new one.

#### Log download

To download the log file, connect Pioneer to the PC via USB and run Pioneer Station. Setup the connection and open **Logs - Download logs from AP by PlazLink** menu tab.

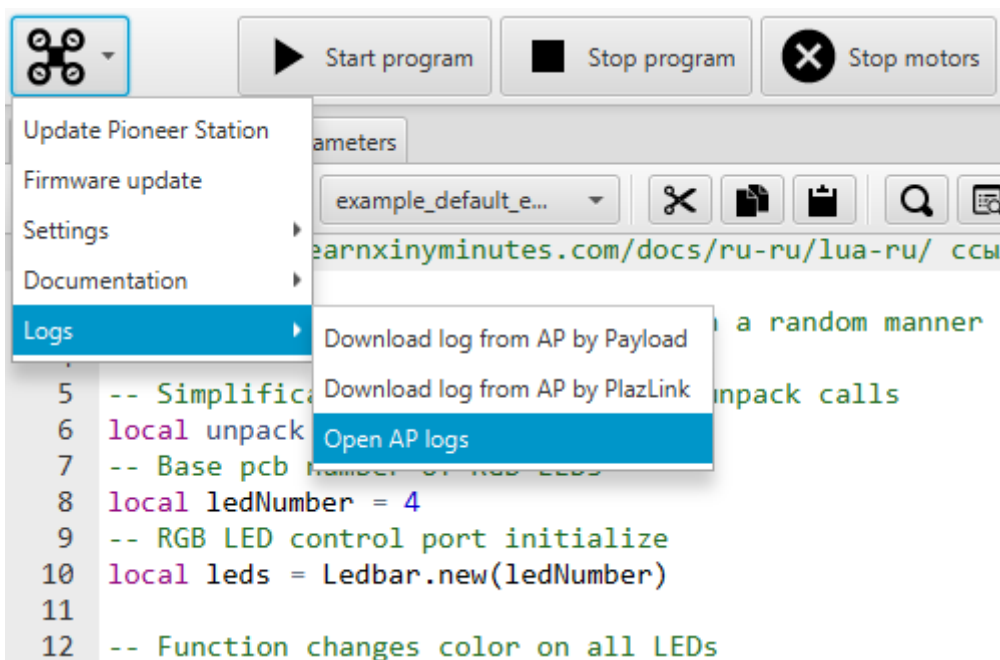


After that, select saving folder and file name. Log download will start automatically. Pioneer will reboot when it's finished.

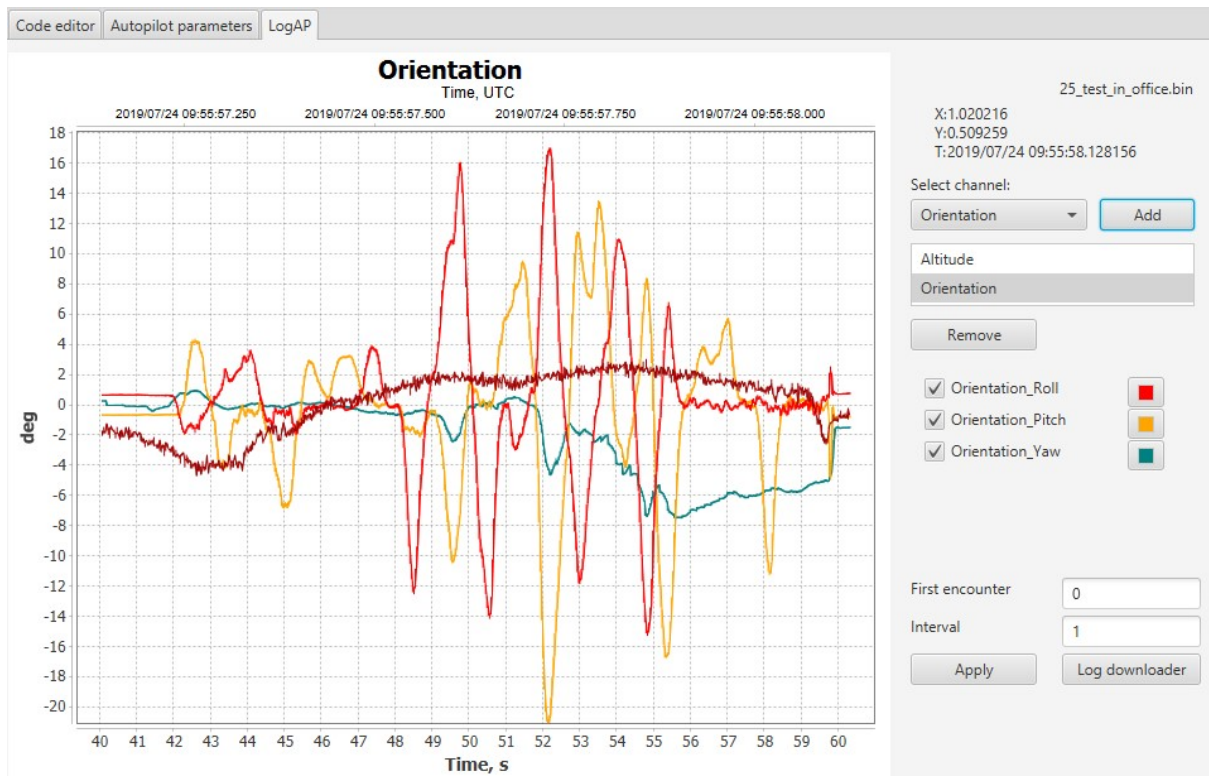
**Note:** Older firmware versions do not support log download by Plazlink protocol. In that case, select **Logs - Download logs from AP by Payload** menu tab. Follow the wizard instructions (press and hold "Start" button on Pioneer board and connect it to the computer). After that select folder and start log download.

### Log preview

To open log file, select **Logs - Open AP log** menu tab.



You can inspect autopilot log data in the opened tab.



- Select the channel you need and **Add** it to the preview window.
- Customize data projection with checkboxes and plot color palette.
- Use right mouse button and mouse wheel for plot navigation and scale change.
- To change time scale (horizontal), press and hold **Ctrl** and use mouse wheel.
- To clear the preview window, select unnecessary channels and press **Remove** button. This will not erase the log, just remove plot image.

Channel	Description
Altitude	Copter altitude using baro sensor
ControlData	Autopilot control signals
IntPowerStatus	Battery voltage data
IntStaticPressure	Baro sensor data
LpsState	Ultrasonic position system data
MagHeading	Magnetometer data
Motor-6X	Motors speed controllers data
NavPosition	GPS position data
NavPrecision	GPS precision data
NavSatInfo	Amount of satellites and their status
NavVelocity	3-axis speed data using GPS
OpticalFlow	Optical flow data
Orientation	Copter orientation data
RawAccelGyroData	RAW accelerometer and gyroscope data and temperature
RemoteControlData	Control signals from RC unit

## 1.3 Flight

This chapter will help you master the quadcopter controls and perform your first flight. It is strongly recommended to read it and learn the safety rules before takeoff.

### 1.3.1 Simulator

If you don't have proper drone control skills or you want to improve it, it is highly recommended for you to use simulator. It allows you to master the controls without risking the drone.

FlySky F6S RC unit can be connected to PC via USB cable to start your training in any simulator.

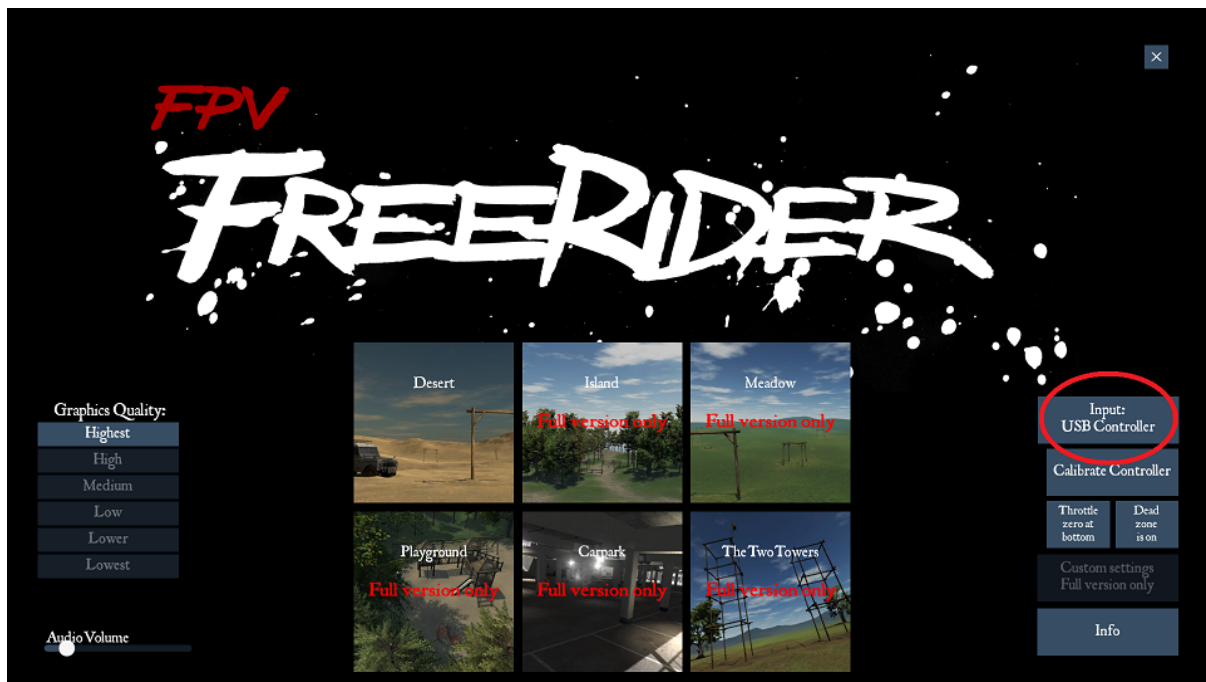
To get familiar with buttons and sticks, read [RC Unit](#)

Pioneer reaction on sticks movement is described in [Controls](#)

Most flight simulators may be quite expensive, but to learn the basics a free or demo version is enough.

We suggest trying FPVFreerider for that purpose. A free demo can be downloaded [here](#). Just scroll the page down and choose the version for your OS.

When installation is complete, plug the transmitter to your PC and turn it on. Then start FPVFreerider. All switches and sticks should be in upper position.



Choose “control using USB” and calibrate controller. Follow the guidance, moving sticks as shown on the screen and clicking OK. When finished, check if everything is done correctly, use sliders to trim for better accuracy. A virtual controller should perform exactly as the real one in your hands.

Then open a available map for flight training session.

It is usually recommended to start flying with third-person view (also called line of sight) in stabilise mode. Use the buttons to the top left corner of your screen to choose self-level, low rates and camera mode like on the screenshot below.



### Flight:

- Gently push the left stick forward. When the quadcopter lifts off the ground, try to hold it at the same altitude and land, using the throttle stick. Learn the moment when the drone starts to lift off.
- Holding the drone at the same altitude, push the right stick in any direction. Note that the drone not only accelerates but also loses altitude when tilted. To compensate it, add a little bit of throttle with the left stick. Learn to fly in different directions without losing altitude. You can click Reset button any time to start over.
- Pushing the left stick to the side controls the drones yaw direction. Changing course makes it harder to control as you have to take the shift into account. Try flying in square trajectory, turning the drone 90° in each corner. It will make it easier for you to learn if you do complex movements step by step.

Flying the drone is a skill that requires some practice. Mastering the flight in simulator you will feel your progress. When you can fly in selected direction without losing or crashing the drone, you are prepared for actual flight using Pioneer.

---

**Note:** The simulator gives the ability to learn first person view (FPV) flying as well. To toggle it, switch the camera button in FPV mode.

---

---

### 1.3.2 LiPo battery



As a power source, a 1300 mAh LiPo battery is used. This type of battery is lightweight and powerful, but may be fire hazardous if used incorrectly. In this section you will learn to use LiPo battery

#### Charging

1. Choose a incombustible surface to place a charging set. This should be away from any flammable objects (like carpets or curtains)
2. Make sure the battery doesn't have any external damages, bumps or shell fractures.
3. Plug in the charging station to 220v grid
4. Plug the balance connector to the charging station
5. The charge indicators should colour red. The green light indicates the battery is charged
6. Wait for LiPo to complete charging and unplug it from charging station. Do not leave the charging battery unattended!

---

**Note:** For long-term storage it is recommended to discharge it 50% and check every 3-4 months to prevent a deep discharge lower than 10%. Part-charge if necessary

---

**Attention:** Charging or using a damaged LiPo is prohibited. It must be stored separately away from flammable objects until the utilization moment

#### Operation

- When placed in the frame, the LiPo wires should be kept away from the props plane
- Plug in the battery before take-off and disconnect it right after the landing
- Don't allow a short circuit between the LiPo contacts.
- Check the battery for damages, punctures or bumps before and after flight. Heating it above 50°C and swelling is unacceptable.

**Danger:** If the battery is severely damaged, do not try to extract it immediately ! The bend and additional air inflow may cause intensive flame and lead to burn. If you notice a flare or smoke from the drone, land on a non-flammable surface immediately.



---

**Do not try to extinguish the burning LiPo with water!** Use dense cloth, sand, soil, carbon dioxide fire extinguisher. If none is accessible, wait for the battery to burn out and utilise it.

LiPo combustion products are extremely **toxic**, do not breathe in the smoke.

### 1.3.3 Flight safety

Pioneer is designed for both indoor and outdoor usage. The temperature should be from 0 to +40 °C and humidity no more than 80%. Quadcopter is not intended to fly in the rainy, snowy or windy conditions. Follow these rules for safe operation:

- Make sure there's no obstacles (tree branches, lamps) above the drone before take-off.
- Look out carefully for wires and cables as they can damage the drone in case of collision. If possible, do not perform flight near such objects.
- Do not fly the drone in gusting wind. A safe wind speed is no more than 5 m/s. Remember that the wind speed and direction may differ with altitude.
- Do not start programs with flight conditions with USB and LiPo connected to the drone. Remove the props in case ESC calibration is required.
- Check the props integrity before every flight. Using a bent or damaged propeller may cause a drone failure.

---

**Note:** During flight, propeller damage can be recognised by change in drone sound and performance. Do not continue the flight with damaged prop, if possible

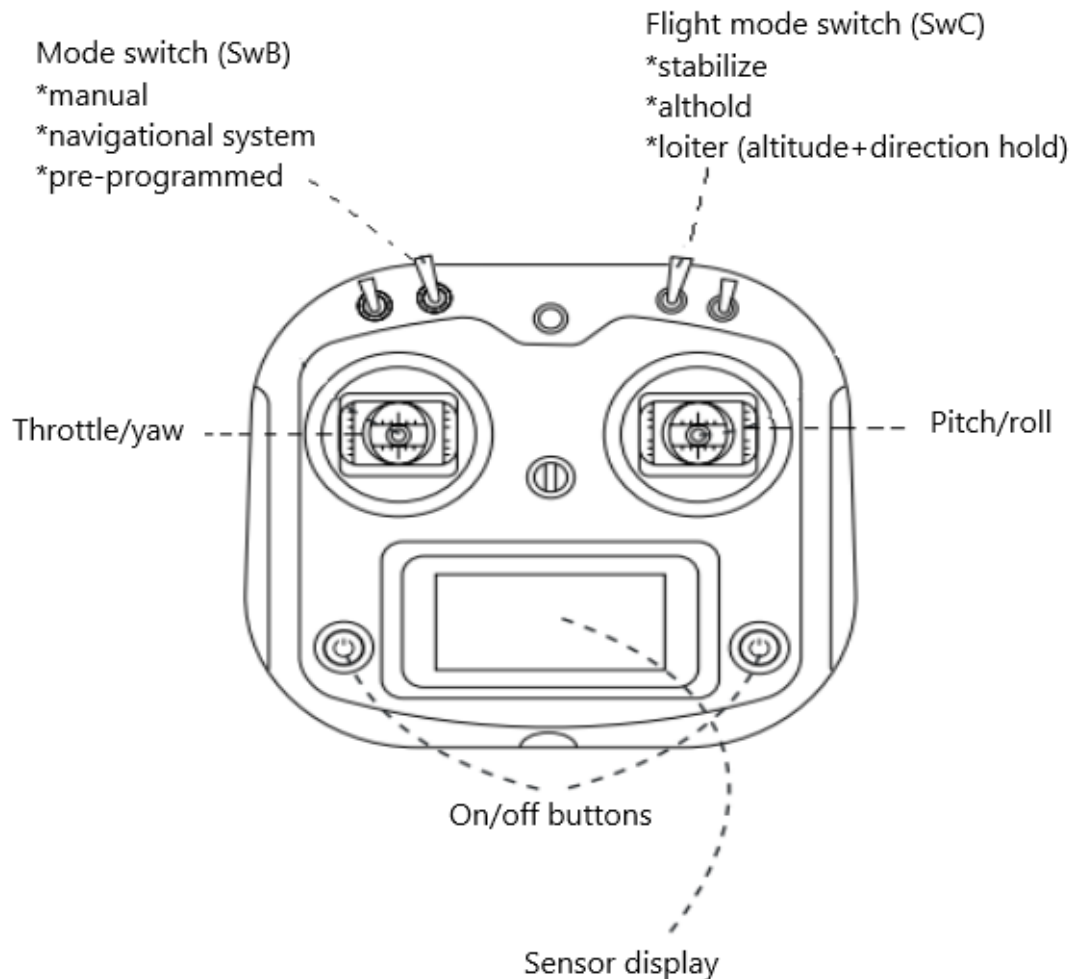
---

### 1.3.4 RC transmitter

This section describes the basics of Pioneer controls using FlySky i6S.



By default the controller is ready to use and doesn't require any additional setting. The drone is compatible with PPM protocol based controllers and uses a FlySky receiver. Use both sticks to control the drone. Use the mode switch to change the flight mode. Push and hold both power buttons to turn the controller on/off.



### Changing the controller mode

- Use SwB switch. The top position is for manual control mode. To set the flight mode, use SwC switch.
- SwB switch in middle position turns navigational system mode on. Quadcopter will use GPS or local positioning system.
- SWB switch in lower position activates a pre-programmed “mission” flight mode
- When performing a mission flight, the operator can take controls by witching in nav system or manual control using SwB switch.

### Flight mode set

- Use SwC switch. In top position - stabilize mode on. In this mode throttle stick changes the power output of motors.
- SWC switch in middle position activates althold mode. Throttle stick controls vertical velocity.
- The lower SwC switch activates headless mode. If you leave the throttle stick in middle position, the drone will hold current altitude. If the stick is moved up, the quadcopter will gain altitude
- If throttle stick is moved down, the drone will loose altitude. SW& switch in lower position activates loiter mode

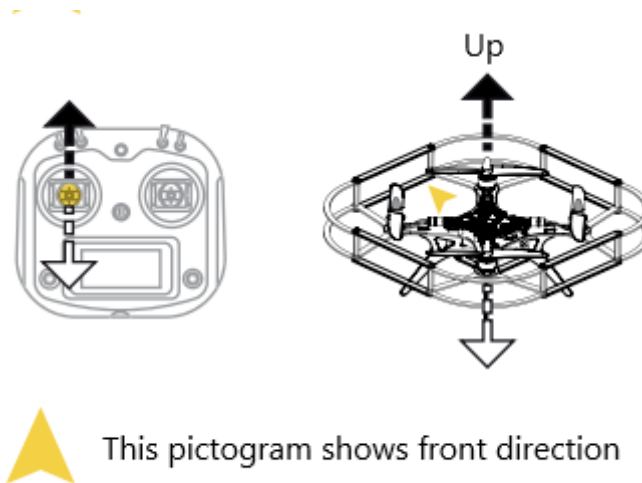
- Altitude control is the same as before, but quadcopter will remember yaw direction when armed
- You can spin the drone with yaw stick but it will maintain the primary orientation

**Attention:** Flight controller uses a barometric sensor to estimate the altitude. When the pressure is changed, the drone will react according to it, not actual altitude change.

### 1.3.5 Controls

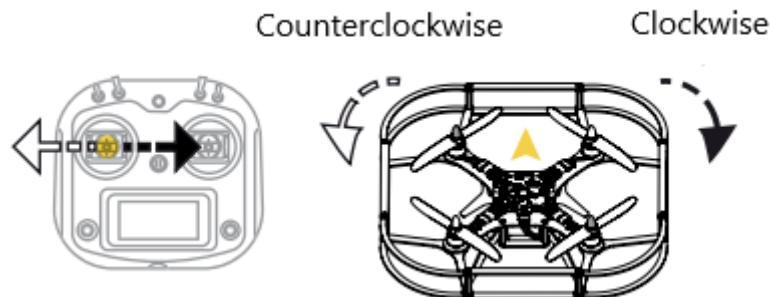
To control the Pioneer via RC transmitter, follow these rules:

#### Altitude control



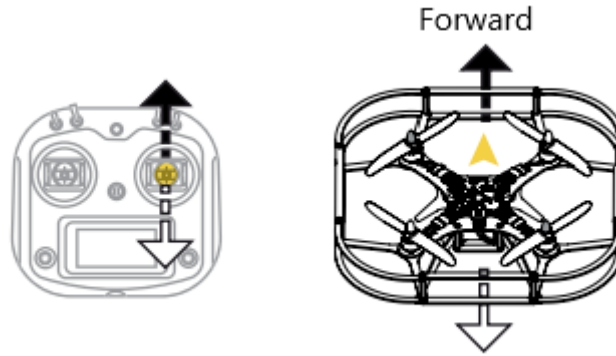
When throttle/yaw stick is pushed forward, the drone will gain altitude and decline when stick pulled down.

#### Direction change



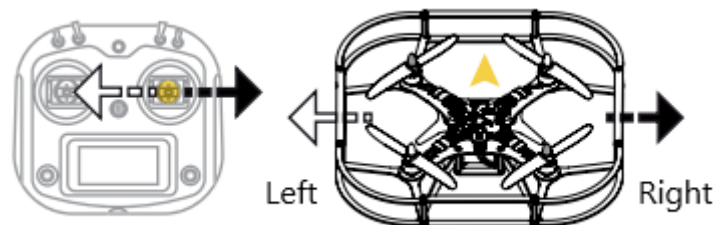
If throttle/yaw stick is pushed right or left, the drone will turn in the given direction.

#### Forward/reverse flight



Push the right stick up to tilt the drone and move forward. When stick is pulled down, the drone will tilt and move backwards.

### Moving left/right



When right stick is moved left or right, the drone tilts and moves in the same direction.

**Attention:** Operating the radio control, move the sticks smoothly. Hold the transmitter with both hands, thumbs on the sticks. Remember that while manoeuvring the drone may decline, and be ready to compensate this by giving a bit more throttle.

To perform your first flight go to [Flight preparation](#)

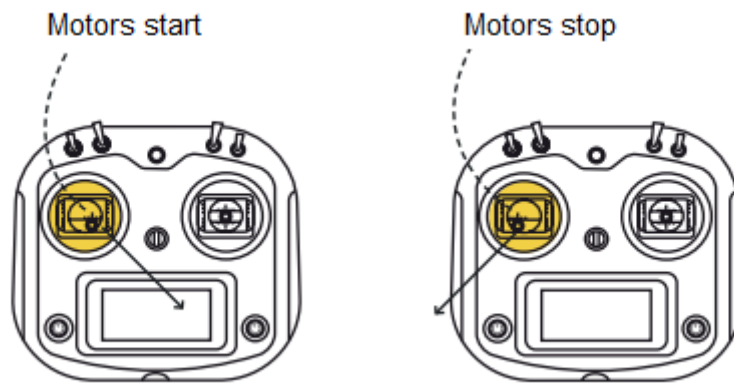
## 1.3.6 Flight preparation - first start

Read [Controls](#) before the flight

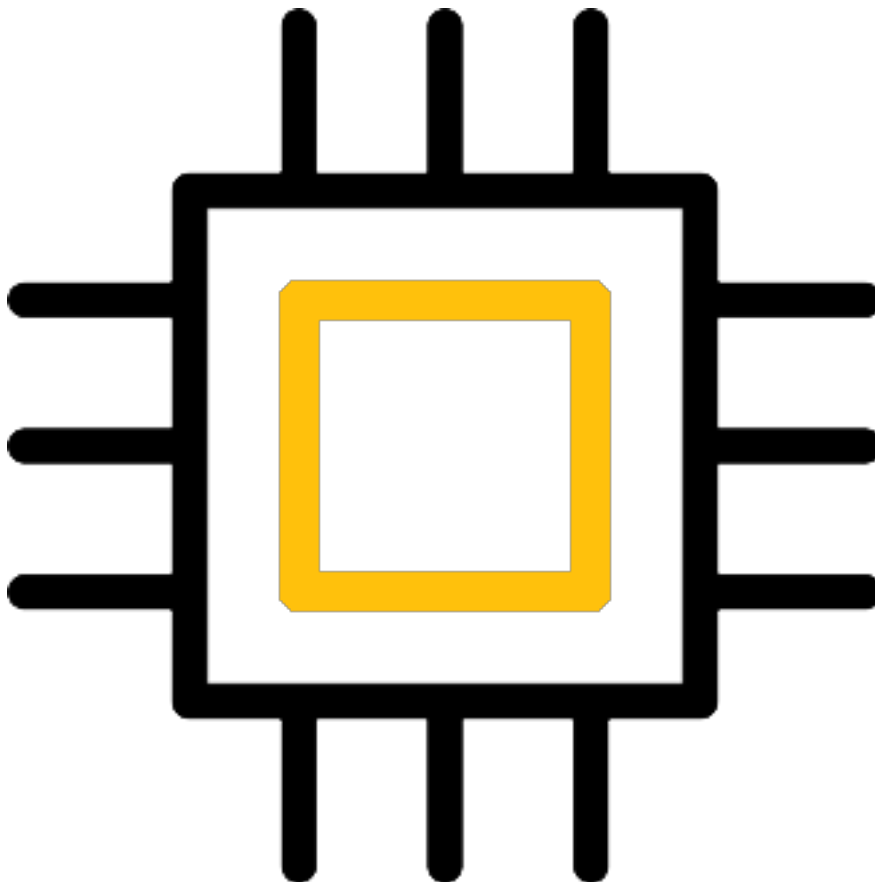
**A proper quadcopter start requires the following action sequence:**

1. Fully charge the LiPo battery and check the transmitter battery charge level.
2. Check if quadcopter has no mechanical damage.
3. Place throttle/yaw stick in the lower position.
4. Turn on the transmitter.
5. Place the LiPo battery in the battery bay underneath the quadcopter. Make sure it is secured tightly.
6. Plug in the battery connector to the board.
7. A beep should be heard and the LEDs light up.
8. Wait until the status LED starts blinking blue.
9. Turn on the motors. To do this, place the left stick in the lower-right position.
10. Wait until all props start spinning and take off smoothly. Stick to safety rules during flight, don't let the drone crash.

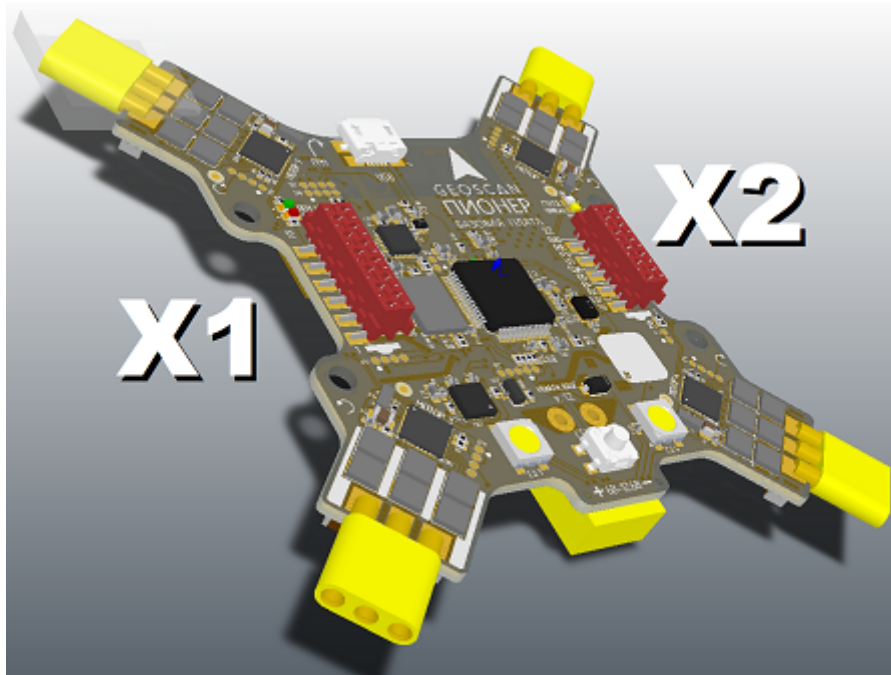
- 
11. Stop the motors after landing. To do this, place the left stick in the lower left position.



## 1.4 Extension modules



Pioneer's functions can be extended with different programmable modules.



X1 and X2 connectors are situated in the upper part of the main board. Most modules are mounted on the extension board and connect to the main board with parallel cables. GPS and indoor navigation modules are installed directly on the main board.

---

**Note:** Two connectors differ with contact amount and physical size, so they are NOT interchangeable. Use marking lip on the cable connector and do not push it hard.

---

The following sections establish connect and programming instructions.

---

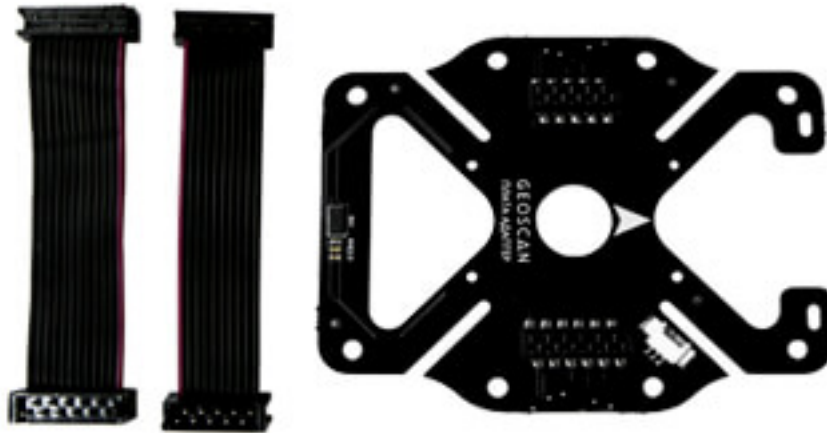
**Note:** Modules are programmed before the flight using [Pioneer Station](#) or [TRIK studio](#)

---

**Note that connected modules consume additional energy, which shortens the flight time**

### 1.4.1 Extension modules board

Extension modules board provides the ability to plug in different module to the base board.



Use the arrow markers to install the board properly (they should point in the same direction). The board itself is mounted on the lower part of the drone instead of battery bay cover. Use 4 stands and M3 screws.

There are two connectors on top of the extension board. They are used to connect with the base board using two parallel cables. Two connectors on the bottom are used to plug in different modules. The connection is established “through” the board.

The board is equipped with laser range sensor as well. It provides the ability to precisely estimate the altitude up to 1.5 m above the surface. The sensor is activated in alt.hold mode. Pioneer will remain the same altitude regarding the surface. If the distance between the drone and the surface changes, Pioneer will react, trying to compensate the change.

You can acquire the sensors data to control the flight. As an example, here is a program which will change the colour of Pioneer LEDs depending on its distance to the ground. Use Pioneer Station to [upload the program](#) to Pioneer.

---

**Note:** The range sensor is not active in GPS flight mode. If the program doesn't work, open the “[autopilot parameters](#)” tab in Pioneer Station window and select LPS or OPT parameters setting. You can also change the BoardPioneer\_modules\_gnss parameter to 0.0 manually.

---

```
-- https://learnxinyminutes.com/docs/lua/ - Lua basic functional manual
-- simpligy function call for laser range sensor distance data
local range = Sensors.range
-- number of leds on the baseboard
local ledNumber = 4
-- creating led control port
local leds = Ledbar.new(ledNumber)

-- led colour change function
local function changeColor(red, green, blue)
    -- changes colour of reach led
    for i = 0, ledNumber - 1, 1 do
        leds:set(i, red, green, blue)
    end
end
```

(continues on next page)

```

end

function callback(event)
end

-- create a timer to get altitude from the sensor every 0.1 second
timerRangeRead = Timer.new(0.1, function ()
    -- get altitude on meters from laser sensor (comes №5 in line)
    _, _, _, distance = range()
    r, g, b = 0, 0, 0
    -- if range is beyond maximum value , it will show constant altitude of 8.19 m
    if (distance >= 8.19) then
        -- in that case leds will colour red
        r = 1
    else
        -- Change leds brightness according to distance ( ~1.5 m is a maximum range for laser
        ↪ sensor on the extension board.)
        g = math.abs(distance / 1.5)
    end
    -- change led colour to stated above
    changeColor(r, g, b)
end)
-- timer start
timerRangeRead:start()

```

## 1.4.2 Cargo module



Cargo module is designed to hold magnetic objects. It is equipped with electromagnet on a flexible gimbal and 4 RGB LEDs.

Cargo module is mounted on the extension board using 4 M3 screws.

Use 8 channel of the RC unit to operate the module. A free SwA flipswitch can be used. To set the channel, turn on the RC unit, use touchscreen to select function-AUX Channels - 8 and choose SwA as



---

a primary. Now you should [upload the program](#) for Pioneer's cargo module. Use the given code, which allows to control the magnet with SwA switch.

```
-- Magnet control port (PC3) initialize for Pioneer_Base v 1.2
local magnet = Gpio.new(Gpio.C, 3, Gpio.OUTPUT)
-- Magnet control port (PA1) initialize for Pioneer_Base v 1.1 (    uncomment line below and
←comment line above)
-- local magnet = Gpio.new(Gpio.A, 1, Gpio.OUTPUT)

-- Total number of LEDs (4 on base pcb and 4 on a cargo module)
local led_number = 8
-- RGB LED control port initialize
local leds = Ledbar.new(led_number)
-- Magnet state (switched on initially)
local magnet_state = true

-- Function that sets LEDs color based of magnet state
local function setLed(state)
  if (state == true) then
    color = {1,1,1}          -- If magnet is on, color is  white
  else
    color = {0,0,0}          -- If magnet is off, color is  black (LEDs are off)
  end
  for i = 4, led_number - 1, 1 do    -- Set color for each of LEDs
    leds:set(i, table.unpack(color))
  end
end

-- Magnet switch function
local function toggleMagnet()
  if (magnet_state == true) then -- If magnet is on, we switch it off
    magnet:reset()
  else
    magnet:set()                -- If magnet is off, we switch it on
  end
  magnet_state = not magnet_state -- Changing magnet state value to  appropriate state
end

-- Required callback function to process events
function callback(event)
end

-- Timer creation, that calls our function each second
cargoTimer = Timer.new(1, function ()
  toggleMagnet()
  setLed(magnet_state)
end)
-- Timer start
cargoTimer:start()
```

---

### 1.4.3 GPS/GLONASS module



The module allows to track current quadcopter's speed and position using global navigation systems. For better accuracy the drone should perform an outdoor flight away from high buildings and metal constructions.

GPS module is mounted on the main board using two connectors and M3 screws. The connector provide “run-through” connection to plug in extra modules on the lower board with parallel cables. GPS module is also equipped with a compass to get orientation data. This can be corrupted near large metal objects and buildings.

When [connected to Pioneer Station](#), make sure the GPS positioning mode is selected in “autopilot parameters”, otherwise activate it manually. You can observe compass working in aviahorizon window when connected to PC in standard mode. The quantity of connected satellites is also shown there. The more satellites is “visible”, the more positioning accuracy you gain. First-time activation in new location (cold-start) requires 1-3 minutes for module to synch. In case of success, green light on the module will stop flashing. Now, if disabled and then turned on in the same location after some time, the synchronization process will be much shorter.

LED	status	todo
red and green	wrong parameters	change parameters for GPS
red	satellite search	wait 1-3 minutes, reboot
green	satellites connected	takeoff and fly

When flying with radio transmitter control, use navigation system mode (SwB switch in middle position) to use GPS module and make the flight significantly more comfortable.

**ap.goToLocalPoint(x, y, z)** function is used for programmed flight. X - axis is directed to the East, Y - axis to the North. Z represents the altitude.

Pioneer gets its position from the switch-on moment. This point is considered to be (0, 0, 0) in ap.goToLocalPoint command.

Also, you can use **ap.goToPoint(x, y, z)** function. Here, x and y show geographical coordinates (latitude, longitude) of a point where Pioneer will fly. Z is still for altitude at this point.

---

**Note:** If stated start point is more than 500 meters away from the actual, quadcopter will not take-off. Also, Flight\_com\_flyAreaSize and Flight\_com\_maxAltitude parameters set possible distance and altitude from the start point. Look for parameters description in Pioneer Station for more info.

---

---

## Example

Below is an example of a GPS module program. Upload it to Pioneer, find a spacious site and connect the LiPo battery. Wait until the module LED stops blinking, then switch SwB to the lower position and push "Start" button on the quadcopter's base board. In 5 seconds Pioneer will take-off, fly 10 meters east and return to the launch point.

Pioneer acquires its position at the moment the motors arm. This point is considered (0, 0, 0) in the `ap.goToLocalPoint` command.

GPS position and altitude error may reach 3 meters, keep this in mind while planning the flight.

```
-- Simplification and caching table.unpack calls
local unpack = table.unpack

-- Base pcb number of RGB LEDs
local ledNumber = 4
-- RGB LED control port initialize
local leds = Ledbar.new(ledNumber)

-- Function changes color on all LEDs
local function changeColor(color)
    -- Changing color on each LED one after another
    for i=0, ledNumber - 1, 1 do
        leds:set(i, unpack(color))
    end
end

-- Table of colors in RGB form for changeColor function
local colors = {
    {1, 0, 0}, -- red
    {0, 1, 0}, -- green
    {0, 0, 1}, -- blue
    {1, 1, 0}, -- yellow
    {1, 0, 1}, -- violet
    {0, 1, 1}, -- cyan
    {1, 1, 1}, -- white
    {0, 0, 0} -- black/switched off
}

-- Flight mission points table formatted as {x,y,z}
local points = {
    {0, 0, 0.7},
    {0, 1, 0.7},
    {0.5, 1, 0.7},
    {0.5, 0, 0.7}
}

-- Current point variable
local curr_point = 1

-- Function that changes LEDs color and flies to the next point
local function nextPoint()
    -- Current color. % - modulo, # - table size
    -- This expression is used in order to circle through color's
    -- table even it is not
    -- the same size as points table
    curr_color = ((curr_point - 1) % (#colors - 2)) + 1
    -- Changing LEDs color
    changeColor(colors[curr_color])
    -- Flying to the next point if its number is less than number of
    -- points in the table
    -- "points"
    if(curr_point <= #points) then
        Timer.callLater(1, function()
            -- Fly to point command in Positioning system
        end)
    end
end
```

(continues on next page)

```

        ap.goToLocalPoint(unpack(points[curr_point]))
        -- Current point variable increase
        curr_point = curr_point + 1
    end)
-- Landing initiate if number of current point exceeds number of points in total
else
    Timer.callLater(1, function()
        -- Landing command
        ap.push(Ev.MCE_LANDING)
    end)
end
end

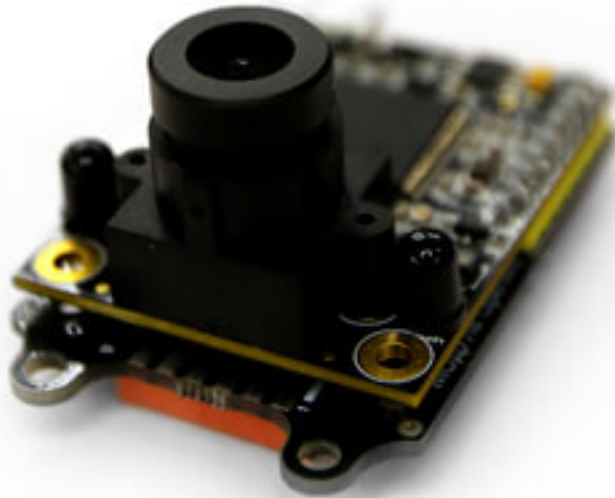
-- Event processing function called automatically by autopilot
function callback(event)
    -- After Pioneer reaches Flight_com_takeoffAlt, it starts mission flight
    if(event == Ev.TAKEOFF_COMPLETE) then
        nextPoint()
    end
    -- When Pioneer reaches current point it initiates flight to next point
    if(event == Ev.POINT_REACHED) then
        nextPoint()
    end
    -- After Pioneer lands, it switches off LEDs
    if (event == Ev.COPTER_LANDED) then
        changeColor(colors[8])
    end
end

-- Pre-start preparations
ap.push(Ev.MCE_PREFLIGHT)
-- Changing LEDs color to white
changeColor(colors[7])
-- Timer, that calls takeoff function after 2 seconds
Timer.callLater(2, function() ap.push(Ev.MCE_TAKEOFF) end)

```

---

## 1.4.4 OpenMV camera



OpenMV camera provides the ability for Pioneer to acquire video feed and process it onboard.

The camera is installed on the extension board using a dedicated module with X1 and X2 connectors. You may use two stands with screws to secure the camera.

For more details see [OpenMV documentation page](#)

OpenMV module communicates with the main Pioneer board using UART interface. You need two separate programs for Pioneer and camera to structure this protocol. Use the following code for Pioneer:

```
-- initialize UART interface
local uartNum = 4 -- UART interface number (USART4)
local baudRate = 9600 -- data speed
local stopBits = 1
local parity = Uart.PARITY_NONE
local uart = Uart.new(uartNum, baudRate, parity, stopBits)
```

You should also activate the connection protocol on the camera. Для этого [Download and run the OpenMV IDE](#). Connect the camera to your PC with microUSB cable and click “Connect” button in the lower left corner of IDE window. From now the code from textbox may be uploaded to the camera by clicking “run” button.

Initialization code for UART interface on the camera:

```
uart = UART(3) [font=monospace] [/font]
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000)
```

Comm protocol between Pioneer and the camera is created now.

An example code is established below. The program will change LED brightness depending on the brightness of OpenMV camera image. Both pieces of code (for Pioneer and the camera) contain UART protocol setting.

Code for Pioneer

```

-- -- https://learnxinyminutes.com/docs/lua/ - Lua basic functional manual
-- number of leds on the baseboard
local ledNumber = 4
-- create led control port
local leds = Ledbar.new(ledNumber)

-- function to change base board led colour
local function changeColor(red, green, blue)
  for i=0, ledNumber - 1, 1 do
    leds:set(i, red, green, blue)
  end
end

-- function to change leds colour red and turn off the timer
local function emergency()
  takePhotoTimer:stop()
  -- As timer function executes once more after its stop, turn leds red in a second
  Timer.callLater(1, function () changeColor(1, 0, 0) end)
end

-- define function to analyse system events
function callback(event)
  -- check battery voltage
  if (event == Ev.LOW_VOLTAGE2) then
    emergency()
  end
end

changeColor(1, 0, 0) -- red

-- initialize UART interface
local uartNum = 4 -- UART interface number (USART4)
local baudRate = 9600 -- data speed
local dataBits = 8
local stopBits = 1
local parity = Uart.PARITY_NONE
local uart = Uart.new(uartNum, baudRate, parity, stopBits) -- create comm protocol

changeColor(1, 0, 1) --purple

--changeColor(1, 0.4, 0) --yellow

local N = 10
local i = 7
local strUnpack = string.unpack
function getData() -- function to get data byte
  --uart:write('abc', 3)
  --return 50
  --return uart:read(1) or 0
  i = i + 1
  if (i == N + 1) then i = 0 end
  buf = uart:read(uart:bytesToRead()) or '0'
  if (#buf == 0) then buf = '\0' end
  --buf = '0'
  leds:set(1, 0, i/N, 0.5 - 0.5*i/10)
  --local chr = buf[#buf]
  if (strUnpack ~= nil) then
    local b = strUnpack("B", buf)
    return b --
  else
    return 20
  end
end

```

(continues on next page)

(continued from previous page)

```
--string.byte(buf)
--return #buf
--return string.byte(string.sub(buf, -1)) or 0
--return string.byte(buf) or 0
end

local takerFunction = function () -- function to read UART data
  local intensity = getData() / 100.0
  changeColor(intensity, intensity, intensity)
end
local interval = 0.1
getMeasureTimer = Timer.new(interval, takerFunction) -- photo timer
getMeasureTimer:start()

changeColor(1, 0.2, 0) -- orange
```

## Code for OpenMV

```
# Hello World Example
#
# Welcome to the OpenMV IDE! Click on the green run arrow button below to run the script!

from pyb import UART, LED

import sensor, lcd, image, time, utime

ledBlue = LED(2)
ledGreen = LED(3)

ledBlue.on()
sensor.reset() # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.LCD) # Set frame size to QVGA (320x240)
sensor.skip_frames(100) # Wait for settings take effect.
clock = time.clock() # Create a clock object to track the FPS.
lcd.init()
#lcd.set_backlight(True)
ledBlue.off()

#Init uart

uart = UART(3)
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000) # init with given parameters

M_LED_COUNT = 10
led_counter = M_LED_COUNT
led_mode = 0
while(True):
  clock.tick() # Update the FPS clock.
  clk = utime.ticks_ms()
  img = sensor.snapshot() # Take a picture and return the image.
  #print(clock.fps()) # Note: OpenMV Cam runs about half as fast when connected
  # to the IDE. The FPS should increase once disconnected.

  for r in img.find_rects(threshold = 40000):
    img.draw_rectangle(r.x(), r.y(), r.w(), r.h(), (255, 0, 0))
    for p in r.corners():
      img.draw_circle(p[0], p[1], 5, color = (0, 255, 0))
  print(r)
```

(continues on next page)

```

lcd.display(img)

print(img.get_histogram().get_statistics().l_mean())
uart.writechar(img.get_histogram().get_statistics().l_mean())
led_counter = led_counter - 1
if (led_counter == 0):
    if (led_mode == 0):
        ledGreen.on()
    else:
        ledGreen.off()
    led_mode = 1 - led_mode
    led_counter = M_LED_COUNT
while (clk + 100 > utime.ticks_ms()):
    pass

```

Use Pioneer station and OpenMV IDE to [upload](#) each program. Turn Pioneer on and run the mission. Test how it works by pointing the camera towards objects with different brightness, or simply cover its lens by hand.

### 1.4.5 Optical flow module



This module gives Pioneer the ability to position itself without using local or global navigation system. Optical sensor tracks object's shift. The module is mounted below on the extension board. Connect it to the base board using flexible wire band.

The module is also equipped with optical distance sensor. It can measure altitude above the floor or other objects from 0 to 1.5 m range. When Pioneer exceeds this value, the barometer is used for altitude control.

`ap.goToLocalPoint(x, y z)` is used for programmed flight. Pioneer sets the start point (0, 0, 0) when the motors are armed. Y - axes is directed forwards, X - axes is directed to the right. Z parameter represents the altitude. The (x, y, z) values set the coordinates where Pioneer will fly, based on start point.

---

**Note:** Positioning error increases with flight distance.

---

Module can recognize the surface below it. The more details like ornament or tile junctures is visible, the more precise the flight will be.

Select navigation system mode (Swb switch in middle position) to use optical flow module during manual-control flight. This will make it much more stabilized.



---

## Optical flow module firmware update

To update the module firmware you should have Pioneer Station installed on your computer. Select **“Firmware update”** in program’s menu and follow the wizard instructions. On “device choice” tab you will see both Pioneer Base and Optical flow module. Click the module checkbox and proceed. It is recommended to select “internal” firmware source. Wait for update to end, the quadcopter will be reloaded to standard mode.

### Example

This is an example program for optical flow module. Use Pioneer Station to [upload it to Pioneer](#). After that, turn the drone on, switch SwB in lower position and push start button on it. In 5 seconds it will lift off, fly 1 meter forward and return to launch point.

```
-- number of leds on base board (4)
local ledNumber = 4
-- create led control port
local leds = Ledbar.new(ledNumber)
-- speeding up function calls
local unpack = table.unpack

-- current state value
local curr_state = "PREPARE_FLIGHT"

-- function to change LEDs colour
local function changeColor(color)
  for i=0, ledNumber - 1, 1 do
    leds:set(i, unpack(color))
  end
end

-- RGB colour code array to transfer changeColor
local colors = {
  {1, 0, 0}, -- red
  {1, 1, 1}, -- white
  {0, 1, 0}, -- green
  {1, 1, 0}, -- yellow
  {1, 0, 1}, -- purple
  {0, 0, 1}, -- blue
  {0, 0, 0} -- black/LEDs turn off
}

-- array of functions, called depending on current condition
action = {
  ["PREPARE_FLIGHT"] = function(x)
    changeColor(colors[2]) -- change LED colour to white
    Timer.callLater(2, function () ap.push(Ev.MCE_PREFLIGHT) end) -- starting motors in 2
↵seconds
    Timer.callLater(4, function () changeColor(colors[3]) end)-- change colour to green in
↵2 more seconds (4 seconds in total since timers start one after another right away)
    Timer.callLater(6, function ()
      ap.push(Ev.MCE_TAKEOFF) -- takoff in 2 more seconds (6 in total after all 3 timers
↵are being set)
      curr_state = "FLIGHT_TO_FIRST_POINT" -- next condition
    end)
  end,
  ["FLIGHT_TO_FIRST_POINT"] = function (x)
    changeColor(colors[4]) -- change colour to yellow
    Timer.callLater(2, function ()
      ap.goToLocalPoint(0, 1, 1) -- go to point with coordinates (0 m, 1 m, 1 m)
      curr_state = "FLIGHT_TO_SECOND_POINT" -- next condition
    end)
  end)
end)
```

(continues on next page)

```

end,
["FLIGHT_TO_SECOND_POINT"] = function (x)
  changeColor(colors[5]) -- change colour to purple
  Timer.callLater(2, function ()
    ap.goToLocalPoint(0, 0, 0.8) -- go to next start with coordinates (0 m, 0 m, 0.8 m)
    curr_state = "PIONEER_LANDING" -- next condition
  end)
end,
["PIONEER_LANDING"] = function (x)
  changeColor(colors[6]) -- change colour to blue
  Timer.callLater(2, function ()
    ap.push(Ev.MCE_LANDING) -- perform landing
  end)
end
}

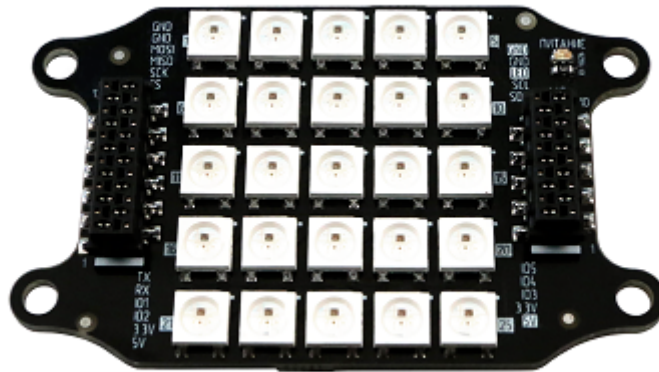
-- condition processing function (created by autopilot automatically)
function callback(event)
  -- if set altitude reached, execute function from the array according to current condition
  if (event == Ev.TAKEOFF_COMPLETE) then
    action[curr_state]()
  end
  -- turn LEDs red in case of collision
  if (event == Ev.SHOCK) then
    changeColor(colors[1])
  end
  -- if set waypoint reached, execute function from the array according to current condition
  if (event == Ev.POINT_REACHED) then
    action[curr_state]()
  end
  -- turn off LEDs after landing
  if (event == Ev.COPTER_LANDED) then
    changeColor(colors[7])
  end
end

-- turn red LED on
changeColor(colors[1])
-- start 2-second timer and execute first array function (flight preparation)
Timer.callLater(2, function () action[curr_state]() end)

```

---

## 1.4.6 LED module



LED module includes a board with 25 programmable leds, which is attached to the extension board. The connection is provided using X2 connector with parallel cable.

Depending on uploaded program, a LED module is used for backlighting or colour indication.

As an example a random colour program is given below. Use Pioneer Station to [upload the program](#)

```
-- Number of leds on base board (4) + leds on module (25)
local ledNumber = 29
-- create led control port
local leds = Ledbar.new(ledNumber)

-- function to set leds colour
local function changeColor(red, green, blue)
  for i=0, ledNumber - 1, 1 do
    leds:set(i, red, green, blue)
  end
end

-- function to turn off leds and timerRandomLED
local function emergency()
  timerRandomLED:stop()
  -- As timer function executes once more after its stop, turn leds off in a second
  Timer.callLater(1, function () changeColor(0, 0, 0) end)
end

function callback(event)
  -- checks battery voltage
  if (event == Ev.LOW_VOLTAGE2) then
    emergency()
  end
end

-- create timer to change leds colour randomly every second
timerRandomLED = Timer.new(1, function ()
  changeColor(math.random(), math.random(), math.random())
end)
-- timer start
timerRandomLED:start()
```

for more complex LED module programming see the example below:

```

local led_count = 29 -- Total number of LEDs
↳ LEDs (4 on base pcb and 25 on the LED module)
local led_offset = 4 -- Number of LEDs on base pcb
↳ base pcb
local matrix_count = 25 -- Number of LEDs on LED module
↳ LED module
local leds = Ledbar.new(led_count) -- RGB LED control
↳ port initialize
local unpack = table.unpack -- Simplification and caching
↳ caching table.unpack calls
local colors = { -- Table of RGB colors.
  red = {1, 0, 0},
  ↳ Color brightness is set via [0;1] interval
  green = {0, 1, 0},
  blue = {0, 0, 1},
  purple = {1, 0, 1},
  cyan = {0, 1, 1},
  yellow = {1, 1, 0},
  white = {1, 1, 1},
  black = {0, 0, 0} -- black - LEDs are off
}

local dig = { -- 1 Table of number representations
  {3, 7, 8, 13, 18, 22, 23, 24}, -- 2
  {2, 3, 4, 9, 12, 13, 14, 17, 22, 23, 24}, -- 3
  {2, 3, 4, 9, 12, 13, 14, 19, 22, 23, 24}, -- 4
  {2, 4, 7, 9, 12, 13, 14, 19, 24}, -- 5
  {2, 3, 4, 7, 12, 13, 14, 19, 22, 23, 24}, -- 6
  {3, 4, 7, 12, 13, 14, 17, 19, 22, 23, 24}, -- 7
  {2, 3, 4, 9, 13, 18, 23}, -- 8
  {2, 3, 4, 7, 9, 13, 17, 19, 22, 23, 24}, -- 9
  {2, 3, 4, 7, 9, 12, 13, 14, 19, 22, 23}, -- 10
  {1, 3, 4, 5, 6, 8, 10, 11, 13, 15, 16, 18, 20, 21, 23, 24, 25}, -- Since Lua starts counting from 1, so 0 is set in explicit form
  [0] = {2, 3, 4, 7, 9, 12, 14, 17, 19, 22, 23, 24} }

local ledMatrix = {} -- Array of output
↳ info for LED matrix

for i = 1, matrix_count + 1, 1 do
  ledMatrix[i] = colors.black -- Array initialization
end

-- Array to matrix output function
local function updateMatrix()
  for i = led_offset, led_count - 1, 1 do
    leds:set(i, unpack(ledMatrix[i-led_offset + 1]))
  end
end

-- Specific pixel of matrix array set function (x - column, y - row, colors - color in RGB form)
↳ form)
local function setPixelMatrix( x, y, colors )
  i = (y - 1) * 5 + x
  if ledMatrix [i] then
    ledMatrix [i] = colors
  end
end

-- Filling matrix array with color function (colors - color in RGB)
local function fillMatrix( colors )
  for i = 1, matrix_count + 1, 1 do
    ledMatrix [i] = colors
  end
end

```

(continues on next page)

```

end

-- Digit symbol to matrix array write function (x - digit, colors - color in RGB form)
local function setDig( x, colors )
  for _, v in ipairs(dig[x]) do
    ledMatrix[v] = colors
  end
end

-- The end of affecting matrix functions
-----
↔-----

function callback( event )
end

-- Example. This function outputs digits from 0 to 9 while changing colors from red to violet
function digitOutput()
  colors_any[1], colors_any[2], colors_any[3] = fromHSV(col, 100, 10)  -- Color generation
  setDig(i, colors_any)  -- Loading digit ↵
↔to array of set color
  updateMatrix()  -- Array output to ↵
↔matrix
  if col < 360 then
    col = col + 1  -- Increasing ↵
↔color value
    elseif i < #dig-1 then
      fillMatrix(colors.black)  -- Clearing matrix ↵
↔array before loading new digit
      col = 0  -- Zeroing color ↵
↔value
      i = i + 1  -- Increasing ↵
↔digit value
    else
      fillMatrix(colors.black)  -- Zeroing color ↵
↔value
      i = 0  -- Zeroing digit ↵
↔value
    end
    Timer.callLater(0.003, function () digitOutput() end)  -- Color updating ↵
↔period
end

colors_any = {0,0,0}  -- Color variable in RGB-format
i = 0  -- Digit variable
col = 0  -- Color variable in HSV-format
digitOutput()  -- Start program

```

---

### 1.4.7 GoPro camera gimbal



The mount is attached to the lower part of the frame with M3x5 screws, providing a solid Gopro camera joint to capture onboard video during flight. Its vertical angle may be adjusted.

Be careful while take-off and landing stages, as the mount with A Gopro affects the quadcopter's balance and dimensions.

---

## 1.4.8 Photo and video camera



This module can be used both for video shooting and FPV flights. Runcam Split camera is mounted on a rotating gimbal. Inside the module's body there is a control board with 2 buttons, microUSB connector and microSD slot on it. 4Gb memory card is already installed, and the footage is stored on it.

Use standard battery cover to mount camera module. Unscrew it from Pioneer, and secure the camera module on it with 4 screws. Then, install it back so that camera is below the quadcopter. Take the lower module cover to get access to Runcam Split control board.

Plug the camera connector to X1 on the Pioneer base board. Connect the battery to set camera modes.

It is convenient to control Runcam Split with smartphone app via Wi-Fi. Download and install it from PlayMarket of AppStore:

[IOS version](#) | [Android version](#)

2 buttons are used for camera controls. Both are situated on module board. "Power" is on the back part in the center, "mode" is near it but closer to the edge. Try pushing it with a small screwdriver. To activate Wi-Fi, push "mode" once. You should hear single beep. Now the camera is working as a Wi-Fi hotspot. Enable Wi-Fi on the smartphone and connect to "RCSplit" network. Password is 1234567890.

Run the Runcam app. Select Split 2S camera and "connect". Now the image is translated from camera directly on the smartphone screen. You can take pictures, shoot video and change camera settings. To open the camera album and copy files to smartphone's memory, push the icon to the left of shutter button. You can see the files directly on the SDcard, copy or delete them if you wish.

See the [Runcam Split instruction manual](#) for more details.

## 1.4.9 FPV kit

FPV (First Person View)

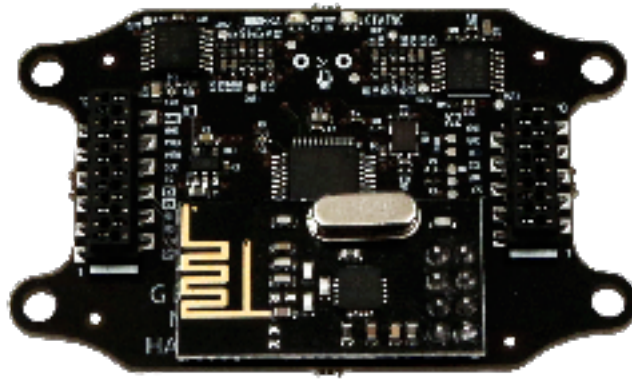
The kit allows the operator to acquire the camera image with minimal latency to control Pioneer's flight. The kit includes:

Runcam Split camera Video transmitter with antenna FPV goggles with a receiver

Runcam Split is also capable of shooting FullHD video and save it on a microSD memory card with up to 64 Gb storage capacity.

---

### 1.4.10 Onboard indoor navigation module



The module is included in indoor navigation system kit along with control module and 4 ultrasonic transducers. It is mounted on top of the main board.

The module is quipped with two microphones that allow the controller estimate phases and delays of ultrasonic signals. It is synced with the control module via radio channel and locates the drone's speed and position.

Turn Pioneer on and place it in ultrasonic beacon's operation area to activate the module.

#### **Additional:** Indoor navigation

Indoor flight can be performed both in manual or mission mode. Example of such mission program is established below. According to it, Pioneer will take-off, gains 1.2 m altitude, then flies to the corner of flyzone with (0:0) coordinates, and lands in the (1:1) point. Use Pioneer station to [upload the program](#)

```
-- current state variable
local curr_state = "PREPARE_FLIGHT"

-- table of functions, depending on state
action = {
  ["PREPARE_FLIGHT"] = function(x)
    Timer.callLater(2, function ()
      ap.push(Ev.MCE_TAKEOFF) -- send start comend to autopilot in 2 seconds
      curr_state = "FLIGHT_TO_FIRST_POINT" -- next state
    end)
  end,
  ["FLIGHT_TO_FIRST_POINT"] = function (x)
    Timer.callLater(2, function ()
      ap.goToLocalPoint(0, 0, 1.2) -- send command to autopilot to fly to piont (0 m, 0 m,
↪ 1,2 m)
      curr_state = "FLIGHT_TO_SECOND_POINT" -- next state
    end)
  end)
end)
```

(continues on next page)



```

end,
["FLIGHT_TO_SECOND_POINT"] = function (x)
    Timer.callLater(2, function ()
        ap.goToLocalPoint(1, 1, 1.2) -- send command to autopilot to fly to piont (1 m, 1 m,
↪ 1,2 m)
        curr_state = "PIONEER_LANDING" -- next state
    end)
end,
["PIONEER_LANDING"] = function (x)
    Timer.callLater(2, function ()
        ap.push(Ev.MCE_LANDING) -- send command to autopilot to land
    end)
end
}

-- turn on the red LED
changeColor(colors[1])
-- Start single 2 seconds timer, then execute first function from the table (preflight)
Timer.callLater(2, function () action[curr_state]() end)

```

### Navigation module firmware update

To update module's firmware, you will need Pioneer Station installed on your PC. Select Firmware update in the menu and follow the instructions on the screen. At 'choose device' step, both Pioneer base board and Navigation module will be displayed on the list. Select the module and click 'next'

It is recommended to choose default firmware source. If module version is not defuned automatically, read the marking on the backside of the plate and select the corresponding firmware file from Pioneer Station folder. Wait for the update to finish. Quadcopter will reboot in standard mode after that.

## 1.5 Programming

Besides flying in RC mode, Pioneer can execute pre-loaded flight mission autonomously. It supports Pioneer Station and TRIK studio to create and upload flight mission. In this section you will learn how to use Pioneer Station and see some mission examples.

### 1.5.1 Visual programming TRIK



TRIK studio is a graphical application intended to create robot missions using functional blocks. This approach simplifies your interaction with Pioneer. You don't need programming experience to create and upload a mission to the drone.

To install the app, click [download TRIK Studio](#) and run .exe file.

You also need [Pioneer Station](#) to work with Pioneer..

#### Creating mission in TRIK studio

Open TRIK studio and create new project (button in the top left corner). To create mission, drag and drop action blocks from Pallet to robot performance diagram, and then connect them.

#### TRIK studio basics

- 
- Every mission should include “start” and “end” blocks. If the mission has multiple branches, each branch has to end with “end” block.
  - To connect two blocks, use right mouse button. Press RMB on the first block, hold it, drag towards another block and release above it. Now these blocks are connected with a visible line.
  - To tune selected block, use “Properties editor” window.
  - To delete unnecessary block, select it with left mouse click and press del. key
  - If you need to select multiple blocks, circle them holding left mouse button, or hold ctrl key and select them one by one.
  - The diagram should not have red arrows and disconnected blocks. For better visibility arrange all blocks from top left to bottom right corner.
  - Learn “hot keys” manual in “settings” tab. **Help (F1)** tab may be useful too.

### Pioneer programming blocks

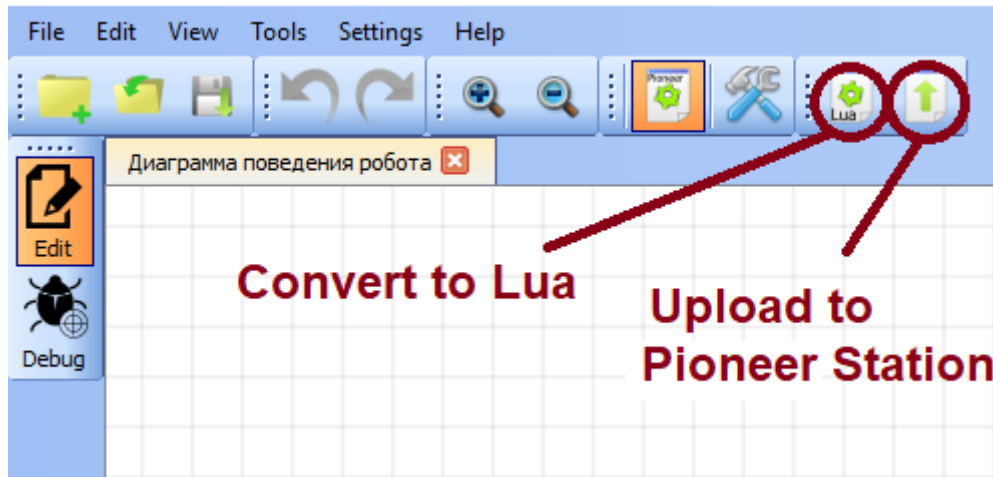
- **Condition** - creates two action scenarios depending on logical condition. The block should have two outgoing connections, one of which has to contain “condition” parameter (True or False)
- **End if** - marks merging point for two conditional operator branches. Provides structural integrity for diagram.
- **Variable initialisation** - states new variable. Use property editor to name and change it.
- **Random initialisation** - randomises selected value. Random range may be set.
- **Comments** - includes comments to provide better structure understanding.
- **Timer**- set time delay in milliseconds before next block will be executed.
- **Takeoff , Landing** - first and last block to perform flight mission.
- **Go to point** - set destination point coordinates. Don’t use dots or commas for latitude and longitude lines. “Altitude” shows distance to the ground (in meters) at the point.
- **Go to local point** - similar to previous, but destination point is set in local coordinates. Takoff point is considered for (0,0,0). Offset is set in meters.
- **LED** - controls Pioneer main board leds. Change each colour value to tune it precisely. Pair with “timer” block to set glowing duration.
- **Magnet** - controls **cargo module** operation. To turn magnet on, place a mark in a property checkbox.
- **System** - executes given instruction. Use Lua to create instruction in the editor and place a mark in a checkbox to run it.
- **Yaw** - controls Pioneer’s horizontal flight direction. To turn clockwise, set the angle with a “minus” sign.

Combining these blocks and connections, you can create any kind of flight mission imaginable.

### Mission example

First of all, learn to control Pioneer’s LEDs. In TRIK studio, press “open project” and select LED file. Mission diagram should appear in the workspace. Once activated, it will change led colour every second. Feel free to configure the mission to your liking.

Now it is necessary to create correct Lua code. Press “generate to Pioneer Lua” button or use ctrl+shift+G key combination.



If there's no errors in the program, a code tab will open. Otherwise, all errors will be shown in the lower part of the screen along with troubleshoot tips.

When code is generated, press [upload to Pioneer](#)

When familiar with led controls, try [more advanced options](#).

### Takeoff-mission-landing

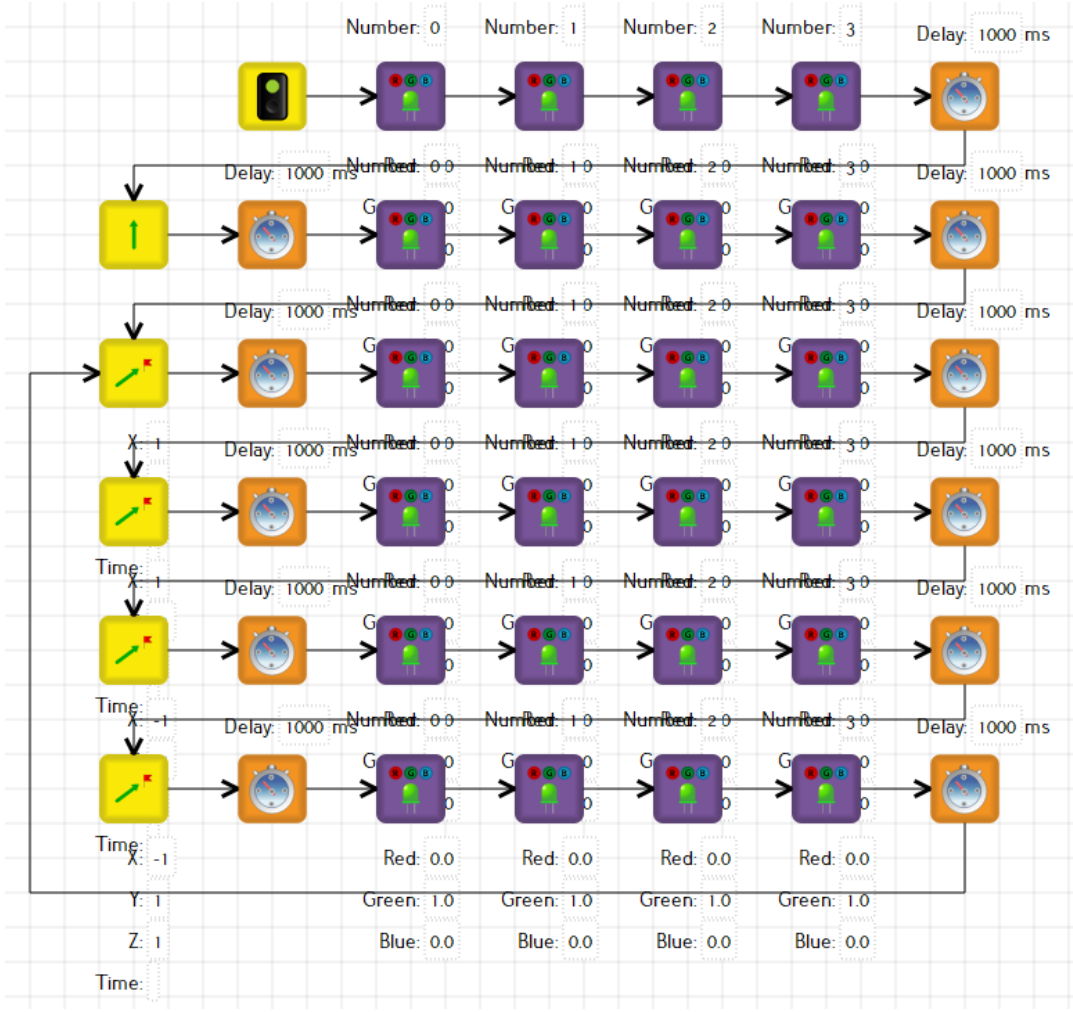
To create real flight mission you can use examples [fly\\_square](#) or [fly-to-point](#) from TRIK library.

---

**Note:** Note that [fly-to-point](#) mission requires local positioning provided by [indoor navigation system](#). The system should be deployed and activated beforehand.

---

To fly in global coordinates, a [GPS module](#) is required.



Add “Yaw” and “Led” blocks to diversify your flight.

Having created the diagram, don't forget to generate Lua code and check it for errors. Mission upload is performed *similarly as before*.

---

## 1.5.2 Pioneer Station



Pioneer Station is a tool to work with the drone. Using it, you can:

- [Update quadcopter's firmware](#)
- [Set autopilot parameters](#)
- [Run modules diagnostics](#)
- [Create and upload flight missions](#)
- [Test scripts](#)

To install the app, click [download Pioneer station](#) and run .exe file.

If you use Linux OS, download and unpack [PioneerStationLinux64](#)

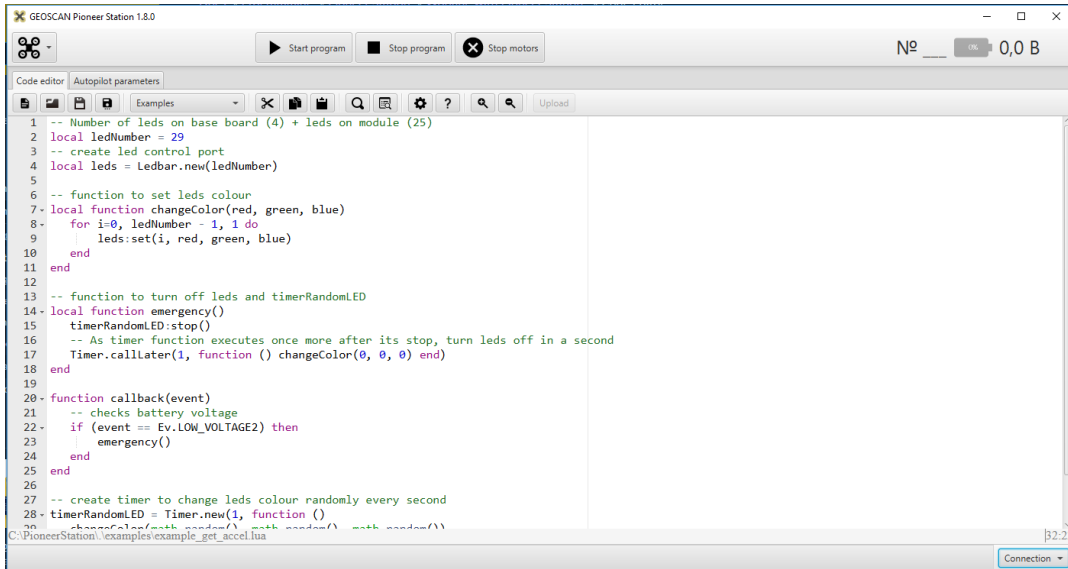
---

**Note:** If program doesn't start, download [32bit version](#), unpack the archive and run.bat

---

### Working with Pioneer Station

This is how program's interface looks like when Pioneer is not connected:



**Attention:** Do not run the flight mission with USB cable attached.

## Code editor









This section describes code editor operation in Pioneer Station








In **Pioneer Station**, main tab is used as a code editor. Here you can develop flight mission code for Pioneer.

Code editor main menu:



Control buttons:

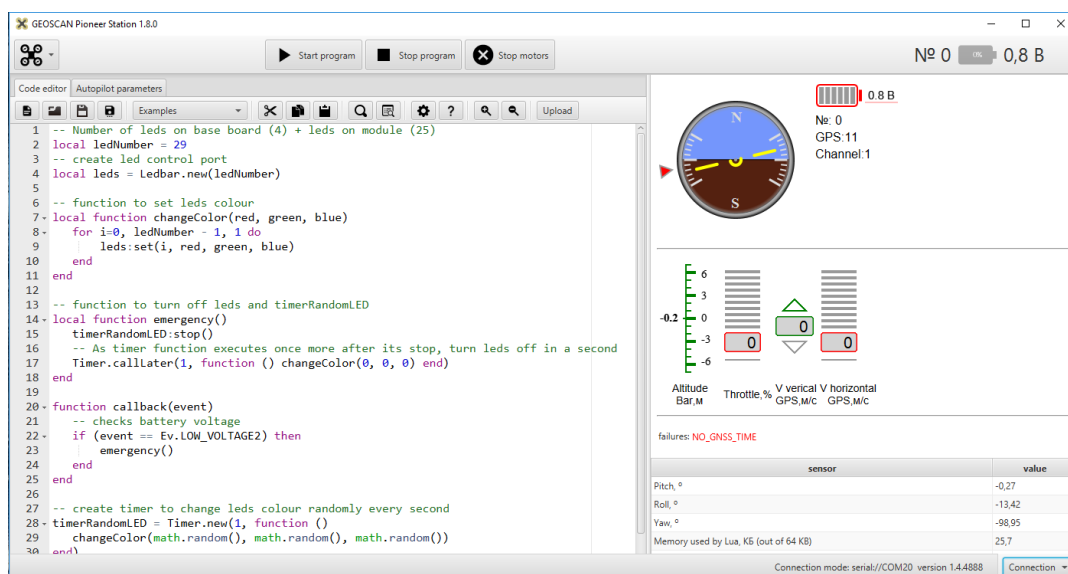
- **Create new file** 
- **Open file** 
- **Save** 
- **Save as** 
- **Open example** 
- **Cut** 
- **Copy** 
- **Paste** 

- Search 
- Insert 
- Settings 
- Help 
- Zoom in 
- Zoom out 
- Upload code to Pioneer (active when drone connected) 

### Program upload to Pioneer

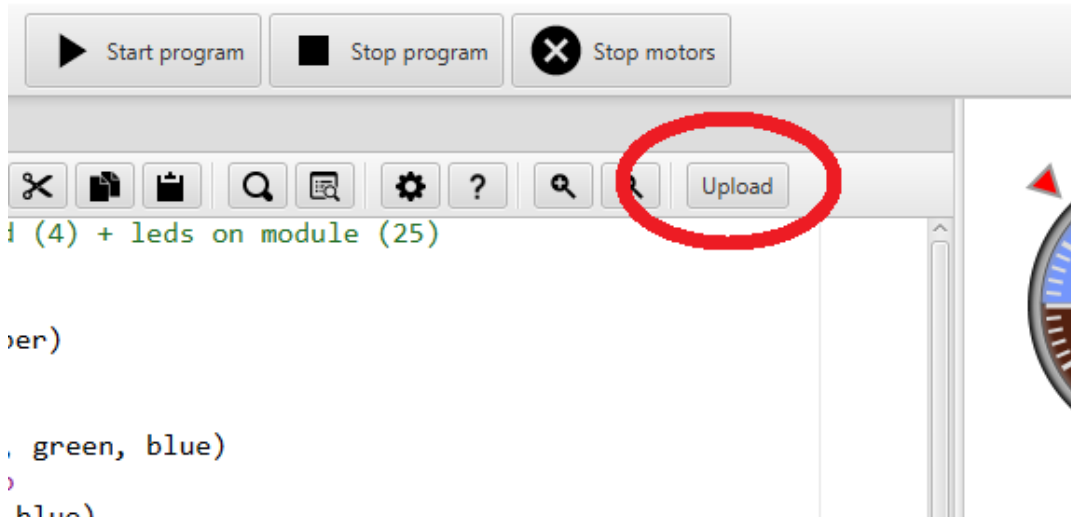
After you developed program using TRIK Studio or Pioneer Station, it is time to upload it to Pioneer. Connect the drone to your PC.

Select “USB cable” connection in the bottom right corner of Pioneer Station window. Pioneer’s current parameters, lean angles and board number should appear on the screen.



**Note:** Try tilting the drone in different directions and make sure the angles are represented correctly. In case of error, check cable connection and firmware version on Pioneer, update it if necessary.

If program was created in TRIK studio, press “upload porgamm to Pioneer”. The code should appear in Pioneer Station window. Then press “Upload” button.



“Receive” LED should blink on Pioneer board, and a message appear in the bottom right corner, indicating successful upload.

There are different ways to start mission execution. If flight is not required, don't disconnect USB cable and press “Mission start” in Pioneer Station. In five seconds you will observe mission execution. USB power is enough to turn LEDs and some [extension modules](#) on.

**In other cases**, disconnect Pioneer from your PC and place it somewhere spacious enough to perform the mission. Plug in the battery. Wait for LEDs to stop blinking and press “Start” button on its board. The mission will execute in 5 seconds.

**Attention:** To fly safely, it is recommended to have an RC transmitter ready and bind it to the drone. Activate mission flight by switching SwB in lower position. All other switches should be in upper position. If necessary, switch SwB in upper position - and now you have controls of the drone.

### USB modem connection

You can use wireless connection to upload missions to Pioneer. A USB-powered wireless modem is required for this.

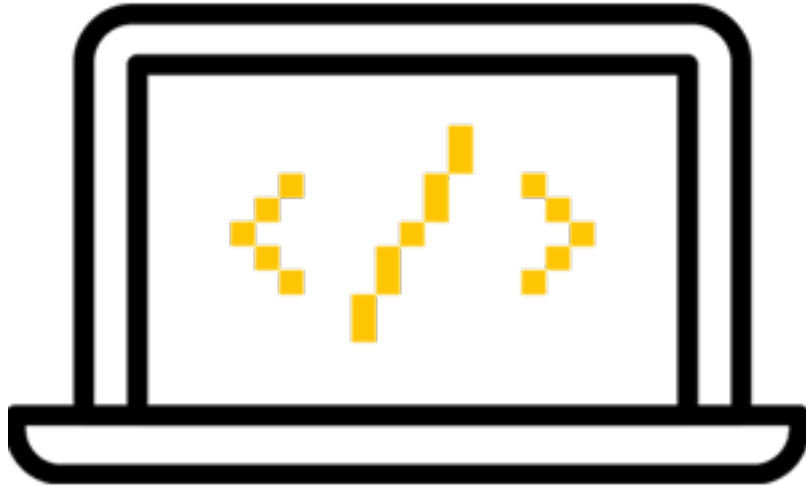




- Remember the channel number of your drone. It is displayed next to the avionic horizon unit when Pioneer is connected via USB cable.
- Select “modem setting” in “Settings” tab. Save your channel number in opened window.
- Select the desired drone to be connected and press OK button.
- To connect Pioneer wirelessly, select “Radiomodem” in the “connection” tab in the bottom right corner. The drone should be powered from LiPo or another USB 5V source.

---

### 1.5.3 Lua programming language



Standard Pioneer kit allows LEDs and flight mission programming. Two options are offered, depending on users competence:

Visual programming using [TRIK Studio](#)

Advanced programming using [Lua](#)

Lua basics are stated in [Learnlua.lua](#) at pioneer stations/examples. Open it with Pioneer Station and get used to syntax and basic command.

---

**Important:** External module can also be programmed and combined with standard Pioneer functions.

---

In Lua section you can learn basic functions used to program Pioneer.

#### Lua API documentation

##### Autopilot interaction API functions

`time()`

Returns time since the moment of autopilot turn on

`deltaTime()`

Returns difference between `time()`, and GNSS time in seconds.

`launchTime()`

Returns launch time for navigation system.

`sleep(seconds)`

Pauses script execution for stated amount of time, argument may be fractional. **It is recommended to use `:class:Timer`, as “sleep” blocks further script execution.**

**Parameters** `seconds` – Sleep time in seconds.

---

`boardNumber`

Returns aircraft ID - access using variable.

### Example

```
local boardNumber = boardNumber
```

`ap.push(Event)`

Add autopilot event (see *Autopilot events*).

**Parameters** `Event` – Event number or name (for example, `Ev.COPTER_LANDED`).

### Example

```
Ev.MCE_PREFLIGHT  
ap.push(Ev.MCE_PREFLIGHT)
```

`ap.goToPoint(latitude, longitude, altitude)`

for GPS mode flight.

### Parameters

- `latitude` – Set latitude in degrees, multiplied by  $10^{-7}$ ;
- `longitude` – Set longitude in degrees, multiplied by  $10^{-7}$ ;
- `altitude` – Set altitude in meters.

Example `ap.goToPoint(600859810, 304206500, 50)`

`ap.goToLocalPoint(x, y, z, time)`

For local positioning system:

### Parameters

- `x` – set point's *x* coordinate, meters.
- `y` – set point's *y*'coordinate, meters.
- `z` – set point's *z*'coordinate, meters.
- `time` – time required for copter to reach the next point, in seconds. If value is not stated, drone will fly with max speed.

Example `ap.goToLocalPoint(1, 1, 1.2)` или `ap.goToLocalPoint(1, 1, 1.2, 10)`

`ap.updateYaw(angle)`

Set yaw.

**Parameters** `angle` – angle in radian..

## RGB LEDs

`class Ledbar`

`new(Count)`

Create new Ledbar with stated amount of leds.

**Parameters** `Count` – Led amount.

`set(self, num, r, g, b)`

Set each led colour.

### Parameters

- `num` – Led number. Numeration 0 - 3 for main board, then for extra modules in series;
- `r` – red colour component intensity in [0;1] interval;

- *g* – green colour component intensity in [0;1] interval;
- *b* – blue colour component intensity in [0;1] interval;

See also *fromHSV()*

### Example

Cargo module includes a magnet and 4 RGB leds.

```
local leds = Ledbar.new(8)
for i = 0, 3, 1 do
  leds:set(i, 1, 0, 0)
end

for i = 4, 7, 1 do
  leds:set(i, 0, 0.5, 0)
end
```

*fromHSV(hue, saturation, value)*

Converts colour code from HSV to RGB. Can be used to set led colour.

#### Parameters

- *hue* – set colour tone. Varies in [0;360] interval;
- *saturation* – set saturation. Varies in [0;100] interval;;
- *value* – set colour; Varies in [0;100] interval;

**Returns** Returns RGB colour components.

### GPIO (general-purpose input/output interface)

class *Gpio*

*new(Port, Pin, Mode)*

create GPIO in settings port..

#### Parameters

- *Port* – *Gpio.A*; *Gpio.B*; ... *Gpio.E*;
- *Pin* – pin number in port;
- *Mode* – *Gpio.INPUT*, *Gpio.Output*, *Gpio.ALTFU*.

*read(self)*

Returns value.

*set(self)*

Set value in 1.

*reset(self)*

Set value in 0..

*write(self, value)*

**Parameters** *value* – set value.

*setFunction(self, num)*

Set alternative function number.

### Example

```
local pin_name = Gpio.new(Gpio.A, 1, Gpio.OUTPUT)
pin_name:read() -- get value
pin_name:set() -- set value 1
pin_name:reset() -- set value 0
pin_name:write(true) -- set value true
pin_name:setFunction(1) -- set alternative function number
```

---

## UART interface

class Uart

`new(num, rate, parity, stopBits)`  
Create UART in settings port..

**Parameters**

- `num` – UART number;;
- `rate` – speed;
- `parity` – Uart.PARITY\_NONE, Uart.PARITY\_EVEN, Uart.PARITY\_ODD, not a necessary parameter, by default Uart.PARITY\_NONE;
- `stopBits` – Uart.ONE\_STOP, Uart.TWO\_STOP, not a necessary parameter, by default Uart.ONE\_STOP.

`read(self, size)`  
Read size byte.

`write(self, data, size)`  
Write (data) length of (size).

`bytesToRead(self)`  
Amount of data available to be read..

`setBaudRate(self, rate)`  
Set speed.

**Parameters** `rate` – UART speed.

### Example

```
local uart = Uart.new(1, 115200)
uart:read(10) -- read 10 bytes
```

## SPI

class Spi

`new(num, rate, seq, mode)`  
Create SPI in settings port.

**Parameters**

- `num` – SPI number
- `rate` – speed
- `seq` – Spi.MSB, Spi.LSB, Spi.MSB\_16, Spi.LSB\_16, not a necessary parameter, by default Spi.MSB;
- `mode` – Spi.MODEo, Spi.MODE1, Spi.MODE2, Spi.MODE3, not a necessary parameter, by default Spi.MODEo.

`read(self, size)`  
read size byte.

`write(self, data, size)`  
Write (data) length of (size).

`exchange(self, data, size)`  
Write (data) length of (size) and read size.

### Example

```
local spi = Spi.new(2, 1000000)
spi:exchange("hello", 5) -- Write (data) length of (size) and read size
```

---

## Timers

class Timer

`new(sec, func)`

Create new Timer.

**Parameters**

- `sec` – Interval time in seconds.
- `func` – Function called with set interval.

`start(self)`

Start timer.

`stop(self)`

Stop timer. Full stop after last function execution.

`callAt(local_time, func)`

Creates and starts new timer with function to be called ONCE.

**Parameters**

- `local_time` – Local time (returns with `time()`), states function call moment.;
- `func` – Function to be called.

`callLater(delay, func)`

Creates and starts new timer with function to be called ONCE.

**Parameters**

- `delay` – Time before function start;
- `func` – Function to be called.

`callAtGlobal(global_time, func)`

Creates and starts new timer with function to be called ONCE.

**Parameters**

- `global_time` – Global time (`time() + deltaTime()`), when to call function.;
- `func` – Function to be called.

---

**Note:** When using `callAt()`, `callLater()`, `callAtGlobal()` functions, note that there can not be more than 16 waiting timers simultaneously. If this amount is exceeded, new tier will not be created.

---

## Example

See *Script example*

## Autopilot events

Events a presented using constants with “Ev” prefix

Name	Description
MCE_PREFLIGHT	Motors start and execute pre-flight check
ENGINES_DISARM	Stop motors
MCE_LANDING	perform landing
MCE_TAKEOFF	perform takeoff
–устаревшие–	
ENGINES_ARM	Start motors

## Get autopilot data

To get autopilot data **Sensors** class is used

---

```
class Sensors
```

```
    lpsPosition()
```

```
        Returns x, y, z
```

```
    lpsVelocity()
```

```
        Returns vx, vy, vz
```

```
    lpsYaw()
```

```
        Returns yaw
```

```
    orientation()
```

```
        Position data.
```

```
        Returns roll, pitch, azimuth
```

```
    altitude()
```

```
        Altitude data fro barometer.
```

```
        Returns altitude in meters
```

```
    range()
```

```
        Data from proximity sensors/
```

```
        Returns Returns proximity sensor distance. Several values.
```

```
    accel()
```

```
        Accelerometer data.
```

```
        Returns ax, ay, az
```

```
    gyro()
```

```
        Gyroscope data.
```

```
        Returns gx, gy, gz
```

```
    rc()
```

```
        RC transmitter data.
```

```
        Returns channel1, channel2, channel3, channel4, channel5, channel6,  
                channel7, channel8
```

### Examples

```
local lpsPosition = Sensors.lpsPosition
local lpsVelocity = Sensors.lpsVelocity
local lpsYaw = Sensors.lpsYaw
local orientation = Sensors.orientation
local range = Sensors.range
local accel = Sensors.accel
local gyro = Sensors.gyro
local rc = Sensors.rc

lpsX, lpsY, lpsZ = lpsPosition()
lpsVelX, lpsVelY, lpsVelZ = lpsVelocity()
yaw = lpsYaw()

roll, pitch, azimuth = orientation()

range1, range2, _,_, range3 = range()

ax, ay, az = accel()
gx, gy, gz = gyro()
aileron, _, _, _, _, _, ch8, = rc()
```

### Service functions of the script

```
function callback(event) -- is called when autopilot event is received
end
```

---

Autopilot events available:

Name	Description
ENGINES_STARTED	Engines started
COPTER_LANDED	Quadcopter performed landing
TAKEOFF_COMPLETE	Quadcopter has reached takeoff altitude
POINT_REACHED	Quadcopter has reached destination point
POINT_DECELERATION	Quadcopter starts decelerating near the point
LOW_VOLTAGE1	low battery voltage, return home mode
LOW_VOLTAGE2	low battery voltage, landing mode
SYNC_START	Synchronised start signal received from positioning system
SHOCK	Collision or hard vibration
CONTROL_FAIL	Pitch angle exceeded maximum value
ENGINE_FAIL	Engine fail

---

**Note:** Ev.ALTIITUDE\_REACHED ( quadcopter has reached destination point) event is no more used starting from autopilot version 1.5.6173.

---

### Script example

```
local boardNumber = boardNumber
local unpack = table.unpack
local points = {
    {-0.6, 0.3, 0.2},
    {0.6, 0.3, 0.2},
    {0, 0, 0.5},
    {0.6, -0.3, 0.2}
}

local curr_point = 1

local function nextPoint()
    if(#points >= curr_point) then
        ap.goToLocalPoint(unpack(points[curr_point]))
        curr_point = curr_point + 1
    else
        ap.push(Ev.MCE_LANDING)
    end
end

function callback(event)
    if(event == Ev.TAKEOFF_COMPLETE) then
        nextPoint()
    end
    if(event == Ev.POINT_REACHED) then
        nextPoint()
    end
end

local leds = Ledbar.new(1)
local blink = 0
leds:set(0,1,1,1)
timerBlink = Timer.new(1, function ()
    if(blink == 1) then
        blink = 0
    else
        blink = 1
    end
end)
```

(continues on next page)



(continued from previous page)

```
    end
    leds:set(0, blink, 0, 0)
end)
timerBlink:start()
ap.push(Ev.MCE_PREFLIGHT)
Timer.callLater(1, function() ap.push(Ev.MCE_TAKEOFF) end)
```

## Description for modules pin connectors

Pioneer\_Base\_v.1.0-v.1.1 autopilot controller pins

X1 (MC pin)	Function	Description
1	5V power (only with LiPo) 2A max.	
2	3.3V power 2A max.	
3 (PA12)	USART1_RTS	
4 (PA11)	USART1_CTS, TIM1_CH4	Module_OpenMV
5 (PA10)	USART1_RX, TIM1_CH3	Module_GPS, Module_USNav
6 (PA9)	USART1_TX, TIM1_CH2	Module_GPS, Module_USNav
7 (PA15)	SPI3_NSS, TIM2_CH1	Module_GPS, Module_OpenMV
8 (PC10)	SPI3_SCK	Module_GPS, Module_OpenMV
9 (PC11)	SPI3_MISO	Module_GPS, Module_OpenMV
10 (PB5)	SPI3_MOSI, TIM3_CH2	Module_GPS, Module_OpenMV
11	Земля	
12	Земля	

X2 (MC pin)	Function	Description
1	5V power (only with LiPo) 2A max.	
2	3.3V power 2A max.	
3 (PC2)	ADCx_IN12	
4 (PC3)	ADCx_IN13	
5 (PA1)	ADCx_IN1, TIM2_CH2, TIM5_CH2	Module_Cargo (упр. магнитом)
6 (PB7)	I2C1_SDA, TIM4_CH2	Module_ToF, Module_OpenMV
7 (PB6)	I2C1_SCL, TIM4_CH1	Module_ToF, Module_OpenMV
8 (PA0)	DATA WS2812B	Уровень 5 В. Module_LED
9	Земля	
10	Земля	

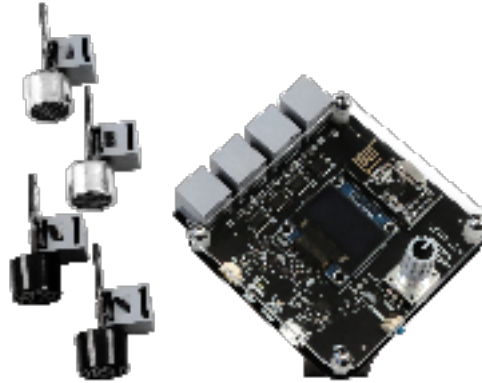
For Pioneer\_Base\_v.1.2:

X2 (MC pin)	Function	Description
3 (PA0)	USART4_TX, ADCx_IN0, TIM2_CH1, TIM5_CH1	Module_OpenMV
4 (PA1)	USART4_RX, ADCx_IN1, TIM2_CH2, TIM5_CH2	Module_OpenMV
5 (PC3)	ADCx_IN13, SPI2_MOSI	Module_Cargo (упр. магнитом)
8 (PC12)	DATA WS2812B	5 V level. Module_LED

---

## 1.6 Geoscan Locus indoor navigation system

The system allows to create a controlled flyzone with max. dimensions of 10x10x4 meters. Safe and precise quadcopter control is provided without using global navigational systems (GPS/GLONASS)



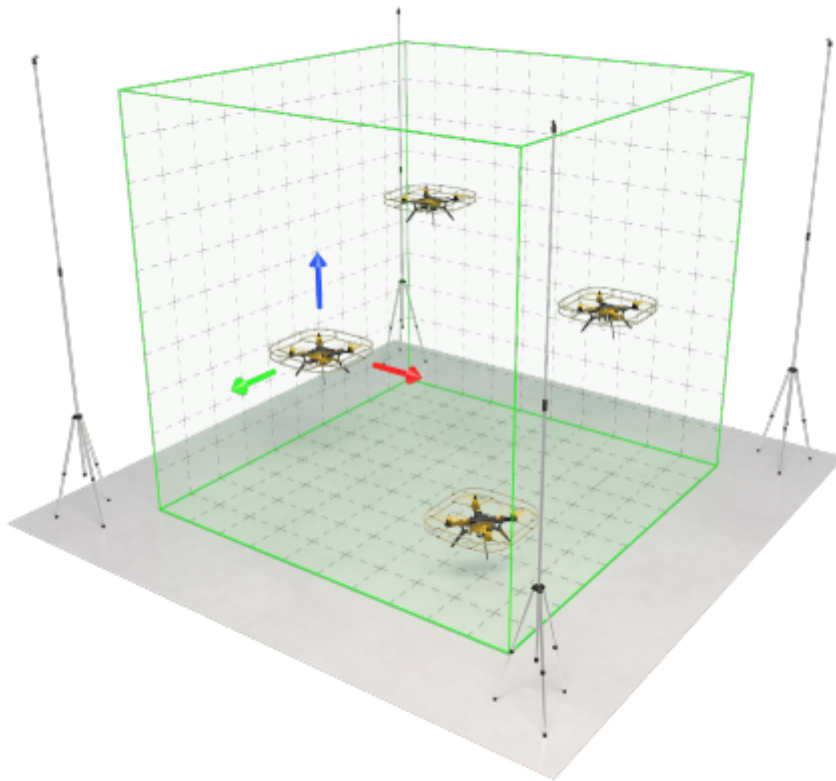
[Download LPS setting manual](#)

The system includes:

- control module;
- 4 ultrasonic transducers;
- onboard module;
- connection cables;
- LPS software to sync with PC ([download latest version](#));

---

### 1.6.1 Flyzone setting



Mount and connect the sensors in future flyzone. Transducers should be placed in the top corners of it in a way that they point in its centre. These conditions should be considered:

- minimum installation height - 2 m
- minimal distance -3 m

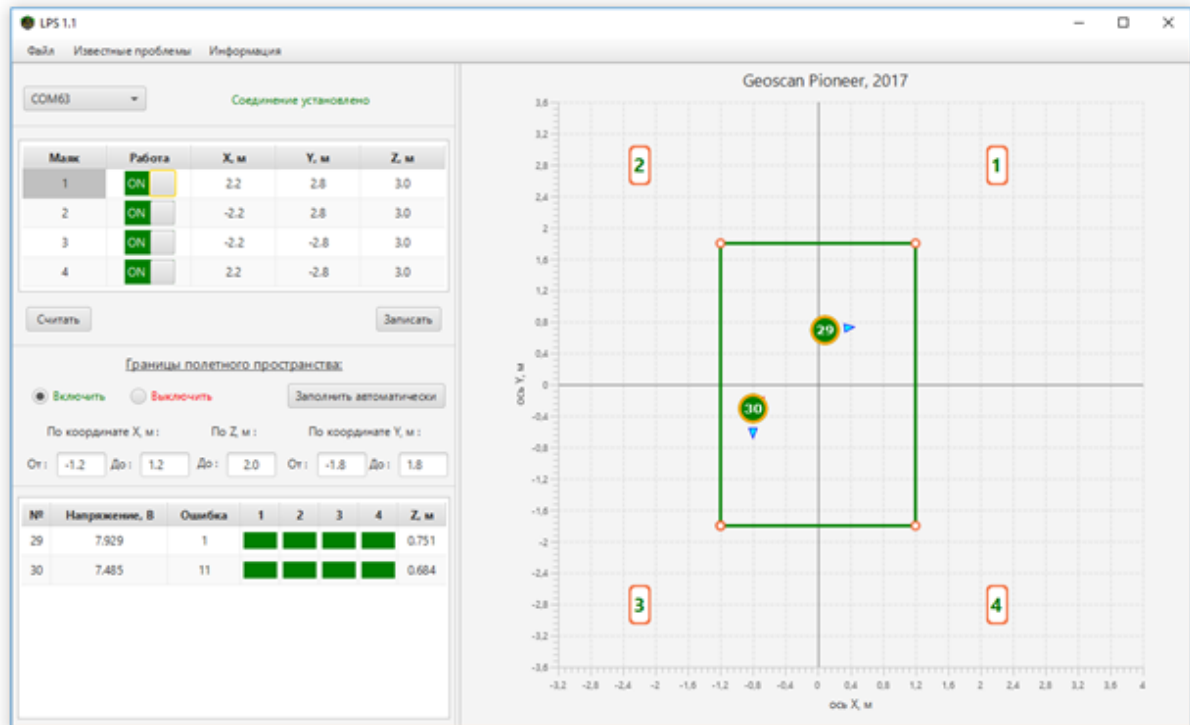
---

**Note:** System works better in rooms with soft floor cover

---

Using established cables, each sensor is connected to the corresponding socket of control module. The module itself should be placed outside of the zone and connected to USB port of a PC

The control and onboard modules are communicating via radio channel. Measure the distance between the transducers and their installation height. A square is the most convenient and easy to set form of flight zone.



LPS user interface is shown on the picture. The program is used to calibrate the positioning system. Fill in the empty beacon coordinates:

- The distance (in meters) between 1 and 2 transducers is divided by 2. The “received value should be written in X string with a “minus” for №2 and №3 “beacons and without “minus” for №1 and №4.
- The distance between 2 and 3 transducers is also divided by 2 and is put in “the Y string with “minus” for №3 and №4 beacons and without it for №1 “and №2.”
- String Z is for each beacon’s height above floor level.

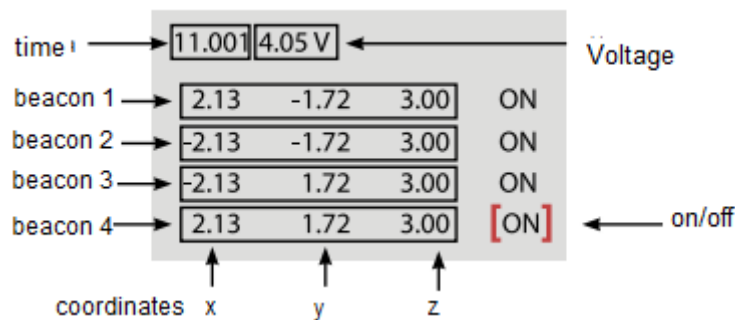
With all coordinates being set, the flyzone is ready and established in the right window with a green line. By default, its corners are 1 m away from the actual beacons.

**Note:** You can also set the flyzone parameters directly on the control module.

Push the start button next to spin selector once to turn it on. A green “”status” and white “power” light should appear. If they didn’t, check the battery charge.

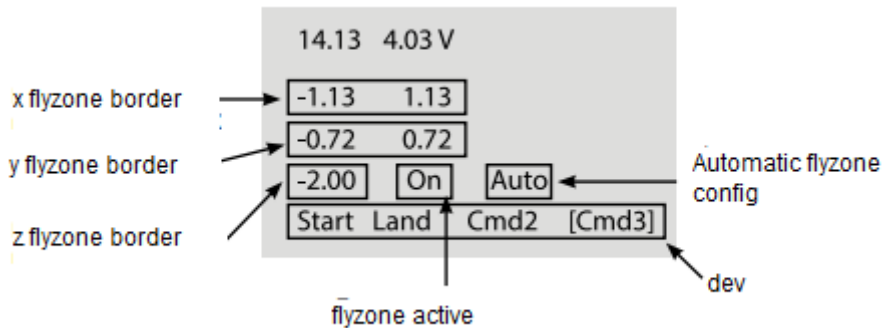
The top line of the screen shows a turn-on timer and battery voltage. If it drops lower than 3 V, a “status” light will start to blink. To charge it, just connect the module to PC or a 5V2A charger.

The coordinates are established in a same way as on LPS screen.



To select a menu item, twist the selector and push it like a button. Change the desired value by twisting it and push once again to confirm the changes and go back to the menu.

The second menu screen allows you to set indents from the flyzone border or toggle “Auto” mode for automatic setting. You can also turn the flyzone limits off.



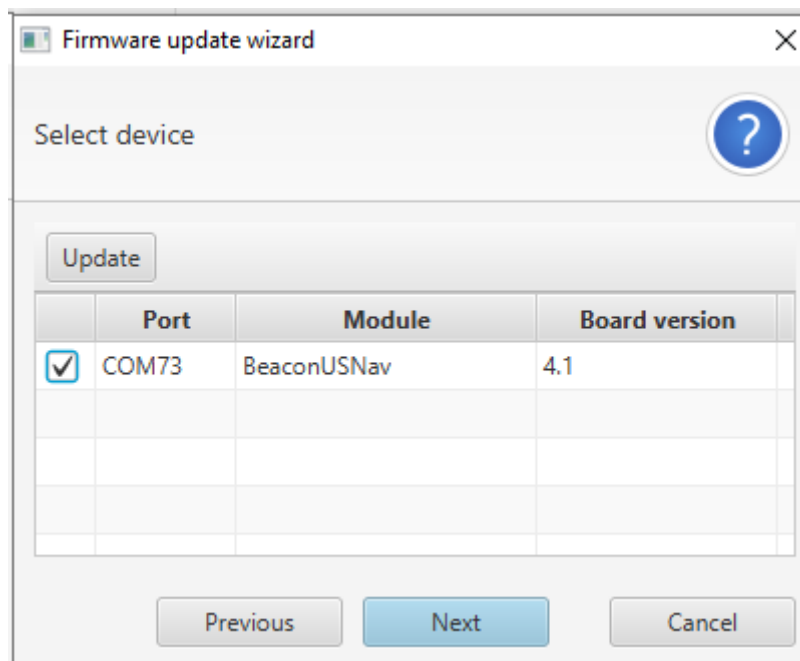
**Note:** You don't need PC for positioning system to operate. Just turn the control module on and place a Pioneer, equipped with onboard module, inside the flyzone

## 1.6.2 Indoor flight

To perform pre-programmed flight, connect battery, switch SwB to bottom position, and push start button on Pioneer base plate. The drone will begin script execution in 5 seconds. For manual flight control, switch SwB in middle position. Pioneer will still maintain its position with Locus navigation system, so the flight will be smooth and easy to control.

## 1.6.3 Navigation system firmware update

To update the system firmware you will need Pioneer Station installed on your PC. Select Firmware update in the menu and follow the instructions on the screen. Press and hold 'Menu' button on the control unit and connect it to your PC via USB cable. Select BeaconUSNav on the list and click 'next'



It is recommended to choose default firmware source. Flashing firmware may take up to 5 minutes, do

---

not disconnect the control unit until it is finished.

## 1.7 FAQ

**The quadcopter is falling to its side while takeoff** - Check if motors and props are installed correctly according to the frame markers. White and black motors must be installed diagonally.

**The quadcopter doesn't take off while props are spinning** - Check if props are installed correctly. The "R" marks should be on the motors with black caps.

**The motor doesn't spin after arm command** - Check the connectors between motor and the frame.

**The quadcopter doesn't react on arming signal** - Check if the control unit is set up correctly and re-bind the receiver.

**The quadcopter doesn't react properly on controls** - check the LiPo and control unit battery charge.

**The quadcopter doesn't react on LiPo connection** - check the structural integrity of the battery and its voltage level, which should not be less than 6.6 V

**Quadcopter vibrates violently** - Check the frame assembly, base board connection. A propeller defect is also a possible issue.

**Empty ports list when connected to Pioneer Station** - check USB cable; download and install [virtual port driver](#) for your OS

**Empty window in Pioneer Station when opening parameters from file** - close the app, and activate run\_extra.bat file in Pioneer Station directory

**Empty window in Pioneer Station when opening parameters from file** - close the app, and activate run\_extra.bat file in Pioneer Station directory

## 1.8 Changelog

### 1.8.1 20/06/2019

1. Pioneer Station update [version 1.10.0](#)
  - Added new protocol (1.5) support for new autopilot integration
  - Now you can download logs in 2 ways:
    - Using Payload (slow). You need bootloader capable of log downloading.
    - Using Plazlink protocol (new faster method). You need firmware with 1.5 protocol support.
  - New Lua script examples for Pioneer, with extended comments and optimised structure
  - New LED module scrips examples
  - Now you can inspect autopilot logs ( "work with logs" section in MENU)
2. Autopilot firmware update (1.5.6173 version)
  - Protocol update to 1.5
  - Log download function
  - Updated takeoff algorithm
  - Fixed bug when 2 events were generated during takeoff. Now it's just **Ev.TAKEOFF\_COMPLETE**. **ALTITUDE\_REACHED** event is no longer used.
  - Added engines start check during programmed flight

---

## 1.8.2 11/02/2019

1. Pioneer Station update [version 1.9.1](#)
  - Fixed code editor and telemetry bug when resizing application's window
  - Fixed autopilot version indication in telemetry table
  - Fixed update lps module notification
  - Added link to changelog menu-documentation

## 1.8.3 1/02/2019

1. Pioneer Station update [version 1.9.0](#)
  - Added accelerometer calibration
  - Added compass calibration
  - Added installed on a pioneer module version output. if its firmware is outdated its version will be highlighted in a table and a window inviting to update it will pop-up
  - Added English localisation, when installing this application you will now have an option to choose between Russian and English
  - Fixed an error when switching between LPS/GPS/OPT
  - Added a fix to blank window with difference between parameters in file and in autopilot
2. Autopilot firmware update (version 1.4.4922)
  - Updated direction hold algorithm
  - Fixed an error with autopilot reboot when launching several timers at once
  - Changed barometer data output, we no longer output raw data but filtered by AP data for tests on Robofest
3. Module USNav firmware update (version 2.1)
  - Updated data acquisition algorithm from transducers

Данное руководство содержит информацию в объеме, достаточном для полного освоения комплекса Геоскан “Пионер”, его сборки, пилотирования и программирования.