PROJECT REPORT

PlutoX Programmable Drone

Rajula Vineet Reddy (IMT2014045)

Vishal Gattani (IMT2015508)

11.05.2019

ToC

INTRODUCTION

In this project, we worked on programming the PlutoX drone to achieve results such as collision avoidance, following a target and getting sensor data mounted on top of the drone. The sensors used to achieve the above said behavior are the IR sensors (V53LOX) and analog sensors to determine the temperature and pressure when the drone is in flight mode.

This report summarizes the various scenarios and difficulties faced during the implementation phases of the project.

Pluto Controller Android/ IOS App

The PlutoX drone comes with an android app 'Pluto controller'. The android app provides the following functionalities

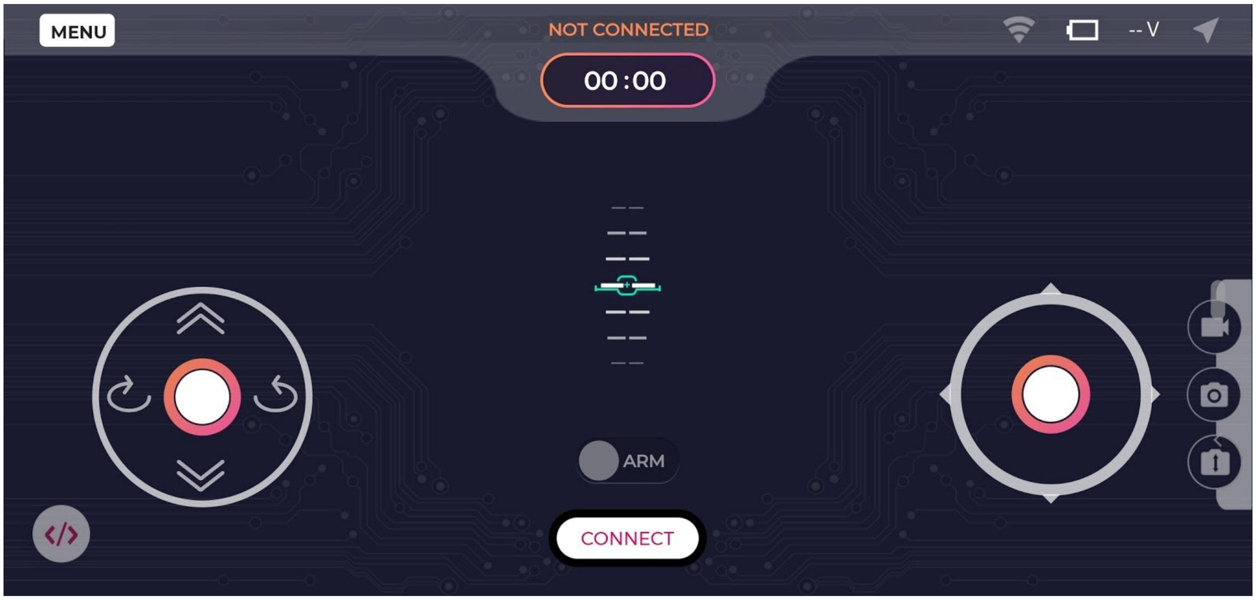
1. Calibrate the drift of the drone
2. Calibrating accelerometer & magnetometer
3. Drone Arming & take off
4. Controlling the drone movement
5. Flashing the drone with the firmware
6. Developer mode

Installation

The android app is available on the playstore under the name "Pluto Controller".

https://play.google.com/store/apps/details?id=com.drona.controller&hl=en_IN

<App versions & bugs>

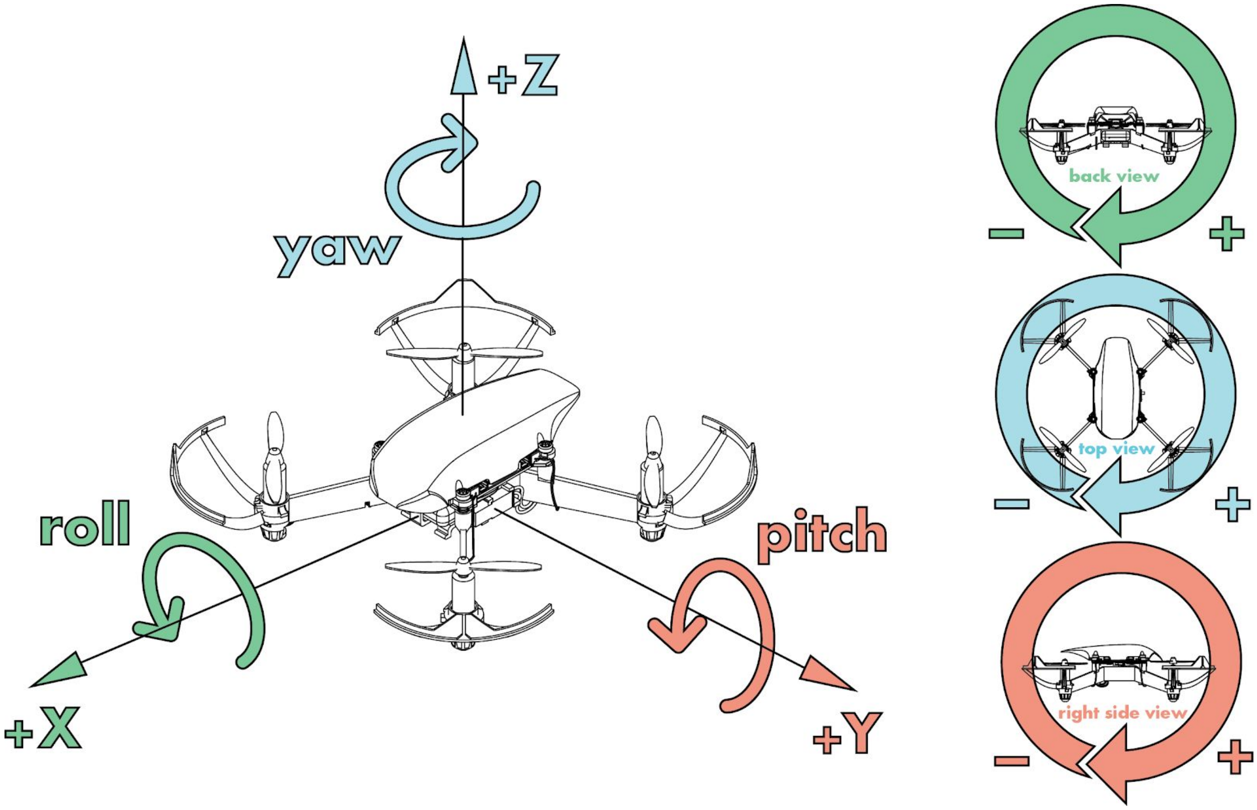


Calibrating Drift

In most of the cases, when you try to start the drone and fly it with/ without the android app the drone keeps drifting to one or two directions. To stabilize it, the drone drift needs to be calibrated. In the android app, it can be calibrated under the ‘Drone Settings’.

The drone works on three different axes as shown in the image below

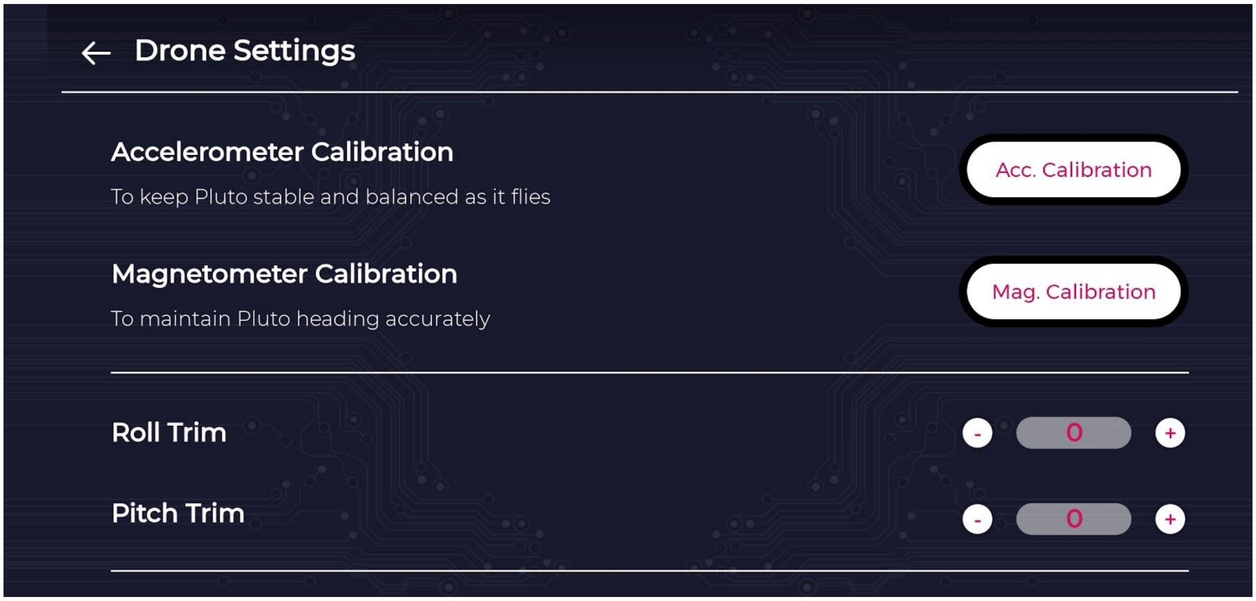
1. Roll
2. Pitch
3. Yaw



There are two parameters ‘Roll’ & ‘Pitch’ that can be changed to calibrate the drift. Refer to the below table

Drift in direction	Set input
Forward	Negative Pitch
Backward	Positive Pitch
Left	Negative Roll
Right	Positive Roll

These settings can be calibrated in the ‘Drone Settings’ section of the app. It is advised to change the values in multiples on 5 first and then it can be calibrated more accurately after finding a upper limit.



← Drone Settings

Accelerometer Calibration

To keep Pluto stable and balanced as it flies

Magnetometer Calibration

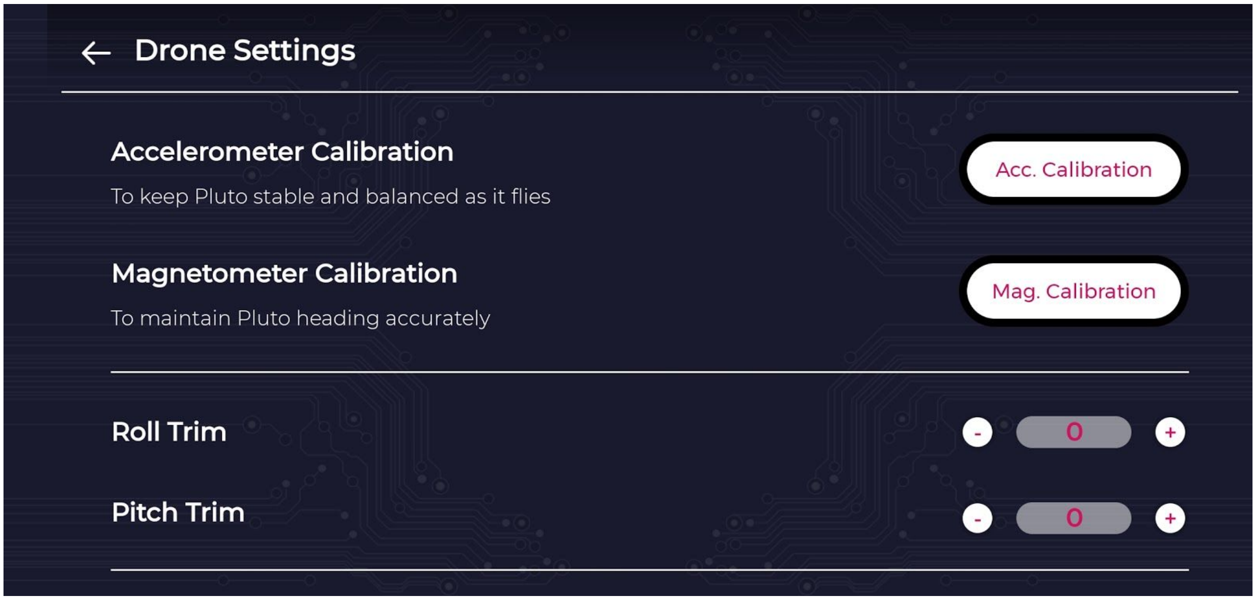
To maintain Pluto heading accurately

Roll Trim - 0 +

Pitch Trim - 0 +

Calibrating Accelerometer & Magnetometer

In some cases, the drone does not fly properly & will keep moving in random directions. In such cases, the accelerometer and magnetometer have to be calibrated. While in other cases, the android app prompts to calibrate. These can be calibrated in the 'Drone Settings' section. For calibrating the accelerometer, 'calibrate accelerometer' button should be pressed & the drone has to be just placed on a flat surface. For calibrating the magnetometer, 'calibrate accelerometer' button should be pressed & the drone has to be rotated in all the three axis namely pitch, yaw and roll as shown in the 'Calibrating Drift' section.



← Drone Settings

Accelerometer Calibration

To keep Pluto stable and balanced as it flies

Magnetometer Calibration

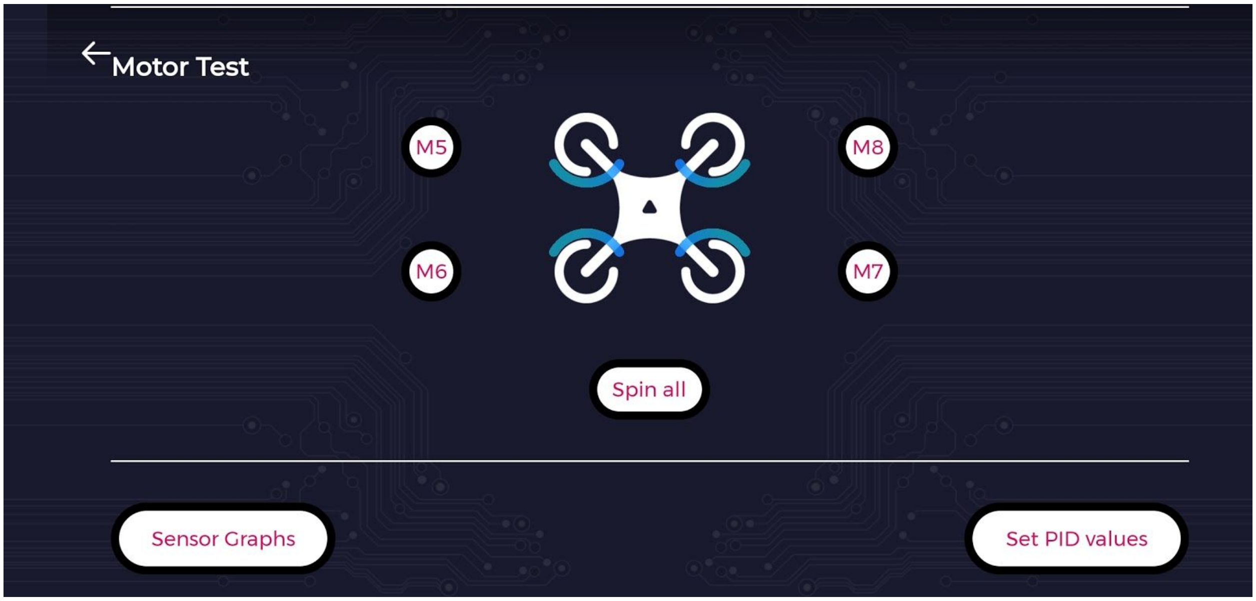
To maintain Pluto heading accurately

Roll Trim - 0 +

Pitch Trim - 0 +

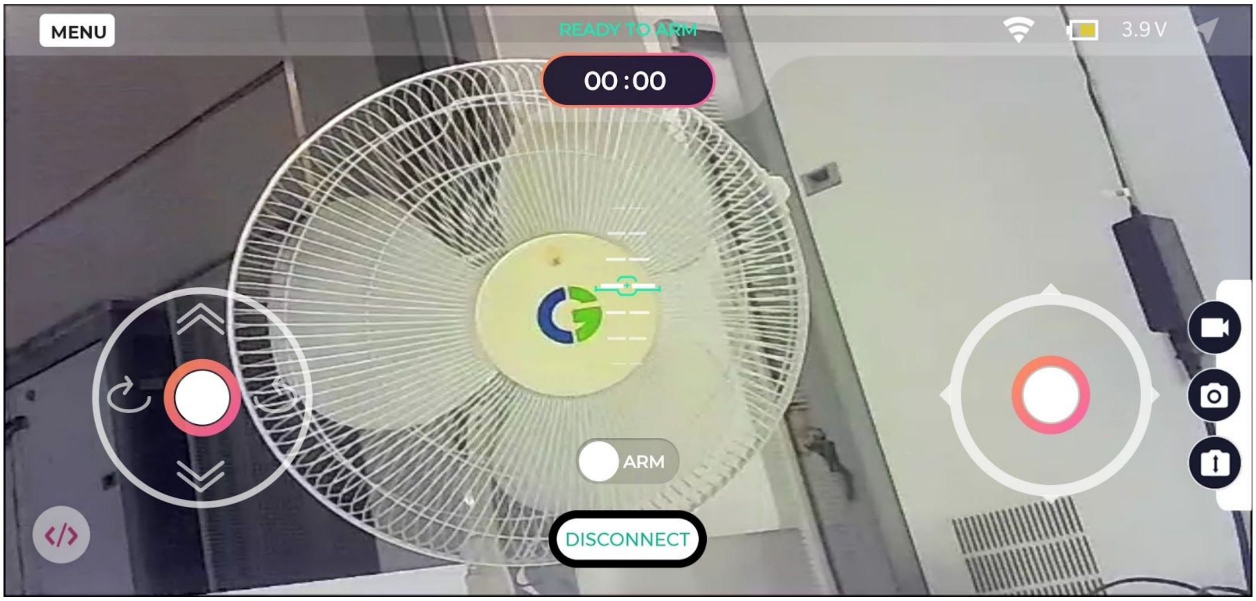
Testing PlutoX connection

The connection to plutoX from the android app can be tested by running the propeller motors. In the ‘Drone Settings’ section, after scrolling down a bit, a model drone image can be seen. By clicking on each of the motors, the propeller connected to that motor can be spinned. All the motors can be run at the same time, by clicking on the ‘Spin all’ button. This feature is useful to verify connection with the drone.



Camera Feed

As of now, the only way to access the camera feed from the on-board camera is through the android app. When the camera is put on the drone and connected correctly, when the android app is opened the feed comes in the background of the home screen of the app. Using the three buttons on the right, video can be recorded, image can be captured and the feed direction can be flipped respectively.

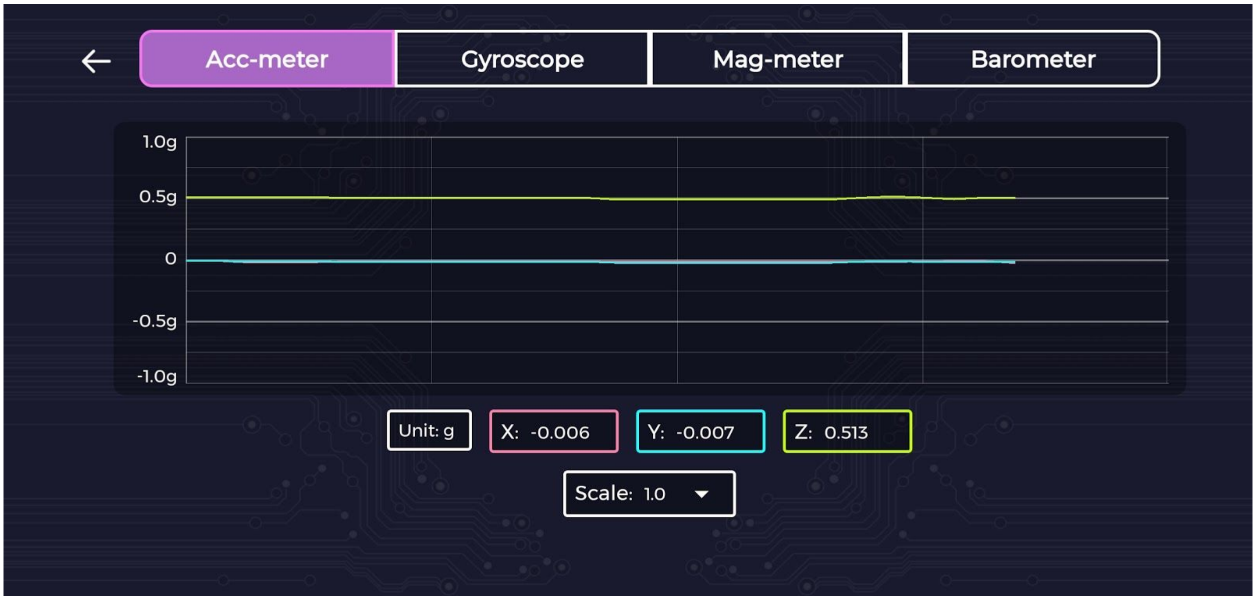


Sensor Graphs

The plutox has the following sensors on board (by default)

1. Accelerometer
2. Gyroscope
3. Magnetometer
4. Barometer

These sensors can be monitored through graphs from the android app it self, by clicking on the ‘Sensor Graphs’ in ‘Drone Settings’ section.



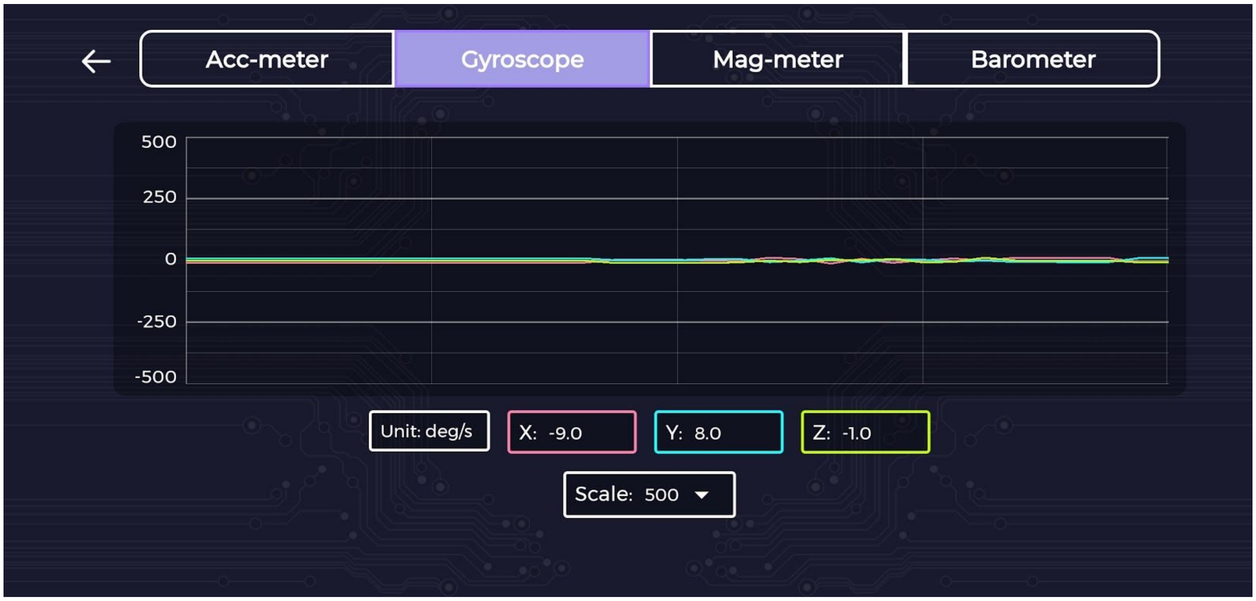
Unit: g

X: -0.006

Y: -0.007

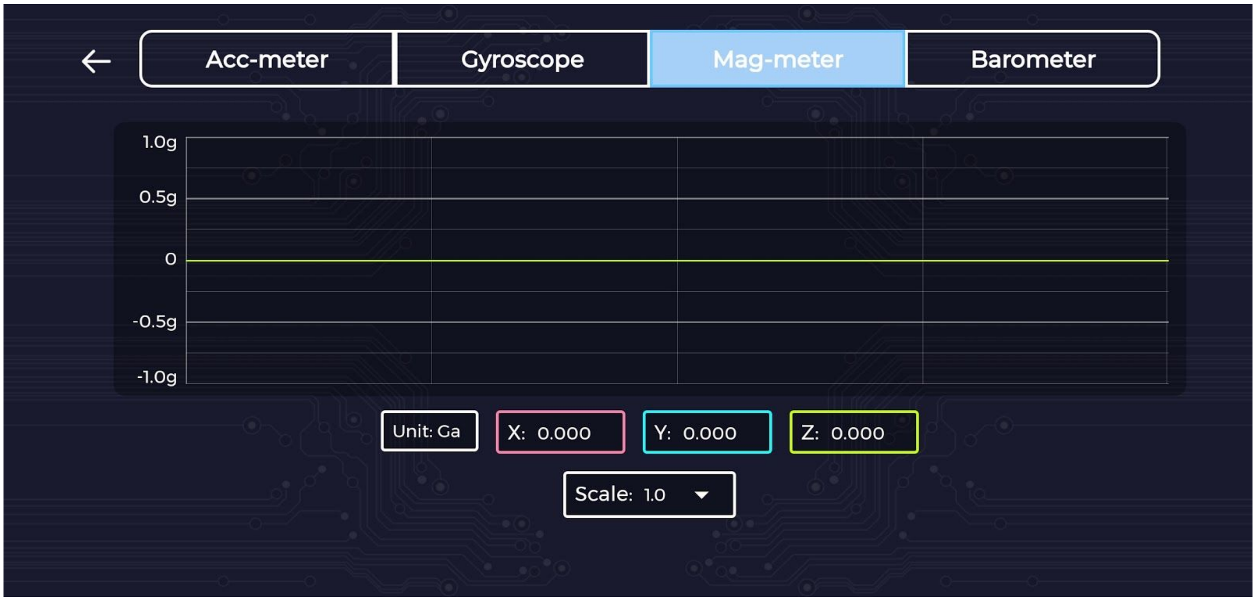
Z: 0.513

Scale: 1.0



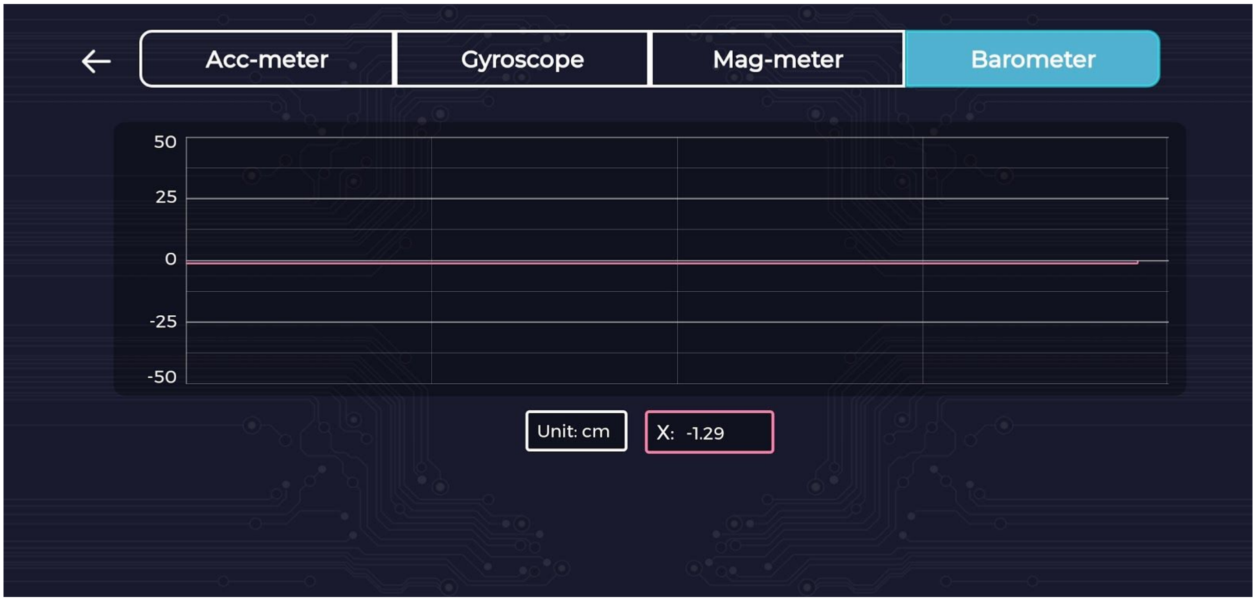
Unit: deg/s X: -9.0 Y: 8.0 Z: -1.0

Scale: 500



Unit: Ga X: 0.000 Y: 0.000 Z: 0.000

Scale: 1.0

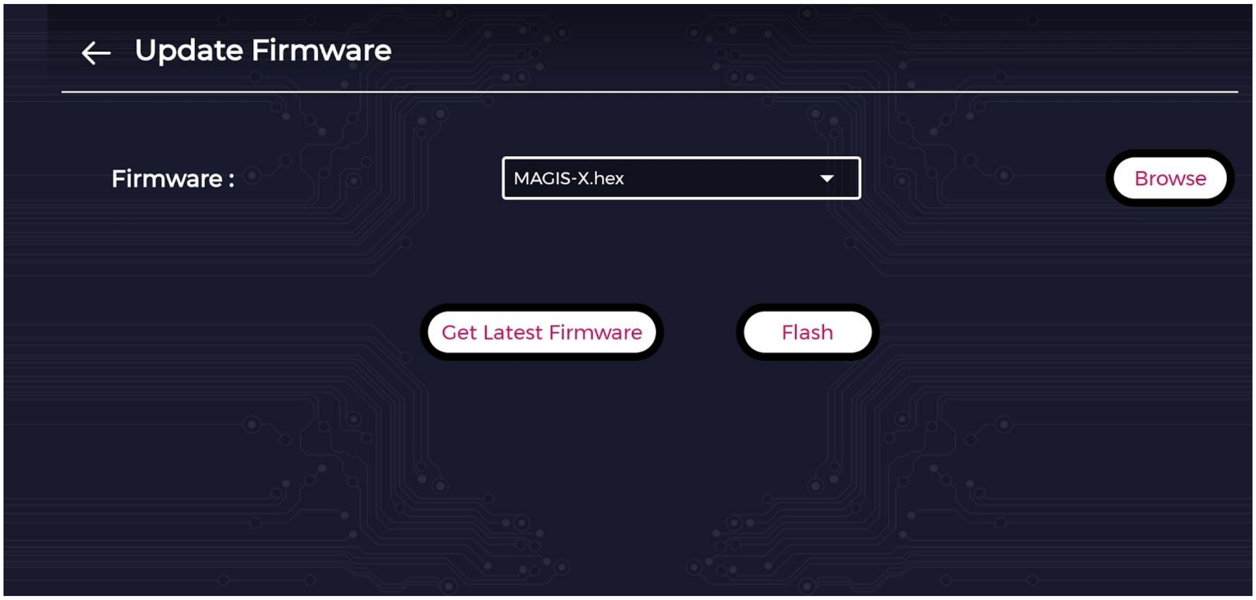


Unit: cm

X: -1.29

Flashing default firmware

At times when the drone is not functioning properly, or when there are issues with communication, or to reset the drone, it needs to flashed with the default firmware. The same can be achieved using the android app, from *menu* > *Update Firmware*. For the first time, by clicking on latest firmware the latest version for Pluto X will be downloaded from the web. Or if already downloaded, it can be browsed using the *browse* button. Once the firmware is selected (from the dropdown list), by clicking on *Flash*. This will sort of reset the drone with the original firmware. All the codes flashed on top of the default firmware will be erased by doing this.



Firmware : MAGIS-X.hex

CYGNUS IDE v1.0.6

Installation

To install the IDE required to program PlutoX, download the IDE from the link: https://www.dronaaviation.com/support/

Extract in the folder required and run 'cygnus.exe'.

There are binaries supported for Linux as well as Windows for this version.

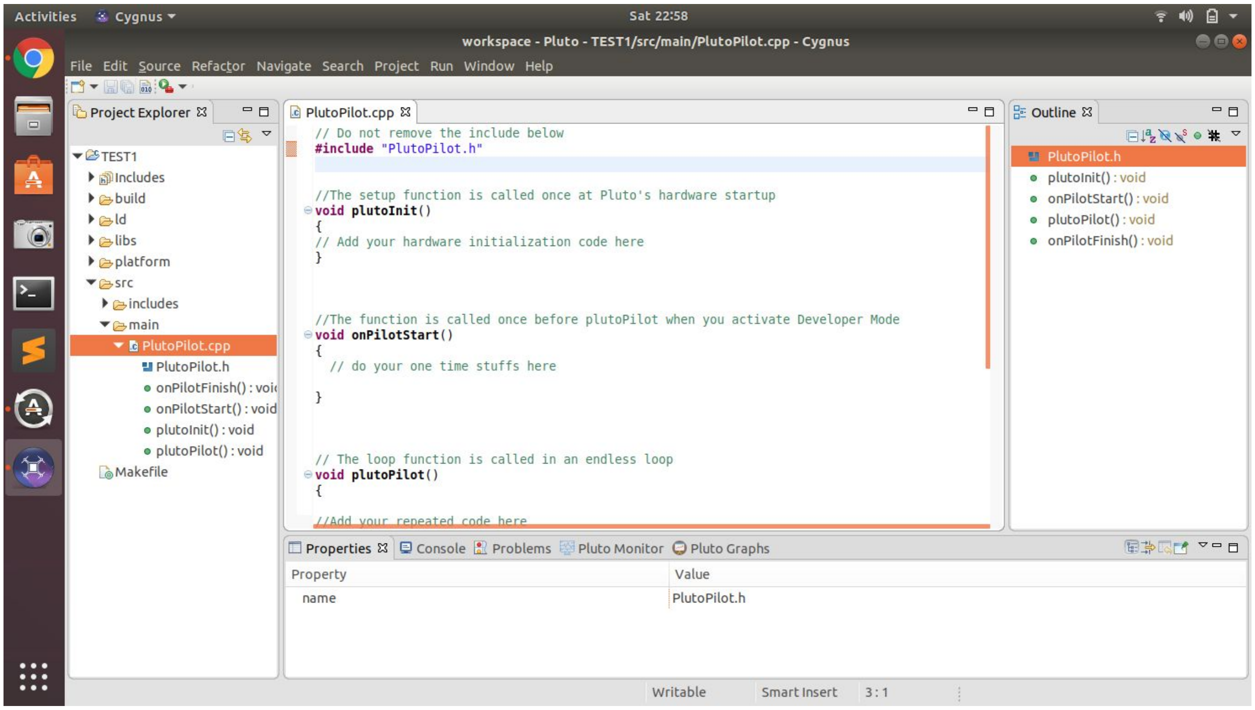
Creating a Project

Click on File -> New -> Pluto-X Project and enter the name of the project.

The resulting project folder is generated consisting of the libraries and the Makefile that is required by the project for compilation and flashing to the drone purposes.

The different functions that a PlutoX program has are as follows:

1. void plutoInit()
2. void onPilotStart()
3. void plutoPilot()
4. void onPilotFinish()



```
// Do not remove the include below
#include "PlutoPilot.h"


//The setup function is called once at Pluto's hardware startup
void plutoInit()
{
// Add your hardware initialization code here
}



//The function is called once before plutoPilot when you activate Developer Mode
void onPilotStart()
{
  // do your one time stuffs here

}



// The loop function is called in an endless loop
void plutoPilot()
{

//Add your repeated code here
```

Property	Value
name	PlutoPilot.h

plutoInit()

This function is executed immediately after the drone powers up. This function is particularly useful for initializing hardware.

onPilotStart()

This function is executed once, after user the user turns the developer mode on. This function can be useful for initialising some variables

plutoPilot()

This function is executed in loop, along with drones internal primary stabilization code. By default this loop runs every 3.5ms. Using APIs, you can modify this loop frequency. This will not affect the drones internal stabilization loop.

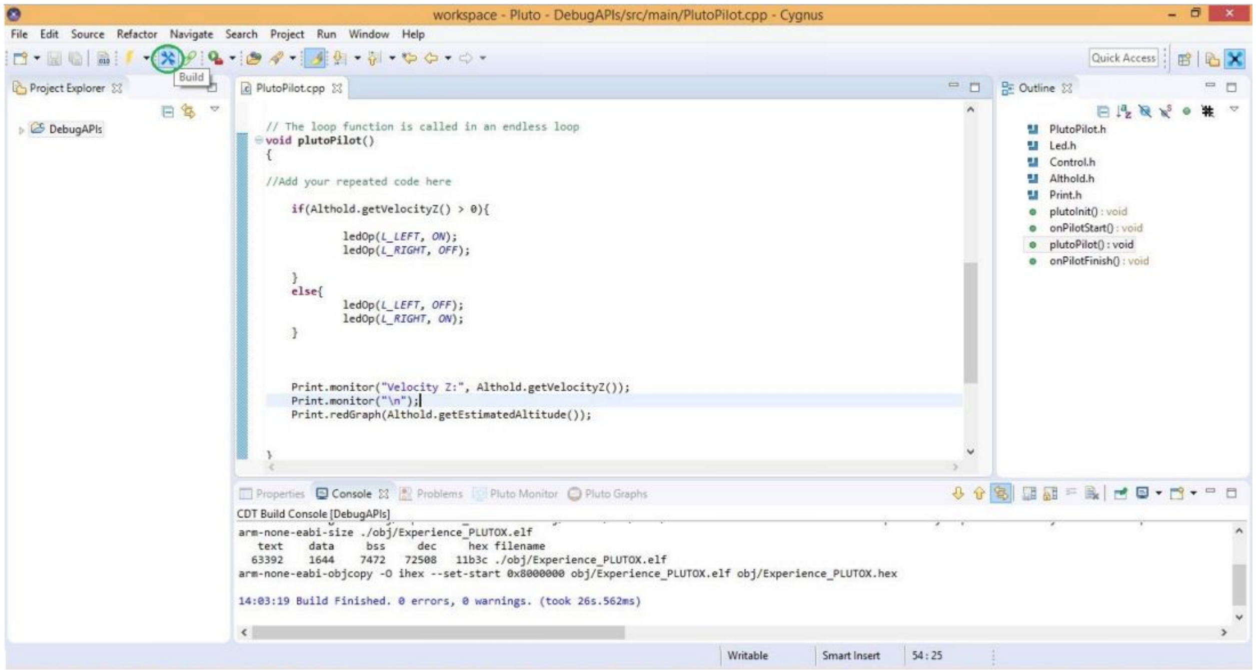
onPilotFinish()

This function is executed once, after the user turns off the developer mode.

Building a Project

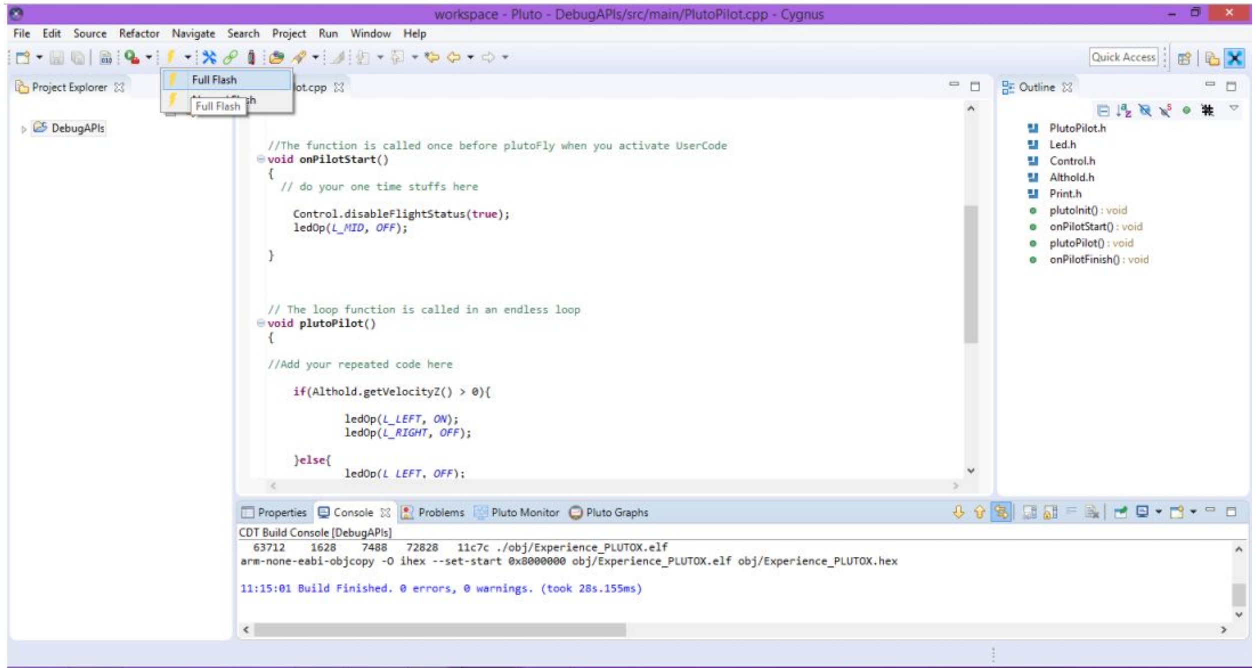
Once the program is ready to be executed by the drone, the first step is to build the project which is just a basic compiling of the code.

Click on the blue icon that will build and output any errors in the code. Once the code is built, a command prompt stating "Project Built Successfully" appears.



```
// The loop function is called in an endless loop
void plutoPilot()
{

//Add your repeated code here

    if(Althold.getVelocityZ() > 0){

            ledOp(L_LEFT, ON);
            ledOp(L_RIGHT, OFF);

    }
    else{
            ledOp(L_LEFT, OFF);
            ledOp(L_RIGHT, ON);
    }



    Print.monitor("Velocity Z:", Althold.getVelocityZ());
    Print.monitor("\n");
    Print.redGraph(Althold.getEstimatedAltitude());


}
```

```
arm-none-eabi-size ./obj/Experience_PLUTOX.elf
   text    data     bss     dec     hex filename
  63392    1644    7472   72508   11b3c ./obj/Experience_PLUTOX.elf
arm-none-eabi-objcopy -O ihex --set-start 0x8000000 obj/Experience_PLUTOX.elf obj/Experience_PLUTOX.hex

14:03:19 Build Finished. 0 errors, 0 warnings. (took 26s.562ms)
```



```
//The function is called once before plutoFly when you activate UserCode
void onPilotStart()
{
  // do your one time stuffs here

    Control.disableFlightStatus(true);
    ledOp(L_MID, OFF);

}



// The loop function is called in an endless loop
void plutoPilot()
{

//Add your repeated code here

    if(Althold.getVelocityZ() > 0){

            ledOp(L_LEFT, ON);
            ledOp(L_RIGHT, OFF);

    }else{
            ledOp(L_LEFT, OFF);
```

```
  63712    1628    7488   72828   11c7c ./obj/Experience_PLUTOX.elf
arm-none-eabi-objcopy -O ihex --set-start 0x8000000 obj/Experience_PLUTOX.elf obj/Experience_PLUTOX.hex

11:15:01 Build Finished. 0 errors, 0 warnings. (took 28s.155ms)
```

To flash the code to the PlutoX drone, follow the following steps:

1. Ensure that the Android Application is **not connected** to the Drone.
2. Connect your computer to the WiFi of the Drone’s SSID.
3. Click on the ‘green’ link button on the top left to establish Pluto and Cygnus IDE connection (Chain will turn into red colour if connected). This is required to flash to the firmware of the drone.
4. Once clicked, the color of the button turns to ‘red’ if it’s connected to the WiFi of the drone.
5. After connecting Pluto WiFi to computer to flash the code select Flash command from toolbar. Flash command burn the code from the project you have selected or it burns the code from the project of current file opened in editor by default.

There are two types of Flash -

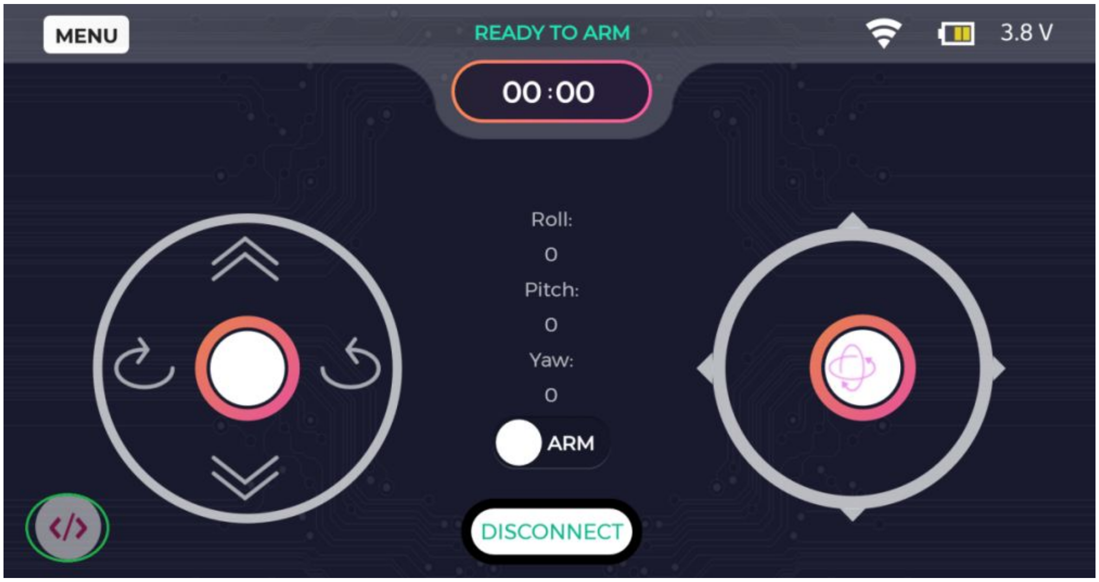
(i) Full Flash- burn the code with erasing ROM data like accelerometer and magnetometer calibration.

(ii) Normal Flash: burn code without erasing ROM data like sensor calibration.

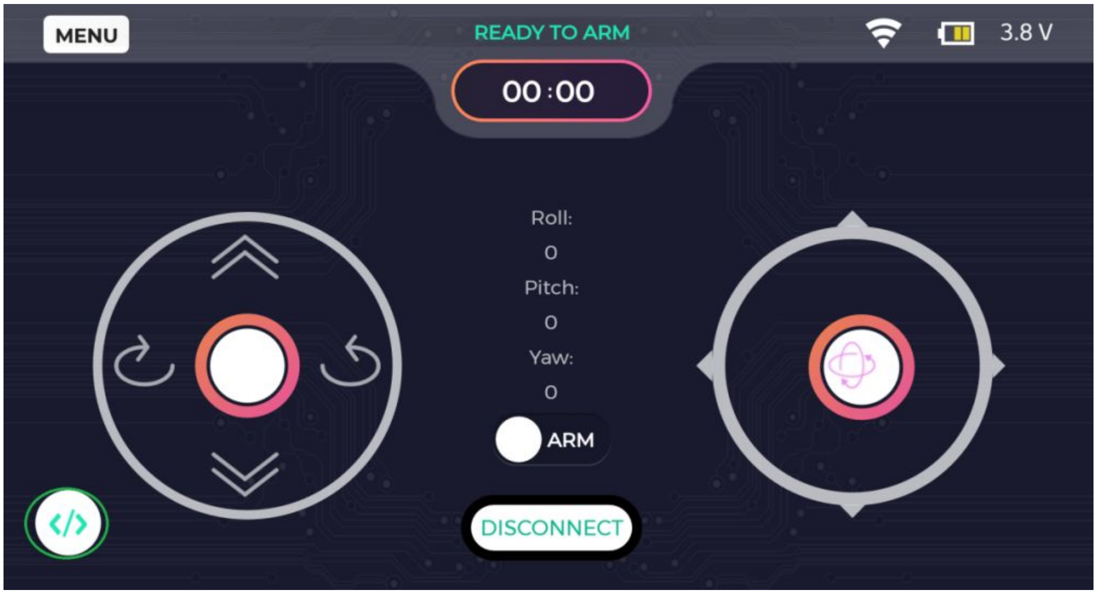
It is recommended to use Full Flash when programming first time and make sure you calibrate accelerometer and magnetometer after Full Flash. Onwards you can use Normal Flash, If you observe any weird behaviour of Pluto after Normal Flash, prefer Full Flashing it again.

Running the code

When you flash your program on Pluto it doesn't directly starts executing, it's in default mode. The User code execution starts when the drone is in developer mode. The developer mode can be controlled using the Pluto Controller app using the button </>. Please note that this button is accessible only when we choose "I am a Developer" from user profile in the App.



To run your program you need to activate Developer Mode using Pluto Controller Android or iOS application.



Once you have activated Developer Mode, the drone will execute the code running in the loop.

When Developer Mode is deactivated, it will go back to its default mode.

The API documentation for v1.0.6 is available at https://drive.google.com/open?id=1wtU-nnGNTPaOsXZ0_GVQ2klOFnSw--eA

***Note:* There are two different versions of CygnusIDE that we have worked on to support different functionalities. The reason for this is because the new firmware supports Breakout Board API functionalities regarding the ADC pins and the GPIO pins.The only difference between the two versions of Cygnus build are the function names and their implementations. The programs written with Cygnus v1.0.6 are not explained here, but are included on the github repository. The same programs are written with Cygnus v2.0.0 and they are included in the further sections**

CYGNUS IDE v2.0.0

Installation

To install the IDE required to program PlutoX, the new Cygnus build is available at the following link:

https://drive.google.com/file/d/1oFHE1I-wxGi8gmjj3N78j8dfCduer1L6/view

Extract in the folder required and run ‘cygnus.exe’.

***Note:* The installation is supported only on *Windows*.**

The different functions that a PlutoX program in this build are as follows:

- plutoInit()
- onLoopStart()
- plutoLoop()
- onLoopStop()

The functionalities of these functions are same as that of the previous build.

However, the implementation of the in-built functions are different.

The API documentation for v2.0.0 is available at https://docs.google.com/document/d/1qUzQ2eYyPjvnZ2zWgFSVFsRz8hdydEv4QQBmXz5xt9M/edit

***Note:* There are two different versions of CygnusIDE that we have worked on to support different functionalities. The following code snippets are compatible with the new build of Cygnus which can be downloaded by the link shown above. (Cygnus 2.0 on Windows)**

Monitoring Graphical Data

PlutoX has functions which can help monitor values of the sensors that come in-built such as the accelerometer on the IDE using the **Pluto Monitor**. At any given time, three graphs can be observed using the code.

```
// Do not remove the include below
#include "PlutoPilot.h"
#include "Sensor.h"
#include "User.h"
#include "Utils.h"

//The setup function is called once at Pluto's hardware startup
void plutoInit()
{
// Add your hardware initialization code here
      setUserLoopFrequency(1000);
}

//The function is called once before plutoLoop when you activate Developer
Mode
void onLoopStart()
{
  // do your one time stuff here
}

// The loop function is called in an endless loop
void plutoLoop()
{
//Add your repeated code here
      int32_t pressure, temp;
      pressure = Barometer.get(PRESSURE)/100;
      temp = Barometer.get(TEMPERATURE)/100 - 7;
      Monitor.println("Pressure " , pressure, 4);
      Monitor.println("Temperature " , temp, 4);
      Graph.red(pressure, 4);
      Graph.green(temp, 4);
}

//The function is called once after plutoLoop when you deactivate Developer
Mode
```

```
void onLoopFinish()
{
// do your cleanup stuffs here
}
```

Back Flip

```
// Do not remove the include below
#include "PlutoPilot.h"
#include "XRanging.h"
#include "Utils.h"
#include "Peripheral.h"
#include "Sensor.h"
#include "User.h"
#include "Control.h"


Interval T2;
//The setup function is called once at Pluto's hardware startup
void plutoInit()
{
// Add your hardware initialization code here
}

//The function is called once before plutoPilot when you activate Developer Mode
void onLoopStart()
{
      Command.takeOff(300);
      FlightMode.set(ATLTITUDEHOLD);
      T2.reset();
}

// The loop function is called in an endless loop
void plutoLoop()
{
//Add your repeated code here
      bool flag;
      flag = T2.set(2000,1);
      if(flag)
      {
            Command.flip(BACK_FLIP);
            Monitor.println("Flipping" );
      }
}
//The function is called once after plutoPilot when you deactivate
Developer Mode
void onLoopFinish()
{
       // do your cleanup stuffs here
}
```

Collision avoiding using X-Ranging Sensors

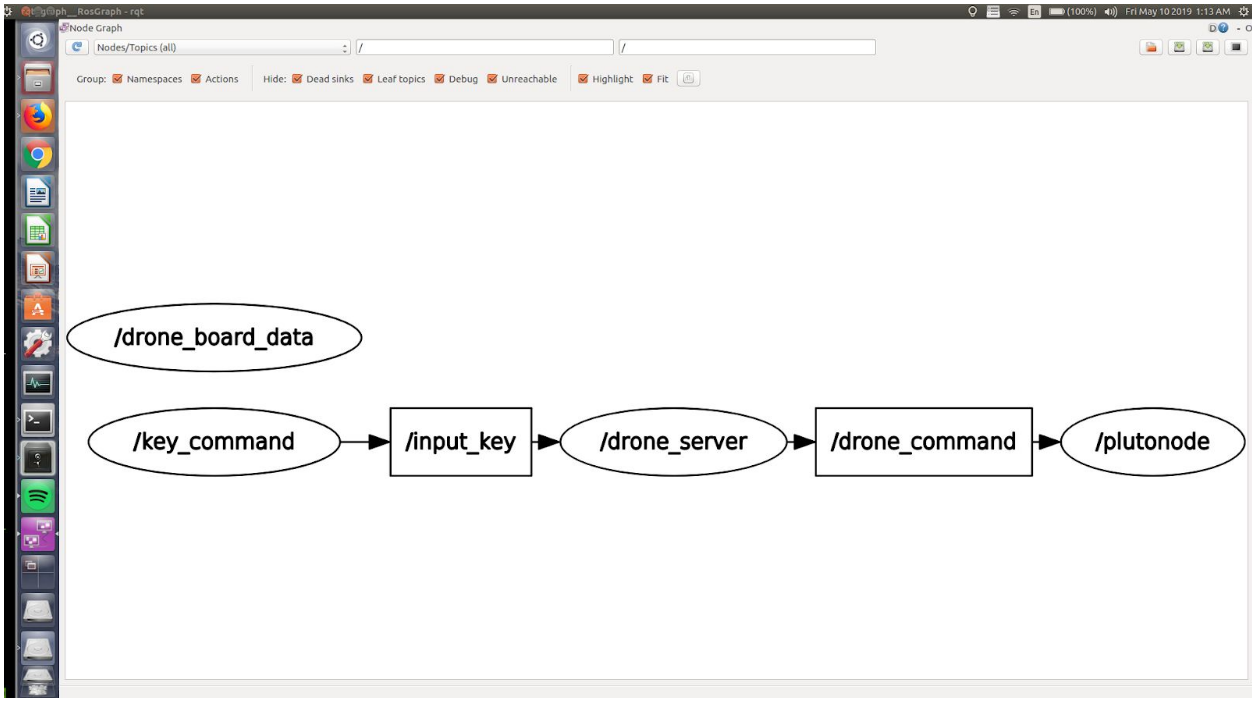
```
// Do not remove the include below
#include "PlutoPilot.h"
#include "XRanging.h"
#include "Utils.h"
#include "Peripheral.h"
#include "Sensor.h"
#include "User.h"
#include "Control.h"


Interval T2;
//The setup function is called once at Pluto's hardware startup
void plutoInit()
{
      XRanging.init(LEFT);
      XRanging.init(RIGHT);
}

//The function is called once before plutoPilot when you activate Developer
Mode
void onLoopStart()
{
      Command.takeOff(250);
      FlightMode.set(ATLTITUDEHOLD);
}

// The loop function is called in an endless loop
void plutoLoop()
{
//Add your repeated code here
      int val = XRanging.getRange(LEFT);
      int val2 = XRanging.getRange(RIGHT);

      if(val >= 0 && val <= 200)
      {
            Monitor.println("LEFT ",val);
            RcCommand.set(RC_ROLL,1550);
      }

      if(val2 >= 0 && val2 <= 200)
      {
            Monitor.println("RIGHT ",val2);
            RcCommand.set(RC_ROLL,1450);
      }

      bool flag = T2.set(2000,1);
      if(flag){
            val = -100;
            RcCommand.set(RC_ROLL,1500);
      }
}

//The function is called once after plutoPilot when you deactivate
Developer Mode
void onLoopFinish()
{
       // do your cleanup stuffs here
//     Xshield.stopRanging();
//     Control.disableFlightStatus(false);
}
```

Following using X-Ranging Sensors

```
// Do not remove the include below
#include "PlutoPilot.h"
#include "XRanging.h"
#include "Utils.h"
#include "Peripheral.h"
#include "Sensor.h"
#include "User.h"
#include "Control.h"


Interval T2;
//The setup function is called once at Pluto's hardware startup
void plutoInit()
{
      XRanging.init(LEFT);
      XRanging.init(RIGHT);
// Add your hardware initialization code here
}

//The function is called once before plutoPilot when you activate Developer Mode
void onLoopStart()
{
      Command.takeOff(250);
      FlightMode.set(ATLTITUDEHOLD);
}

// The loop function is called in an endless loop
void plutoLoop()
{
//Add your repeated code here
      int val = XRanging.getRange(LEFT);
      int val2 = XRanging.getRange(RIGHT);

      if((val >= 100 && val <= 250) || (val2 > 10 &&  val2 < 80))
      {
            Monitor.println("LEFT ",val);
            RcCommand.set(RC_ROLL,1450);
      }
      if((val2 >= 100 && val2 <= 250) || (val > 10 && val < 80))
      {
            Monitor.println("RIGHT ",val2);
            RcCommand.set(RC_ROLL,1550);
      }


      bool flag = T2.set(2000,1);
         if(flag){
            val = -100;
            RcCommand.set(RC_ROLL,1500);
         }
}

//The function is called once after plutoPilot when you deactivate
Developer Mode
void onLoopFinish()
{
       // do your cleanup stuffs here
}
```

Robot Operating System

The third method to program and control the drone is using the Robot Operating System. The instructions followed for this method are from here. Rosgraph for a typical ROS setup to control through keyboard is as shown in the figure below.



Nodes/Topics (all)

Group: ☑ Namespaces ☑ Actions Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ Unreachable ☑ Highlight ☑ Fit

Installation and Usage

The brief instructions for the ROS setup are as follows

1. ROS can be installed by following the instructions from here.
2. Once ROS is installed and running, PlutoX ROS package should be installed by using the following instructions
    a. Navigate to catkin workspace (*~/catkin_ws/src*)
    b. Clone the git repository with PlutoX ROS package in the *source* directory `git clone https://github.com/simmubhangu/pluto_drone.git .`
    c. Run *catkin_make* to build the package
    d. To add the installed packages into environment run `source ~/catkin_ws/devel/setup.zsh` . Change the setup file based on which shell is being used. Usually this command needs to be every time new terminal session is started. Or this can be automated by adding this line to *~/.zshrc* or *~/.bashrc*, based on the shell
3. To start controlling the drone using ROS, the following commands needs to be in order
    a. Run `roscore` in one session
    b. Run `rosrun plutoserver data_via_rosservice.py` in another session. This keeps getting data from magnetometer, accelerometer, gyro and altitude

c. **Controlling through Keyboard** - For this, run `roslaunch plutoserver drone_comb.launch` in another session. From the same session, use the below keys to control the drone

Moving around:

i

j k l

m

k : Reset

a : Arm drone

d : Dis-arm drone

r : stop smoothly

w : increase height

s : decrease height

d. **Controlling by sending commands** - The drone can also be controlled by sending individual commands one after the other, rather than sending commands continuously from the keyboard. A new terminal session should be created to run these commands.

i. Arm

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1000, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

ii. Disarm

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1000, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1000}"
```

iii. Increase Roll value to move forward with respect to x-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1600, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

iv. Decrease Roll value to move backward with respect to x-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1400, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

v. Increase Pitch value to move forward/left with respect to y-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1600, rcYaw: 1500, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

vi. Decrease Pitch value to move backward/right with respect to y-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1400, rcYaw: 1500, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

vii. Increase Throttle value to move up with respect to z-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1600, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

viii. Decrease Throttle value to move down with respect to z-axis

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1500, rcThrottle: 1400, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

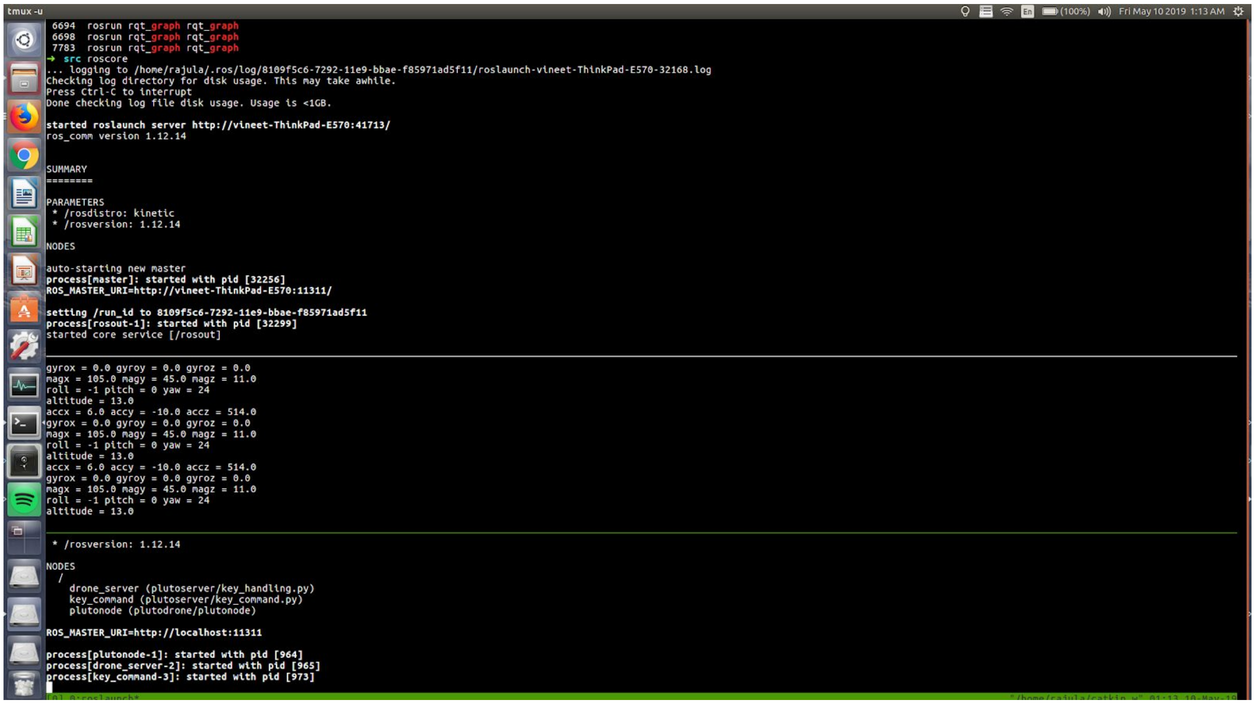
ix. Increase Yaw value to rotate in clockwise direction

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1600, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

x. Decrease Yaw value to rotate in anti-clockwise direction

```
rostopic pub /drone_command plutodrone/PlutoMsg "{rcRoll: 1500, rcPitch:
1500, rcYaw: 1400, rcThrottle: 1500, rcAUX1: 0, rcAUX2: 0, rcAUX3: 0,
rcAUX4: 1500}"
```

In the following image, screen utility ‘tmux’ is used to create three horizontal panes, one for roscore, rosservice and roslaunch for keyboard control.



```
 6694  rosrun rqt_graph rqt_graph
 6698  rosrun rqt_graph rqt_graph
 7783  rosrun rqt_graph rqt_graph
→ src roscore
... logging to /home/rajula/.ros/log/8109f5c6-7292-11e9-bbae-f85971ad5f11/roslaunch-vineet-ThinkPad-E570-32168.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://vineet-ThinkPad-E570:41713/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [32256]
ROS_MASTER_URI=http://vineet-ThinkPad-E570:11311/

setting /run_id to 8109f5c6-7292-11e9-bbae-f85971ad5f11
process[rosout-1]: started with pid [32299]
started core service [/rosout]
```

```
gyrox = 0.0 gyroy = 0.0 gyroz = 0.0
magx = 105.0 magy = 45.0 magz = 11.0
roll = -1 pitch = 0 yaw = 24
altitude = 13.0
accx = 6.0 accy = -10.0 accz = 514.0
gyrox = 0.0 gyroy = 0.0 gyroz = 0.0
magx = 105.0 magy = 45.0 magz = 11.0
roll = -1 pitch = 0 yaw = 24
altitude = 13.0
accx = 6.0 accy = -10.0 accz = 514.0
gyrox = 0.0 gyroy = 0.0 gyroz = 0.0
magx = 105.0 magy = 45.0 magz = 11.0
roll = -1 pitch = 0 yaw = 24
altitude = 13.0
```

```
 * /rosversion: 1.12.14

NODES
  /
    drone_server (plutoserver/key_handling.py)
    key_command (plutoserver/key_command.py)
    plutonode (plutodrone/plutonode)

ROS_MASTER_URI=http://localhost:11311

process[plutonode-1]: started with pid [964]
process[drone_server-2]: started with pid [965]
process[key_command-3]: started with pid [973]
```

This ROS package should support Multi drone connections according to the wiki page. But, we weren’t able to achieve this.

Accessories

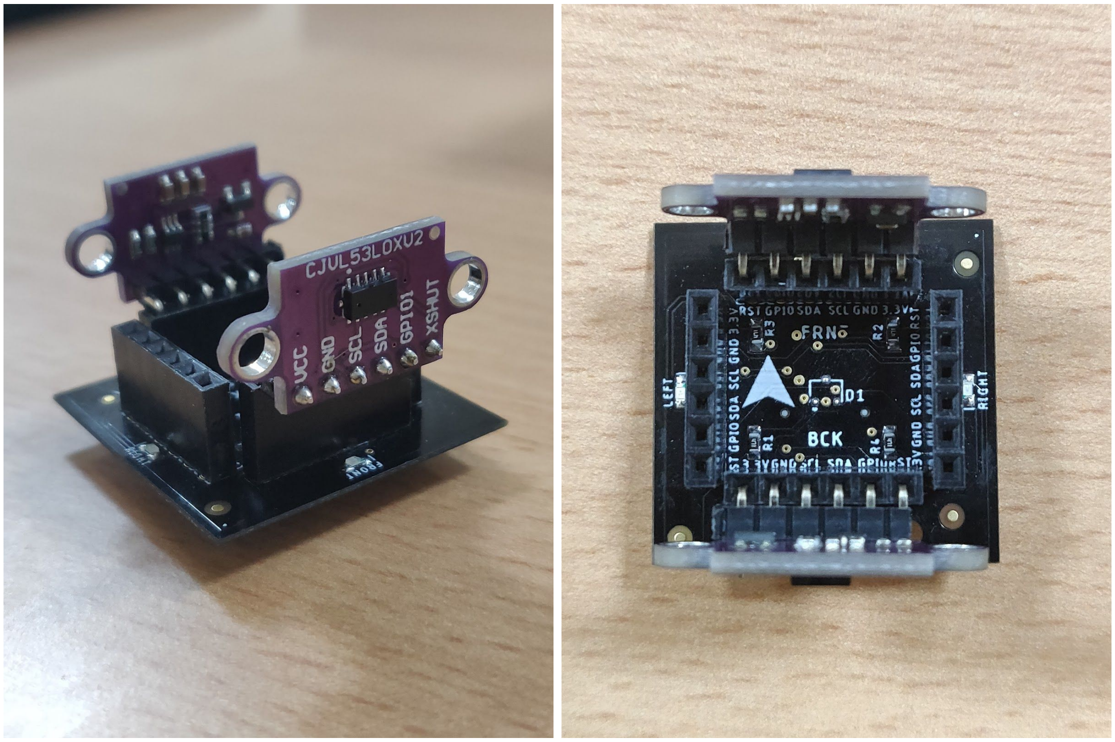
X-Ranging

This API is required to access the X-ranging Shield. The functions that come along with this API are as follows:

void init() : Initializes all the four sensors i.e., *FRONT*, *BACK*, *LEFT*, *RIGHT*.

void init(laser_e laser) : Initializes any one of the mentioned sensors among the following from *FRONT*, *BACK*, *LEFT*, *RIGHT*.

int16_t getRange(laser_e laser) : Instantaneous range from a particular sensor where laser can be any one of the following from *FRONT*, *BACK*, *LEFT*, *RIGHT*.



The X-Ranging sensor consists of 2 IR sensors(V53LOX)

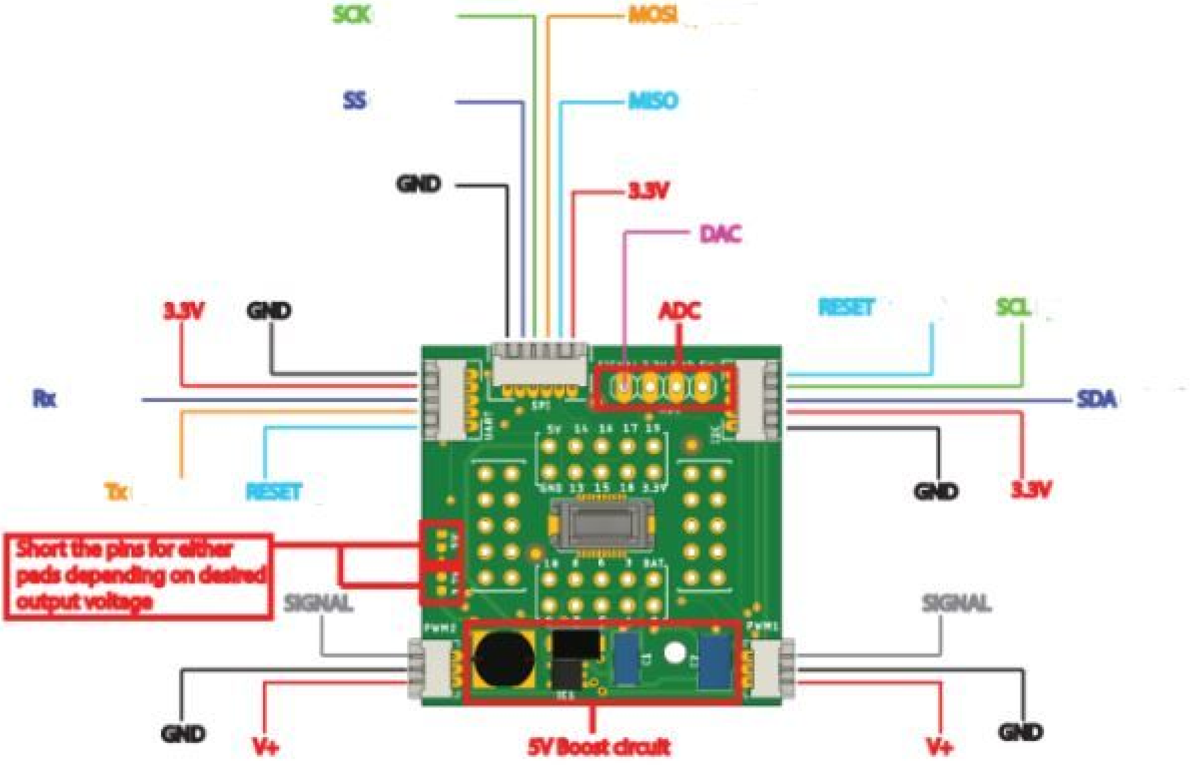
The IR sensor consists of 5 connections namely VCC, GND, RESET, I2C pins SCL and SDA. The breakout board provides I2C connections. Getting range from IR sensors is done by using Xshield API.

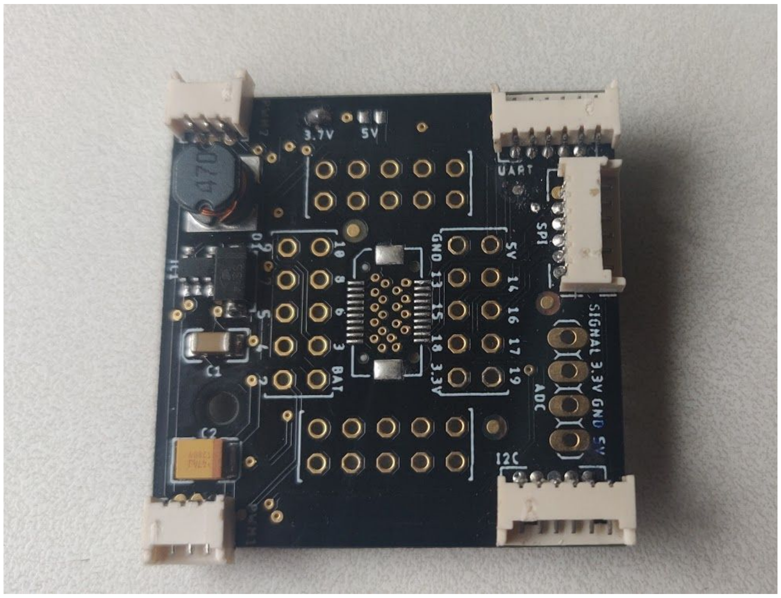
X-Breakout

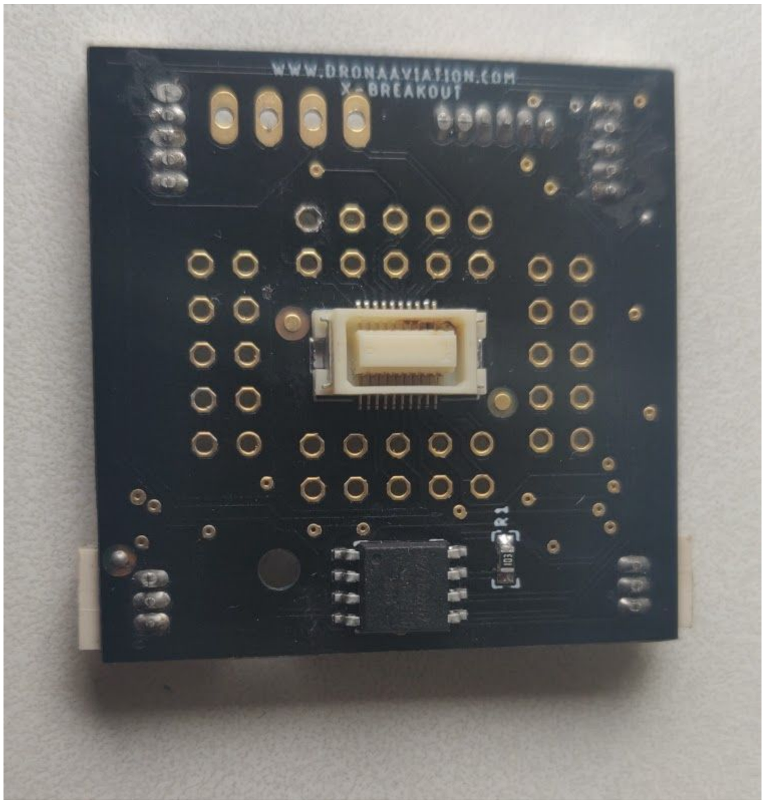
X-Breakout is an add-on shield using which any external sensor can be integrated with PlutoX.

It has special slots/ports provided for UART, I2C, SPI, ADC/DAC, PWM which can be used directly to integrate any external modules.

All the 20 unibus pinouts are accessible through X-Breakout and also 20 general purpose pinouts are available.


SCK
MOSI
SS
MISO
GND
3.3V
DAC
3.3V
GND
ADC
RESET
SCL
Rx
SDA
Tx
RESET
GND
3.3V
Short the pins for either pads depending on desired output voltage
SIGNAL
SIGNAL
GND
V+
5V Boost circuit
V+
GND





More details about the board can be found at https://docs.google.com/document/d/1SUjlZsFbw44WE9ejXZPoOXL9aCLifMGXwu8EQpqV8Hs/edit

PROBLEMS ENCOUNTERED

Over the course of the project, we encountered various types of problems with the drone. We are listing out the problems and solutions we know of, so that it helps anyone working on this in the future.

Sensors

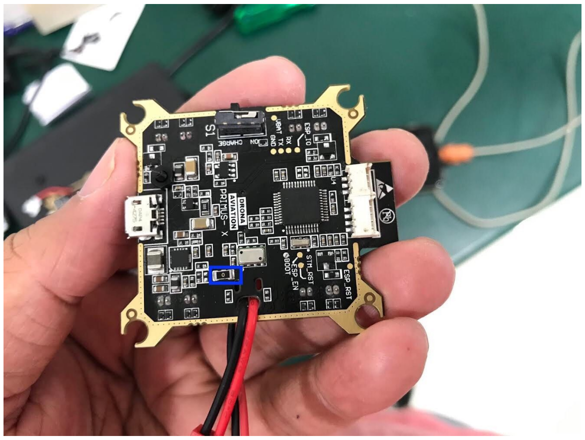
When couldn't interface digital sensors (say DHT11), on to the drone using the X-breakout board. The program to read data from DHT11 required interval/ clock function, i.e to able to read data after a given interval. These functions were available for both of the Cygnus versions, but we weren't aware of them initially and what are these functions. Later on, we found out that the functions *millis()* and *micros()* can be used for this purpose, to program. Besides this, due to lack of proper resources we couldn't interface the analog sensors on the old Cygnus. Only after contacting the Pluto technical support, we were told that there were problem with the old IDE and there is a newer version, that support reading analog values. By following the existing libraries of the sensors for arduino or related boards, the code can be ported on to the drone.

Apart from the API issues, there were also connecting issues with the sensors. There are exact jumper pins on the breakout board to connect the sensors. When we were able to get data from the sensor with the drone disarmed, we wanted to get data when the drone is being flown around. For this, we wanted to solder the sensor on the board. When we soldered, because of the soldered blob being a little big and because the breakout board is a 2-sided board, when we tried to connect the board after soldering, the drone malfunctioned and the breakout board slot on the board and on the drone got burnt a little bit. The drone kept spinning it’s fans the moment it was switched on since then. After getting in touch, with the technical support we were asked to short a resistor, we were able to get the board working again. We are not sure if it soldering error for sure, but it seems so.

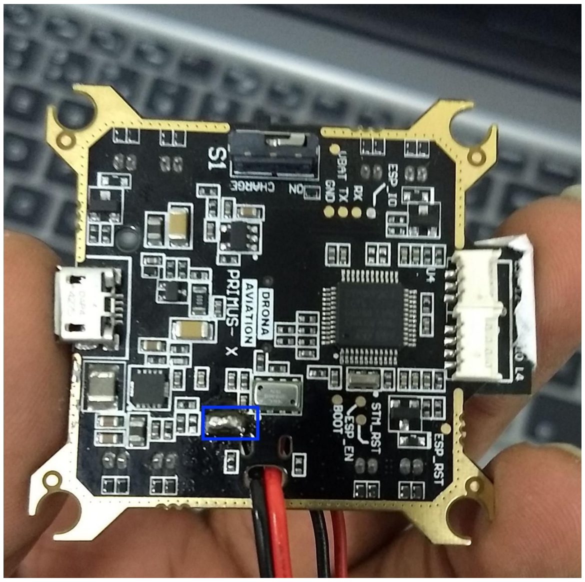
Try the following solution:

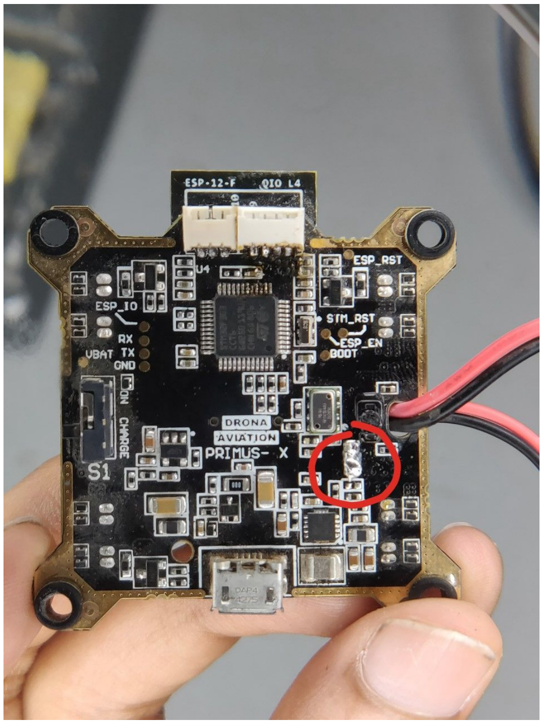
1) Unscrew the flight controller PrimusX from the drone frame.

2) Check back part of PrimusX



3) Remove the marked resistor as shown in the above picture and short it using solder as shown in the below picture.


S1
CHARGE
ON
VBAT
TX
RX
GND
ESP_IO
DRONA
AVIATION
PRIMUS- X
STM_RST
ESP_EN
BOOT
ESP_RST
U4
QIO L4


ESP-12-F
QIO L4
U4
ESP_RST
ESP_IO
RX
VBAT
TX
GND
STM_RST
ESP_EN
BOOT
ON
CHARGE
S1
DRONA
AVIATION
PRIMUS- X

Camera API

Going by the initial plan of trying to make the drone do image processing and follow a particular object, even though the camera module was available and the camera feed was accessible through the android app, we weren’t able to get the feed through ROS or through cygnus. If we would’ve got the feed through ROS, we can send the feed to the system on which *roscore* is running, do the required image processing, detect the object, find the direction and distance, and then using ROS drone can be commanded to move in that direction for the given distance. This keeps running in a loop, feed is sent by ROS node on drone, to the core system, where image processing is done and the command to move is given back to the node. The support for camera feed was still in development and wasn’t available over the course of the project. The beta version of the support for camera feed on ROS was made available by the company a few days back (i.e few days before the final demo). Therefore, we couldn’t work on the camera as planned initially. The Camera API ros package sent to us can be found here.

Cygnus versions

Since we were working with different Cygnus versions, the usage of APIs is different in both of them. Although there are separate documentations for both the versions, we faced difficulty to understand and use the APIs. For the same, besides following the API documentation we had to look into the header files present in *project_folder* > *platform*. By looking into the header files the return types, input parameter types and enum types defined are useful when using the APIs.

Swarm using ROS and telnet (STA and AP)

When we were trying to add Multi Drone support on top of ROS, the first step in it is to change the WiFi mode of ESP8266 from Access Point mode (AP mode) to Station mode (STA mode). And then the drone(s) have to be connected to a common wifi and commands have to be passed. To change the mode, the utility *telnet* has to used. Using *telnet*, the drones wifi mode can be changed. Sometimes, this mode will be not be reset when the drone is restarted. The mode can be reset back again using *telnet* or by flashing base firmware. We couldn’t get this working, as there was some problem with the connection and the commands were not being sent to the drone.

CONCLUSION & FUTURE WORK

Since the PlutoX drone is still in development, we had to face a lot of problems over the course of the project. Besides this, we were able to get basic functionalities of the drone working. The support for ROS, make it great to execute real time functionalities by sending data from drone to a remote node. The camera feed support which was released can be used to run image processing while the drone is in the air. If the right set of jumper pins/ if the sensors are properly soldered on to the drone, data can be retrieved from the drone when it is being flown around. The drone API is really great and it supports controlling motors, direction, speed, etc. Apart from all the bugs and lack of docs, overall PlutoX drone is a great platform to learn and program drones. We tried our best to include all the required details in this documentation, and we also included all the resources we have come across so that it makes it easier for anyone continuing this project.

REFERENCES

1. PlutoX website - https://www.dronaaviation.com/plutox/
2. PlutoX support page - https://www.dronaaviation.com/support/
3. API documentation -
    a. v1.0.6
       https://drive.google.com/open?id=1wtU-nnGNTPaOsXZ0_GVQ2klOFnSw--eA
    b. v2.0.0
       https://docs.google.com/document/d/1qUzQ2eYyPjvnZ2zWgFSVFsRz8hdydEv4QQBmXz5xt9M/edit
4. Other documentations -
   https://drive.google.com/drive/folders/12yho1OL4OuOJdStSYlG2r4aEH-reXx16
5. ROS support - http://wiki.ros.org/pluto_drone
6. Miscellaneous links -
    a. https://github.com/da-piyushpatel/PlutoX-hackathon

    b. https://dronebotworkshop.com/plutox-introduction/

    c. https://hackaday.io/projects?tag=plutox

    d. https://github.com/search?q=pluto%20x&type=Everything&repo=&langOverride=&start_value=1

7. X - breakout board Docs

    a. https://iiitborg-my.sharepoint.com/:b:/g/personal/vineet_reddy_iiitb_org/ER_C0kAOkCVKpEdm7z9k6ZUBp_ZDUdMGEQInjJ9k2kIN9Q?e=OW6uas

    b. https://docs.google.com/document/d/1SUjlZsFbw44WE9ejXZPoOXL9aCLifMGXwu8EQpqV8Hs/edit

8. ROS Camera package - https://iiitborg-my.sharepoint.com/:u:/g/personal/vineet_reddy_iiitb_org/ETPN7u4fC0ZAoA9lmN1fG4cBN_1mOofX0opTMMH-oaV0Cw?e=KW5NKg

9. Github link for our project - https://github.com/vishalgattani/PlutoX

10. Videos of the project - https://iiitborg-my.sharepoint.com/:f:/g/personal/vineet_reddy_iiitb_org/EqA6YE_tefVIjb5TWBBm7hoBo1KAZOtxntqKiJeaTLCmeg?e=koQdpO