AUTONOMOUS CONTROL OF A HEXACOPTER USING PIXHAWK AND BEAGLEBONE BLACK

By

Mugesh Shanmugam Pillai Gnanasekar

Project Guide: **Dr.Nina Mahmoudian**

CONTENTS

ABSTRACT

This Project Report presents one of the methods for autonomous control of an unmanned aerial vehicle with six rotors, a hexacopter. The hexacopter used consists of the frame and an autopilot installed on it and controlled using a companion on-board Linux computer. The Frame is built using DJI Flame wheel F550 ARF kit and the autopilot used is Pixhawk from 3DR Robotics. The companion computer used is a Beaglebone Black.

The main task of this project is to attain 'Off-board' control over the Pixhawk using the companion computer to send the set-points that we want the hexacopter to follow. This is done using MAVLink Protocol (Micro Air Vehicle Communication Protocol) between the Pixhawk and the Beaglebone Black.

At First, the DJI F550 frame is built from the ARF kit and all the arms are mounted to the Power Distribution Board (PDB) and the motors are mounted in the arms and the wires are welded to the PDB. The motors are controlled by E300 ESC's which is connected in-turn to the Pixhawk to take the PWM signals from the autopilot.

Once the built is done, next thing was to establish a serial UART connection between the Pixhawk and Beaglebone Black to communicate with each other via MAVLink messages. The connection was made and the Beaglebone Black can send and receive MAVLink messages from the Pixhawk. Once the communication is set, the code for offboard control of Pixhawk is written using C++ language and it will stream the setpoints to the Pixhawk controller to go to that position using GPS.

The offboard mode of Pixhawk needs continuous stream of target setpoints sent to it from the companion computer and the position controller compares the current position of the hexacopter using GPS and sends commands to the rotors to move to the target position. The program has been developed and compiled in the companion computer which will stream setpoints to the Pixhawk to move to the target position.

One major disadvantage of this approach is that there is no control over the velocity with which the hexacopter is moving. Hence the hexacopter will move rapidly to the target position with maximum speed which can be dangerous at times. Hence it is important to use intermediate setpoints to reduce the danger.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my project guide **Dr.Nina Mahmoudian** for giving me the opportunity to work on this project and providing constant support for completing the project and also providing me the opportunity to work in Nonlinear and Autonomous Systems Laboratory.

I also want to thank **Mr.Barzin Moridian** who guided me throughout the project and provided constant support and encouragement to complete the project.

I want to thank my co-worker in lab **Mr**.**Nithin** for helping me with handling the hardware components and making test runs.

I also thank **Mr.Lorenz Meier, Mr. Trent** and all users of PX4 google group for their timely help and making me understand about Pixhawk and how to communicate using Mavlink.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 OVERVIEW

Unmanned Aerial Vehicles have always been an area of interest for many researchers, hobbyists to use the powerful robots to do some meaningful tasks. Many researches are going all around the world to use these flying robots for surveillance applications in military, civil applications such as policing and firefighting, and nonmilitary security work such as inspection of power lines or mining areas where manual inspection is too dangerous and inaccessible. Its growing popularity have increased the interest in lot of research institutes to create powerful unmanned aerial vehicles for various applications. A multirotor is a type of aircraft like a helicopter but with more than two rotors. Currently, there are quadcopters (4 rotors), hexacopters (6 rotors), and octocopters (8 rotors) are commercially getting popular. When the number of rotors increase, it increases the weight carrying capacity of the UAV which are very useful in transporting heavy objects from place to place.

This project uses a hexacopter which can fly autonomously without any Radio Control units. The navigation of the hexacopter is achieved using a Pixhawk autopilot system with a companion computer which is Beaglebone Black.

The position of the hexacopter is obtained from the GPS installed on the Pixhawk. This limits the use of this hexacopter only outdoors as GPS won't work indoors. For indoor applications, a motion capture system is or computer vision systems are required to get the position of the hexacopter.

1.2 OBJETIVES OF THE PROJECT

The main objective of the project is to create an autonomous hexacopter which can fly to the target position using GPS and on board sensors installed on the Pixhawk in 'Offboard' mode. The companion computer communicated with Pixhawk over UART serial ports and access all of its sensor data using MAVLink messages. The position of the hexacopter is obtained from the GPS in the local co-ordinate frame and the target setpoints are given in the same co-ordinate frame from the Beaglebone Black.

2. SYSTEM HARDWARE

In this section we will talk about all the hardware components and the specifications of them used in this project.

2.1 HEXACOPTER FRAME

The first thing to do in this project was to build the frame of the hexacopter. The selection of the Frame had three major criterions.

I. It should be robust in nature to withstand the forces acting on it.
II. It should be able carry its own load and added components.
III. At the same time, it should be light-weight for better maneuverability.

Based on the above conditions, we selected the Flame wheel F550 ARF kit from DJI INNOVATIONS. It had the following qualities:

I. Ultra-strength Material for robustness
II. Light weight
III. Huge Assembly space
IV. Integrated PCB wiring

The main advantage is that it comes with an integrated Power Distribution Board to which we can just weld the ESC's wires to supply power to the motors from the central battery.

The specification of the frame is given in the following table:

Spec Name	Value
Diagonal Wheelbase	550mm
Frame Weight	478g
Takeoff Weight	1200g ~ 2400g

Table 2.1.1 Specification of the Frame

The part list from out of the box is shown in following table:

Part List	Number of items
Top Board 550FBT	1
Bottom Board 550FBT	1
Arms 550FAC	4
Arms 550FAW	2
920 KV Motors	6
ESCs	6
9.4 inch Propeller Pairs	4
Screws 550-M2.5×6	36
Screws 550-M3×8	24
Magic Strap 550MSX	1
Battery Band 550BBX	1
Power Line Pair 550PLP	1

Table 2.1.2 Part List of DJI Flame wheel F550 kit

The following figure shows the basic frame built out of the kit box.

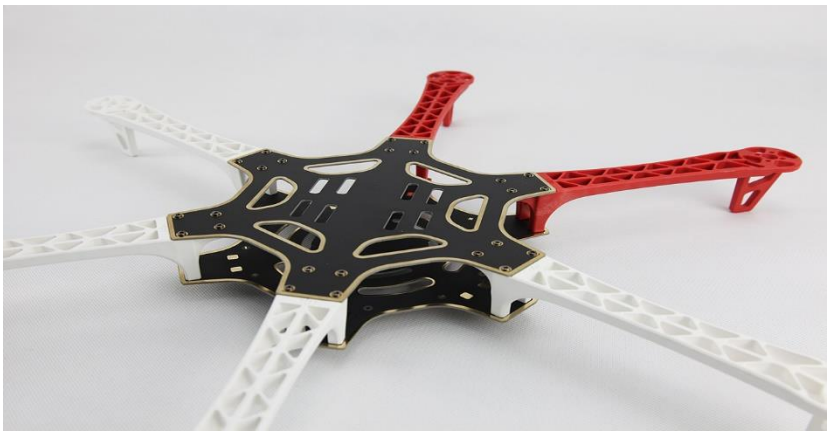


Fig 2.1.1 DJI Flame wheel F550

The motors and ESC's can be mounted to arms and the wires of the ESC can be welded to the PDB board and connected to the motor as shown in the following figure.

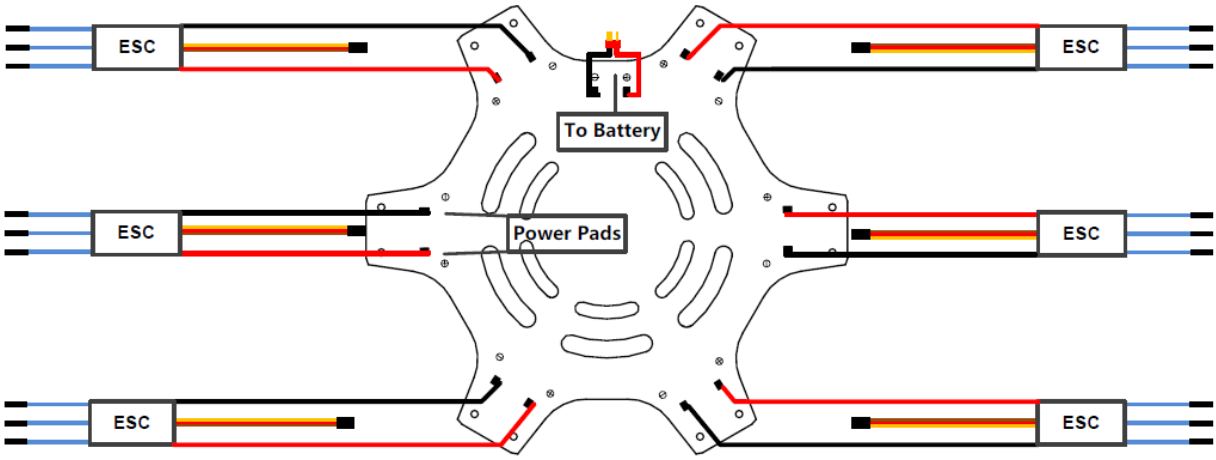


Fig 2.1.2 Schematic diagram of ESC wiring to PDB board

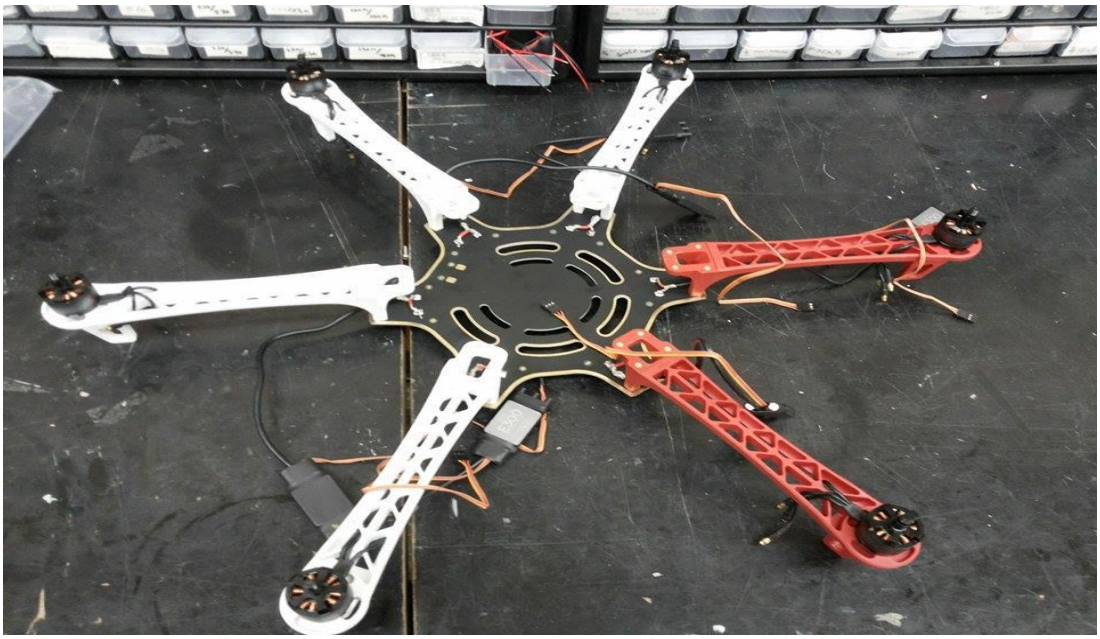


Fig 2.1.3 Motors and ESC's mounted in Frame

This is the basic build of the hexacopter which we are using in this project and all other components are mounted on to the frame.

2.2 ESC's AND MOTORS

An Electronic Speed Controller or ESC is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake. ESC is widely used in control of DC Brushless motors in various applications. This is used to control the motors of the hexacopter according to the PWM signals given by the autopilot. In this project, we are using a DJI E300 Propulsion system to control the motors. It has the specifications shown in the following table.

Spec Name	Value
Current	15A OPTO
Signal Frequency	30Hz ~ 450Hz
Battery	3S ~ 4S LiPo

Table 2.2.1 Specification of E300 ESC

The motors used are also DJI motors that comes with the Kit. There are three motors rotating clockwise and the other three rotating anti-clockwise and installed in an alternate manner. The following table gives the specification of the motors used. The motor rating is denoted in terms of 'KV Rating' which is nothing but RPM per volt. For example, if we use 11.2 Volt (3S) battery, the maximum RPM of the motor will be 11.2*920 = 10,212 RPM.

Spec Name	Value
Stator size	22×12mm
KV Rating	920 KV
Battery	3S ~ 4S LiPo

Table 2.2.2 Specifications of Motor used

2.3 BATTERY SELECTION

The battery selected is the one which is going to supply power to all the six motors, Autopilot system and the companion computer. The recommended batteries for the DJI systems are 3S or 4S lithium-ion polymer battery which is re-chargeable and very less in weight as compared to other batteries.

The battery life is extremely dependent on the weight of the aircraft and the amount of current that is drawn by the motors during flight. The flight time is decided by these criterions. The following Matlab code will give the flight time in minutes for a given battery depending on its capacity. The weight of the drone will be decided by all the components that it is going to lift. Once the rpm of the motor is known, determined using the propeller's pitch and diameter using the below equation:

```
Max thrust = (10^-10*(pitch)*(dia)^3*(rpm_max)^2)*6*28.35
```

This equation will give the maximum thrust that can be produced with the given set of motors and propellers. The RPM of motor will be decide by the voltage of the battery. This thrust should be at least double that of the total weight of the hexacopter so that it can be lifted by the motors easily. The following Matlab code will give the 'Flight Time in minutes' that can be obtained for the given battery capacity. With the help of this code, we can determine which battery to use depending upon the flight time that is needed for the application.

Matlab Code for flight time calculation:

```
%% Flight time and lift calculation
frm=478;  % 1.frame weight= 478g
con=38;   % 2.Controller weight = 38g
mot=318;  % 3.Motors = 6*53 = 318 g
prop=48;  % 4.Props = 6*8 = 48 g
beag=37;  % 5.BBB = 37g
batt=550; % 8.approx battery wt=550g
oth=100;  % 9.other componets = 100 g

%% All up weight
auw = frm+con+mot+prop+beag+batt+oth

%% Required Thrust on each motor
t_r = auw/6

%% Max thrust produced by 9443 dji prop with 920 kv motor
pitch=4.3 %inch
dia = 9.3 %inch
rpm_max = 10000;
thr_g=t_r*.0352;
thr_max=(10^-10*(pitch)*(dia)^3*(rpm_max)^2)*6*28.35 %maximum thrust by all 6
motor in grams

rpm=sqrt((thr_g*10^10)/(pitch*dia^3))
Power=5.3e-15*(rpm^3)*(dia^4)*pitch
Volt=11.1;
I=Power/Volt; %current required by single motor
Tot_I= 6*I % Total current required

%Type of cell= 3S Lipo
capacity=2500; %mAh
Ah=capacity/1000; %Ampere hour
C_rate= 40; %C
discharge=capacity*C_rate;

Flight_time= Ah*60/Tot_I %Flight time in minutes for hovering
```

We can get the amount of flight time in minutes for a given configuration of battery, motor, propellers and the total weight of the drone used.

2.4. AUTOPILOT SYSTEM – PIXHAWK PX4

In this section, we will discuss about the autopilot system that is used in this project which is PIXHAWK FLIGHT CONTROLLER from 3DR Robotics.

2.4.1 PX4 OVERVIEW

PX4 is an independent, open-source, open-hardware project Aiming at providing a high-end autopilot to the academic, hobby and industrial communities (BSD licensed) at low costs and high availability. It is a complete hardware and software platform, much like a computer, and can run multiple autopilot applications (e.g. the PX4 flight stack or APM). It is supported by the PIXHAWK Project of the Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology) and by the Autonomous Systems Lab and the Automatic Control Laboratory as well from a number of excellent individuals available from 3D Robotics and international 3DR distributors.

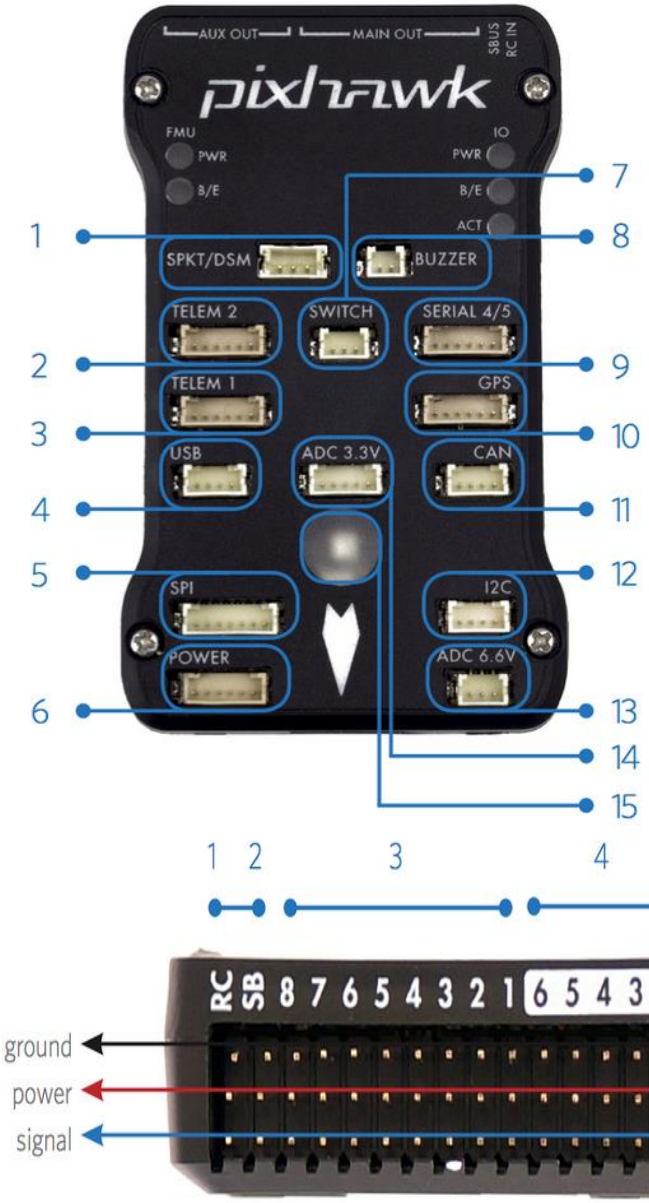
2.4.2 PIXHAWK HARDWARE

Pixhawk is a highly efficient autopilot system which has various in-built sensors (IMU unit) and various ports to connect GPS, telemetry, serial connection etc. which provides a lot of control options for multi-rotors.

The key features include the following:

I. It has Cortex M4 processor
II. 14 PWM/Servo outputs
III. Lot of connecting interfaces (UART, CAN, I2C)
IV. Multicolor LED, External safety switch
V. Micro SD card for storing large data-logging and storing firmware

The following figure shows the available ports on the Pixhawk PX4:



1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I^2C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator

1 Radio control receiver input
2 S.Bus output
3 Main outputs
4 Auxiliary outputs

Fig 2.4.1 Top and Front view of Pixhawk PX4

2.4.3 ON-BBOARD SENSORS

Pixhawk PX4 has in-built on board sensors like gyroscope, accelerometers, barometers, magnetometer to give the altitude, acceleration and attitude of the hexacopter.

1. ST Micro L3GD20H 16 bit gyroscope
2. ST Micro LSM303D 14 bit accelerometer / magnetometer
3. MEAS MS5611 barometer

2.4.4 GPS MODULE

In this project, we use a GPS module to get the position estimate of the hexacopter which limits the use of this method for only outdoor purposes where GPS can get good signal levels. We are using “Ublox NEO 7M GPS” Module that comes along with an integrated compass.

2.5 BEAGLEBONE BLACK

In this project, the companion computer that is going to do the offboard control of Pixhawk is Beaglebone black which is a low cost, community supported powerful computer capable of booting Linux under 10 seconds. The key features of Beaglebone Black are as follows:

Specification	Item
Processor	AM335x 1GHz ARM® Cortex-A8
RAM	512MB DDR3
On-board flash storage	4GB 8-bit eMMC
Accelerator	• 3D graphics • NEON floating-point accelerator
Connectivity	• USB client for power & communications • USB host • Ethernet • HDMI • 2x 46 pin headers

Table 2.5.1 Key Features of Beaglebone Black

The main reason to select the Beaglebone Black is its fast processing capability and it has a dedicated serial pin header for setting up serial communication with the Pixhawk. We used 'Debian' version of Linux installed in the Beaglebone black and the C++ code generated for offboard control is compiled in it and it will communicate with Pixhawk for reading and writing MAVlink messages.

3. SOFTWARE INSTALLATION AND SENSOR CALIBRATION

Once all the hardware components are selected and bought, the next step is install all the required software packages in the autopilot system and the companion computer and the on-board sensors need to be calibrated initially and it will be saved in appropriate files.

3.1 FLASHING THE FIRMWARE

The first thing to do once the autopilot system arrived is to flash the PX4 Flight stack into the Hardware. This can be done using 'QGroundControl' which is an Open Source Micro Air Vehicle Ground Control Station / Operator Control Unit. This software can be very effective to connect to the autopilot using MAVLink protocol and it can communicate well with the autopilot and provide a good ground control station.

The PX4 firmware is open source and can be downloaded directly from the link https://github.com/PX4/Firmware and can flash it to Pixhawk using QGroundControl by connecting over the USB.

3.2 SENSOR CALIBRATION

Once the firmware has been flashed, the next step is to calibrate the on-board sensors present in the Pixhawk PX4 to get better outputs from the sensors. This can be effectively done using the custom made software QGroundControl. The software communicates with the autopilot via MAVLink and writes the calibrated data to the appropriate parameter files in the PX4 firmware.

First, connect the Pixhawk to the computer with QGroundCntrol(QGC) installed on it. Select the appropriate 'COM Port' in which Pixhawk is visible and set the baudrate as 115200 which is the recommended value for sending MAVLink messages over USB and press 'Connect' button at the top right corner. Then the following sensors can be calibrated.

Once the vehicle is connected to QGC, move to the "Set up" Tab and select Sensor Calibration to find the three options for calibrating.

3.2.1 GYROSCOPE CALIBRATION

I. First, Keep the hexacopter on a level surface and click the 'GYRO button'. One beep will be heard and a message pops out 'Starting Gyro Cal'.
II. Without moving the hexacopter, wait for three beep sounds indicating that the calibration is done.
III. The results are written automatically to the appropriate parameter file in the PX4 firmware.

3.2.2 ACCELEROMETER CALIBRATION

I. This can be done by keeping the hexacopter in all six different orientation (level, on back, left side, right side, pitch up, pitch down). It is important to estimate the accelerometer offset and scale for each axis.
II. Press the 'Accel button' and one bee will be heard saying 'Accel calibration Started'.
III. Place the hexacopter in one of the six position and hold still. After 3 to 4 seconds staying still, one beep will be heard and the next beep indicates the calibration is done for that position.
IV. Keep repeating this step for all the 6 positions. The results will be written to appropriate file.

3.2.3 MAGNETOMETER CALIBRATION

I. Keep the hexacopter on a level surface and click the 'Mag button'. One beep will be heard and a message will appear saying "starting mag cal".
II. Without moving the hexacopter, wait for one beep to be heard and a message saying "rotate around X-axis", take the hexacopter and roll it around the forward facing axis until 1 beep is heard.
III. The message appears saying "rotate around y-axis", pitch the hexacopter around the sideways axis until one beep is heard.
IV. The message appears saying "rotate around z-axis", now yaw the hexacopter around the vertical axis until 3 beeps are heard and the message appears saying "mag calibration done".
V. The results are automatically written to the parameters file.

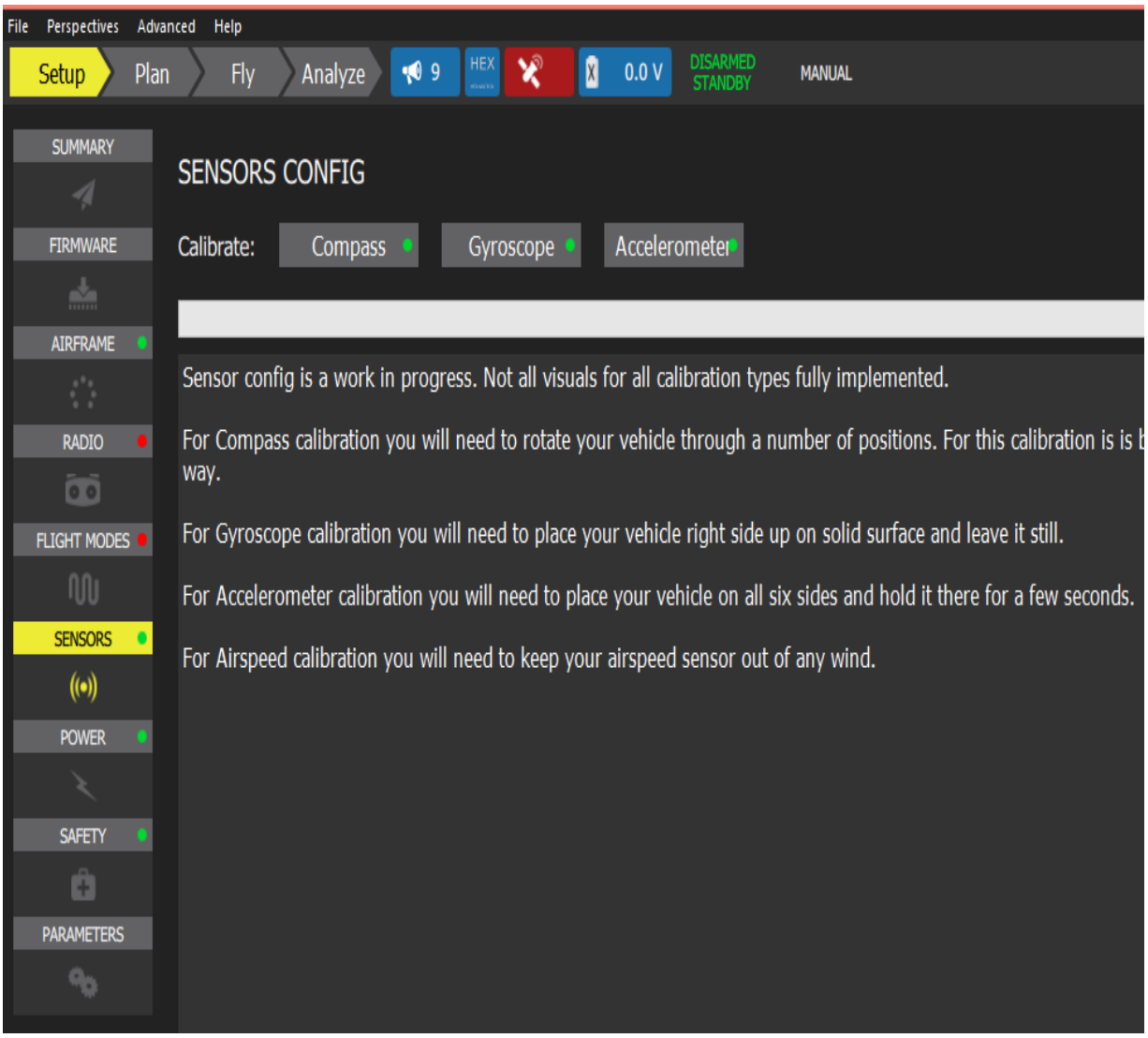


Fig 3.2.1 Sensor calibration using QgroundControl software

4. OFFBOARD CONTROL OF PIXHAWK

The primary focus of this project is to do the 'Offboard Control' of PX4 based hexacopter using a companion computer and communicate using MAVLink protocol. The MAVLink message set is attached in the Appendix 1 of this report which gives all the message set to send commands to the Pixhawk and receive messages from the Pixhawk.

4.1 MAVLINK PROTOCOL

MAVLink is a light-weight, header only message marshalling library for aerial vehicle autopilot system like Pixhawk. It packs C-structs over serial channels like UART with high efficiency and send these packets to the ground control station. It is extensively tested on the PX4, PIXHAWK, APM platforms and serves as communication link for the MCU/IMU communication as well as for Linux inter-process and ground link communication. MAVLink was first released early 2009 by Lorenz Meier under LGPL license. He is also the one who is working on the Pixhawk autopilot projects.

The MAVLink is an open-source software that can be downloaded from http://qgroundcontrol.org/mavlink/start and the common message list has been added in the Appendix. Also in order to know how to create a Mavlink message and encode them and decode them using the message set, one can take a look at the following link which explains it. http://qgroundcontrol.org/mavlink/create_new_mavlink_message.

This project completely uses Mavlink messages to send and read data from the Pixhawk and make decisions in the Beaglebone black.This serves as a great communication link between Pixhawk and the companion computer like Beaglebone Black to send and receive data between the two over serial communication channels.

4.2 UART INTERFACE BETWEEN PIXHAWK AND BEAGLEBONE BLACK

First step in starting the offboard control of Pixhawk is to establish a serial communication between the Pixhawk and Beaglebone Black. UART interface should be always considered for aerial operations over USB connection because USB's behavior is unpredictable and it can lead to garbage flight data which is

extremely dangerous.In this project, a UART-UART connection is made between the serial pins of Beaglebone black and Telem 2 Port of Pixhawk. This connection is made using a DF13 cable and the connection looks like below.

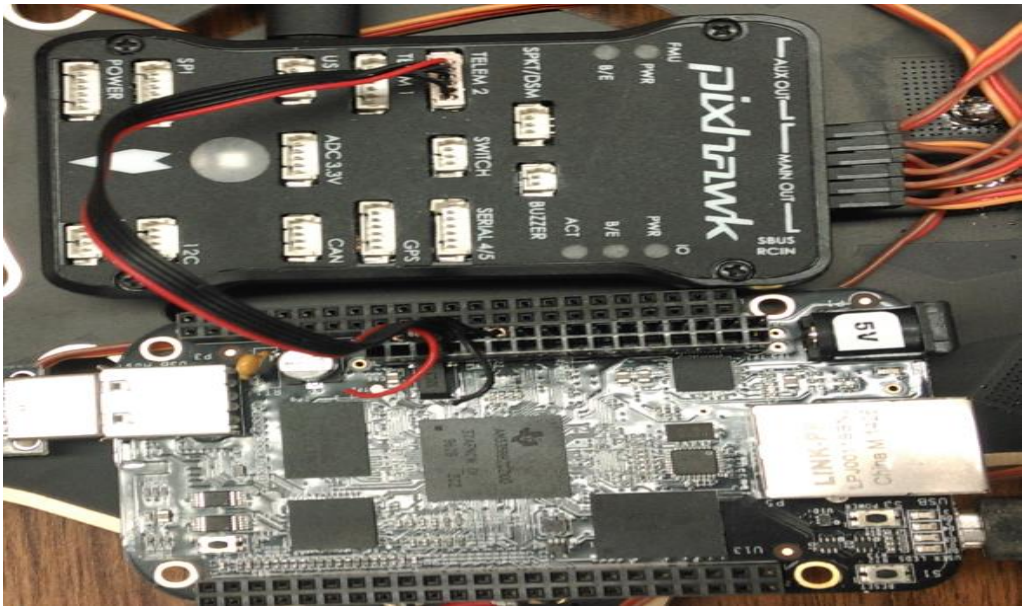


FIG 4.2.1 UART-UART CONNECTION

The pin out of Telem 2 and Serial Port of Beaglebone is given below.

TELEM 2		SERIAL	
1	+5V (red)	1	GND
2	Tx (out)	2	
3	Rx (in)	3	
4	CTS (in)	4	Tx(out)
5	RTS (out)	5	Rx(in)
6	GND	6	

Table 4.2.1 Pinout configuration of Telem 2 and Serial Ports

4.3 PRE-CAUTIONS FOR OFFBOARD MODE

I. Always it is good to use RC control connected to Pixhawk to switch to manual mode when something goes wrong.
II. Offboard control is dangerous to some extent because you will have no control over the hexacopter except the ability to switch over into a manual mode to regain control.
III. It's our responsibility to ensure that adequate preparation, testing and safety precautions are taken before offboard flights
IV. Always test without propellers before we are sure of the program.
V. Fully understand the autopilot modes and the difference between them (manual, position control, altitude control, automatic, offboard, etc.)

5. CODE GENERATION

Once all the hardware connections are done, it is time to generate the code for sending and receiving messaged to Pixhawk. The Offboard control is the ability to control a PX4 based MAV using off-board software. In this project, the position of the hexacopter is obtained using GPS Module attached to Pixhawk.

5.1 BODY NED FRAME

Pixhawk works in a NED (North-East-Down) co-ordinate system.

I. The x axis points towards the north pole.
II. The y axis points to the east
III. The z axis points downwards to the center of the earth

The GPS module will give the location of the position of the hexacopter in the Global frame which is converted internally by the Pixhawk using transformation matrices to give the position in “Local NED frame”. Hence we can easily get the current position using the GPS as it is internally converted to Local body frame by Pixhawk. The origin for this is set at the boot and at first discovery of GPS. But GPS is not very accurate and can give an error around one to two meter about the position. If we need accurate position control, one must consider using motion capture system. But the scope of this project is to use GPS as the position provider.

There are two specific tasks that needs to be done by the generated code in order to control the hexacopter in offboard mode.

- Switch to offboard mode using Mavlink command.
- Stream the target setpoints to the Pixhawk using the serial connection for it to follow.

5.2 ENABLING OFFBOARD MODE USING MAVLINK COMMAND

The Pixhawk will not start accepting the input target position from the companion computer until we enable the offboard mode in it. This can be done in two ways. One is by assigning a RC switch to enable this mode using QGC. Another way is to enable it using Mavlink command. We do not use RC control in this project. Hence, we do it using the Mavlink command. The command that is used to enable offboard mode is "MAV_CMD_NAV_GUIDED_ENABLE". The following code snippet will show how to enable and disable Offboard control.

MAV_CMD_NAV_GUIDED_ENABLE	hand control over to an external controller
Mission Param #1	On / Off (> 0.5f on)

```
toggle_offboard_control( bool flag ) {
        mavlink_command_long_t com;   //preparing the command using this message
        com.target_system    = system_id;
        com.target_component = autopilot_id;
        com.command          = MAV_CMD_NAV_GUIDED_ENABLE;
        com.confirmation     = true;
        com.param1           = (float) flag; // flag >0.5 => start, <0.5 => stop //Depends on
        flag variable, it will enable/disable
        // Encode
        mavlink_message_t message;
        mavlink_msg_command_long_encode(system_id, companion_id, &message, &com);
        // Send the message
        int len = serial_port->write_message(message);
        return len;
}
```

To enable the offboard command, set the ‘Mission Param 1 i.e. Flag’ in the above snippet as ‘True’ and call the toggle function to enable offboard mode. To disable, set that as ‘False’.

```
enable_offboard_control() {
        if ( control_status == false )   // Check whether it is already ON or not
        {       printf("ENABLE OFFBOARD MODE\n")
                // Sends the command to go off-board
                int success = toggle_offboard_control( true );
                // Check the command was written
                if ( success )
                        control_status = true;
                else
                {
                printf("Error: off-board mode not set, could not write message\n");
                }
        }}
disable_offboard_control()
{
        if ( control_status == true )
        {
                printf("DISABLE OFFBOARD MODE\n");
                int success = toggle_offboard_control( false );
                if ( success )
                        control_status = false;
                else
                {
                        ("Error: off-board mode not set, could not write message\n");
                }
}}
```

5.3 STREAMING TARGET SET POINTS

First we need to get the initial position of the hexacopter. Hence once the read thread is started, the initial position in Local NED frame can be decoded from the messages that s streamed by default once the Mavlink application is started in the Pixhawk This can be done by the following Mavlink command:

```
{mavlink_msg_local_position_ned_decode(&message, &(current_messages.local_position_ned));
//Decodes the local NED position from current message structure
```

Once the message has been decoded, the local position can be stored in the variable that we want and print the initial position of the hexacopter.

```
Mavlink_Messages local_data = current_messages;
        initial_position.x      = local_data.local_position_ned.x;
        initial_position.y      = local_data.local_position_ned.y;
        initial_position.z      = local_data.local_position_ned.z;
        initial_position.vx     = local_data.local_position_ned.vx;
        initial_position.vy     = local_data.local_position_ned.vy;
        initial_position.vz     = local_data.local_position_ned.vz;
        initial_position.yaw    = local_data.attitude.yaw;
        initial_position.yaw_rate = local_data.attitude.yawspeed;
        printf("INITIAL POSITION XYZ = [ %.4f , %.4f , %.4f ] \n", initial_position.x, initial_position.y,
initial_position.z);
        printf("INITIAL POSITION YAW = %.4f \n", initial_position.yaw);
```

This will get the initial position of the hexacopter in local NED frame. Once we have this information, we can set the target position and write the position to the pixhawk to follow using the write thread software developed by pixhawk developers.

The following snippet will enable the offboard mode by calling the 'enable offboard mode' function and then set position function will set the target position using 'SET _POSITION_TARGET _LOCAL_NED' command and we need to update the set points continuously using 'update setpoint()' function in the code which will update the current setpoint and start streaming by 'write setpoint()' that target position to the pixhawk and the pixhawk will move the hexacopter to that target position.

```
char response;
cout << "Do you want to enable offboard control? [Y/N]" << endl;
cin >> response;
if (response == 'Y') {
api.enable_offboard_control(); // api is an object created for the
autopilot interface class where other functions are written
usleep(100); // give some time to let it sink in
}
else {
printf ("cannot enable offboard control") }
// now the autopilot is accepting setpoint commands
printf("SEND OFFBOARD COMMANDS\n");
mavlink_set_position_target_local_ned_t sp;
mavlink_set_position_target_local_ned_t ip = api.initial_position;
//set the target position in local NED frame
 set_position( ip.x - 5.0 ,  ip.y - 5.0 ,  ip.z  ,   sp        );
api.update_setpint();
```

The set_position function is as follows:

```
void
set_position(float x, float y, float z, mavlink_set_position_target_local_ned_t &sp)
{       sp.type_mask = MAVLINK_MSG_SET_POSITION_TARGET_LOCAL_NED_POSITION;
        sp.coordinate_frame = MAV_FRAME_LOCAL_NED;
        sp.x   = x;         sp.y   = y;         sp.z   = z;
        printf("POSITION SETPOINT XYZ = [ %.4f , %.4f , %.4f ] \n", sp.x, sp.y, sp.z);
}
```

The update_setpoint and write_setpoint functions are as follows:

```
update_setpoint(mavlink_set_position_target_local_ned_t setpoint)
{       current_setpoint = setpoint;
}
Write_setpoint()
{       mavlink_set_position_target_local_ned_t sp = current_setpoint;
        sp.target_system    = system_id;
        sp.target_component = autopilot_id;
        mavlink_message_t message;
        mavlink_msg_set_position_target_local_ned_encode(system_id, companion_id, &message,
&sp);
        int len = write_message(message);
        return;
}
```

Finally printing the current position of the pixhawk as it will start moving to the target position can be done as follows:

```
for (int i=0; i < 100; i++)
	{
		mavlink_local_position_ned_t pos = api.current_messages.local_position_ned;
		printf("%i CURRENT POSITION XYZ = [ % .4f , % .4f , % .4f ] \n", i, pos.x, pos.y, pos.z);
		sleep(1);
	}
```

Hence we discusses about how to enable/disable offboard mode using Mavlink commands and about writing target setpoints to the Pixhawk to go to. Once the target position message is encoded and sent to pixhawk, it will start moving the hexacopter to reach that target position using the position feedback from the GPS. The position control algorithm in the PX4 flight stack will make the hexacopter move to achieve the target position.

6. PROGRAM EXECUTION AND RESULT

Once the program is created and compiled using G++ compiler in beaglebone black, we can execute the offboard control using the procedure given below:

Step 1: First step is to start the Mavlink application in the Pixhawk shell. Pixhawk shell is available to access only over USB. Hence connect the pixhawk to your computer using USB and 'screen' into the shell using the following command.

```
Screen /dev/ttyACM1 57600 8N1
```

Step 2: When we press Enter, the 'nsh' prompt appears. Then start the mavlink application at the Telem 2 port using this command:

```
nsh> mavlink start –d /dev/ttyS2
```

After starting the mavlink application, disconnect the pixhawk.

Step 3: Now change the directory in beaglebone black to the one in which the software program for offboard control is stored using 'cd' command.

Step 4: Now run the program. It will start reading messages from Pixhawk and stream target setpoints that we want the hexacopter to follow.

In the example result shown, the target position is given as (-5,-5,0) from the initial position in the local NED frame. The output obtained is as follows:

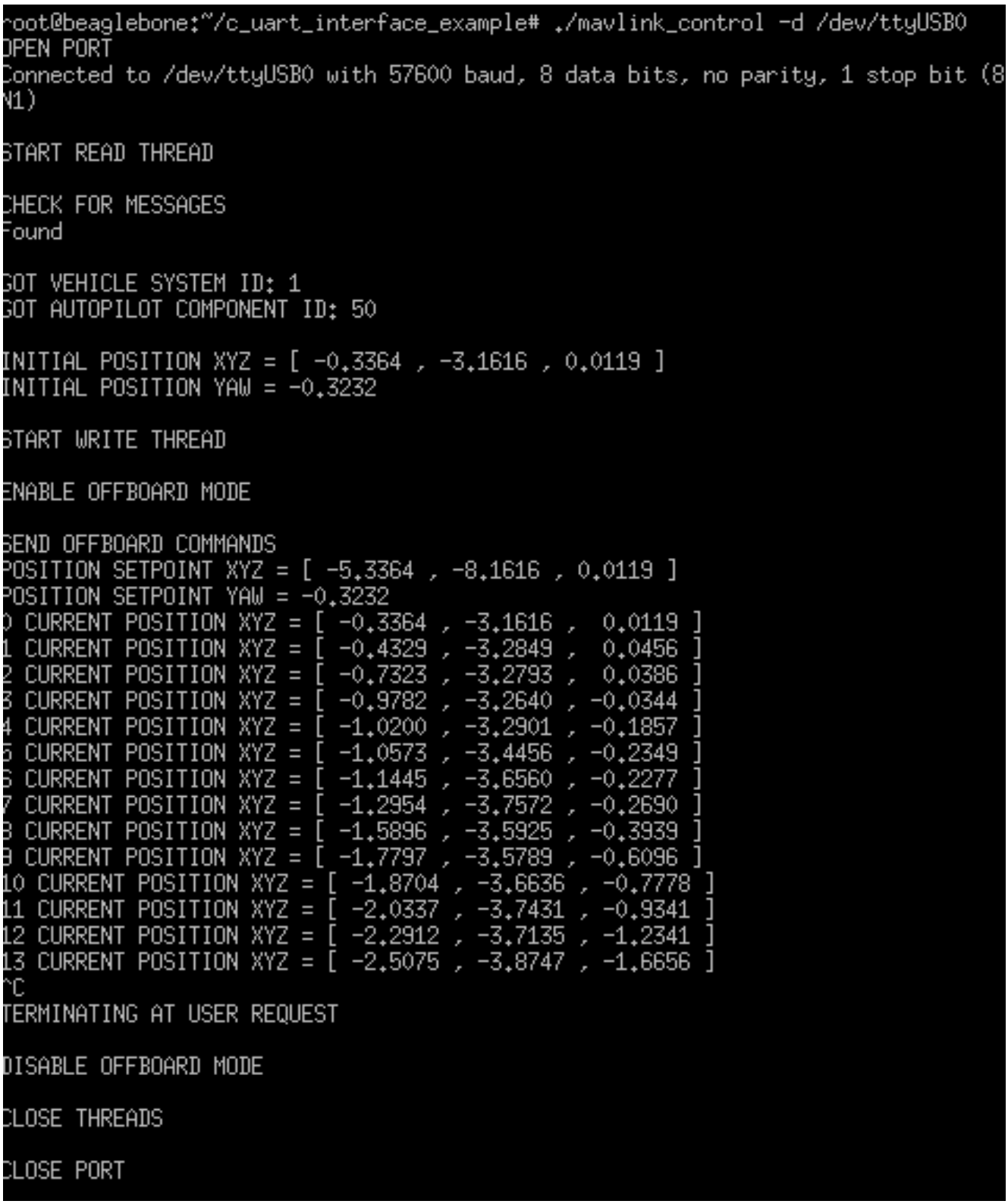


Fig 6.1 Example Output of offboard control using beaglebone black

7. CONCLUSION

In this project, we discussed in detail about how to create an autonomous UAV using GPS as position providing system. This autonomous vehicle is created using a powerful autopilot system and a companion computer to control the autopilot externally. This eliminates the use of Radio Control for controlling the aerial vehicle and gives autonomous ability to follow a user given path.

Just by giving the target set point, we have made the UAV to move to that position autonomously. As we use GPS for determining the position in this project, this limits the use of this system only out-doors as GPS signals will not be effective indoors. Also we can only stream the setpoints to the Pixhawk to follow but the speed of it cannot be controlled which is big disadvantage of this project.

The UAV may take maximum speed to reach the set point. It is completely out of control of the user. Also the offboard control of pixhawk can be dangerous and go out of control sometimes. Hence it is always safer to use back RC control and assign a switch to disable offboard control whenever it is going out of control.

Thus, Pixhawk offers a great ability in creating autonomous UAV's by connecting it with a companion computer to send the target positions. Instead of GPS, a motion capture system or vision computing algorithm can be used to make this work for indoor applications. This can give more accurate position estimate of the hexacopter and accurate control can be achieved. This project provides a very basic idea of creating an autonomous UAV using offboard control of Pixhawk with beaglebone Black.

This can be further developed to achieve great autonomous functions and the hexacopter can be made to do useful tasks autonomously.

APPENDIX A

MAVLINK COMMON MESSAGE SET

MAV_MODE_FLAG:

These flags encode the MAV mode.

CMD ID	Field Name	Description
128	MAV_MODE_FLAG_SAFETY_ARMED	0b10000000 MAV safety set to armed. Motors are enabled / running / can start. Ready to fly.
64	MAV_MODE_FLAG_MANUAL_INPUT_ ENABLED	0b01000000 remote control input is enabled.
32	MAV_MODE_FLAG_HIL_ENABLED	0b00100000 hardware in the loop simulation. All motors / actuators are blocked, but internal software is full operational.
16	MAV_MODE_FLAG_STABILIZE_ENABLED	0b00010000 system stabilizes electronically its attitude (and optionally position). It needs however further control inputs to move around.
8	MAV_MODE_FLAG_GUIDED_ENABLED	0b00001000 guided mode enabled, system flies MISSIONs / mission items.
4	MAV_MODE_FLAG_AUTO_ENABLED	0b00000100 autonomous mode enabled, system finds its own goal positions. Guided flag can be set or not, depends on the actual implementation.
2	MAV_MODE_FLAG_TEST_ENABLED	0b00000010 system has a test mode enabled. This flag is intended for temporary system tests and should not be used for stable mplementations.
1	MAV_MODE_FLAG_CUSTOM_MODE_ENABLED	0b00000001 Reserved for future use.

LOCAL_POSITION_NED:

The filtered local position (e.g. fused computer vision and accelerometers). Coordinate frame is right-handed, Z-axis down (aeronautical frame, NED / north-east-down convention)

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp (milliseconds since system boot)
x	float	X Position
y	float	Y Position
z	float	Z Position
vx	float	X Speed
vy	float	Y Speed
vz	float	Z Speed

GLOBAL_POSITION_INT:

The filtered global position (e.g. fused GPS and accelerometers). The position is in GPS-frame (right-handed, Z-up). It is designed as scaled integer message since the resolution of float is not sufficient.

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp (milliseconds since system boot)
lat	int32_t	Latitude, expressed as * 1E7
lon	int32_t	Longitude, expressed as * 1E7
alt	int32_t	Altitude in meters, expressed as * 1000 (millimeters), AMSL (not WGS84 - note that virtually all GPS modules provide the AMSL as well)
relative_alt	int32_t	Altitude above ground in meters, expressed as * 1000 (millimeters)
vx	int16_t	Ground X Speed (Latitude), expressed as m/s * 100
vy	int16_t	Ground Y Speed (Longitude), expressed as m/s * 100
vz	int16_t	Ground Z Speed (Altitude), expressed as m/s * 100
hdg	uint16_t	Compass heading in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT16_MAX

SET_POSITION_TARGET_LOCAL_NED:

Set vehicle position, velocity and acceleration setpoint in local frame.

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp in milliseconds since system boot
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
coordinate_frame	uint8_t	Valid options are: MAV_FRAME_LOCAL_NED = 1, MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED = 9
type_mask	uint16_t	Bitmask to indicate which dimensions should be ignored by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates that none of the setpoint dimensions should be ignored. If bit 10 is set the floats afx afy afz should be interpreted as force instead of acceleration. Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw rate
x	float	X Position in NED frame in meters
y	float	Y Position in NED frame in meters
z	float	Z Position in NED frame in meters (note, altitude is negative in NED)
vx	float	X velocity in NED frame in meter / s
vy	float	Y velocity in NED frame in meter / s
vz	float	Z velocity in NED frame in meter / s
afx	float	X acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N

afy	float	Y acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
afz	float	Z acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
yaw	float	yaw setpoint in rad
yaw_rate	float	yaw rate setpoint in rad/s

POSITION_TARGET_LOCAL_NED:

Set vehicle position, velocity and acceleration setpoint in local frame.

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp in milliseconds since system boot
coordinate_frame	uint8_t	Valid options are: MAV_FRAME_LOCAL_NED = 1, MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED = 9
type_mask	uint16_t	Bitmask to indicate which dimensions should be ignored by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates that none of the setpoint dimensions should be ignored. If bit 10 is set the floats afx afy afz should be interpreted as force instead of acceleration. Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw rate
x	float	X Position in NED frame in meters
y	float	Y Position in NED frame in meters
z	float	Z Position in NED frame in meters (note, altitude is negative in NED)
vx	float	X velocity in NED frame in meter / s
vy	float	Y velocity in NED frame in meter / s
vz	float	Z velocity in NED frame in meter / s
afx	float	X acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
afy	float	Y acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
afz	float	Z acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
yaw	float	yaw setpoint in rad
yaw_rate	float	yaw rate setpoint in rad/s

SET_POSITION_TARGET_GLOBAL_INT:

Set vehicle position, velocity and acceleration setpoint in the WGS84 coordinate system.

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp in milliseconds since system boot. The rationale for the timestamp in the setpoint is to allow the system to compensate for the transport delay of the setpoint. This allows the system to compensate processing latency.
target_system	uint8_t	System ID
target_component	uint8_t	Component ID
coordinate_frame	uint8_t	Valid options are: MAV_FRAME_GLOBAL_INT = 5, MAV_FRAME_GLOBAL_RELATIVE_ALT_INT = 6, MAV_FRAME_GLOBAL_TERRAIN_ALT_INT = 11
type_mask	uint16_t	Bitmask to indicate which dimensions should be ignored by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates that none of the setpoint dimensions should be ignored. If bit 10 is set the floats afx afy afz should be interpreted as force instead of acceleration. Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw rate
lat_int	int32_t	X Position in WGS84 frame in 1e7 * meters
lon_int	int32_t	Y Position in WGS84 frame in 1e7 * meters
alt	float	Altitude in meters in AMSL altitude, not WGS84 if absolute or relative, above terrain if GLOBAL_TERRAIN_ALT_INT
vx	float	X velocity in NED frame in meter / s
vy	float	Y velocity in NED frame in meter / s
vz	float	Z velocity in NED frame in meter / s
afx	float	X acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
afy	float	Y acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
afz	float	Z acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / s^2 or N
yaw	float	yaw setpoint in rad
yaw_rate	float	yaw rate setpoint in rad/s

REFERENCES:

https://pixhawk.ethz.ch/mavlink/ - Full information about Mavlink and developing messages

https://github.com/PX4/Firmware - PX4 flight stack open-source download

https://pixhawk.ethz.ch/ - Pixhawk project group

https://pixhawk.ethz.ch/software/imu/start - Software used in Pixhawk

https://pixhawk.org/start - Pixhawk community website providing all the information required

https://github.com/mavlink/c_uart_interface_example - An example for offboard control using companion computer

http://www.pixhawk.org/users/sensor_calibration - Documentation about calibration of sensors

http://mavlink.org/messages/common - Common Mavlink message set that work well with Pixhawk PX4.

http://dev.ardupilot.com/wiki/building-the-code/building-for-beaglebone-black-on-linux/ - Building ardupilot software in Beaglebone black

https://groups.google.com/forum/#!forum/px4users - Common mailing list for all PX4 users

https://pixhawk.org/dev/nuttx/building_and_flashing_console - Flashing firmware from console

http://qgroundcontrol.org/users/start?&#installation_on_linux_ubuntu_-_set_proper_permissions - QGC tutorials

https://pixhawk.org/dev/offboard_control - Full details about offboard control of Pixhawk using companion computer