Precise Unmanned Aerial Vehicle (UAV) Flight Control

Fine Control and Tracking of Drones

[redacted]

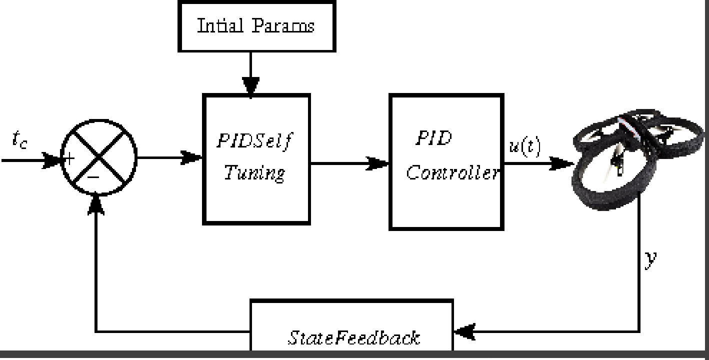
Problem Statement & Motivation

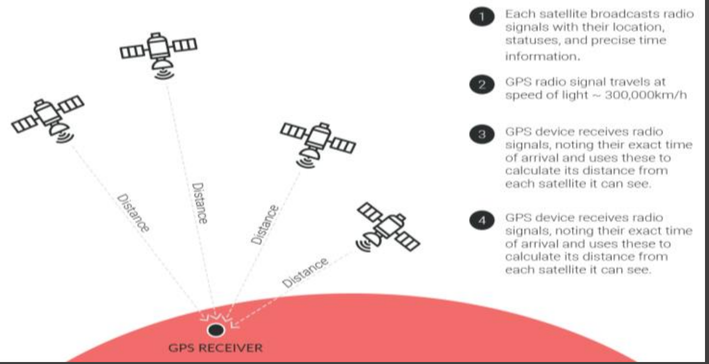
Precise PID control of the drone

Precise positioning of the drone

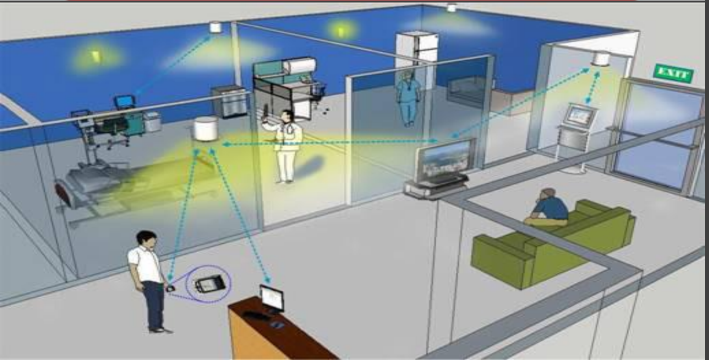
Towards precise movement of the drone in fine (cm) steps!

Motivation





1. Each satellite broadcasts radio signals with their location, statuses, and precise time information.
2. GPS radio signal travels at speed of light ~ 300,000km/h
3. GPS device receives radio signals, noting their exact time of arrival and uses these to calculate its distance from each satellite it can see.
4. GPS device receives radio signals, noting their exact time of arrival and uses these to calculate its distance from each satellite it can see.



- Precise PID control of the drone

- Precise positioning of the drone

[1] Designing of self tuning PID controller for AR drone quadrotor
[2] Develop App with Geolocation (https://theappsolutions.com/)
[3] Cognitive Indoor Localization (http://sampl.eelabs.technion.ac.il)

Literature Review

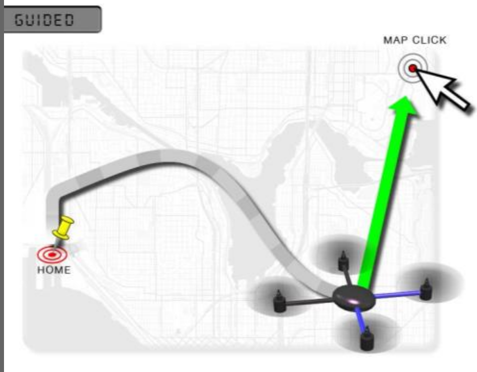
PID

- Drone is equipped with various PID controllers
- Prior Work have focused on manually tuning these PID controllers

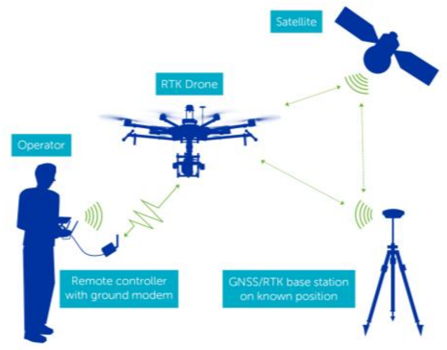
Positioning

- Auto Navigation exists with GNSS
- GNSS based positioning can be improved with RTK

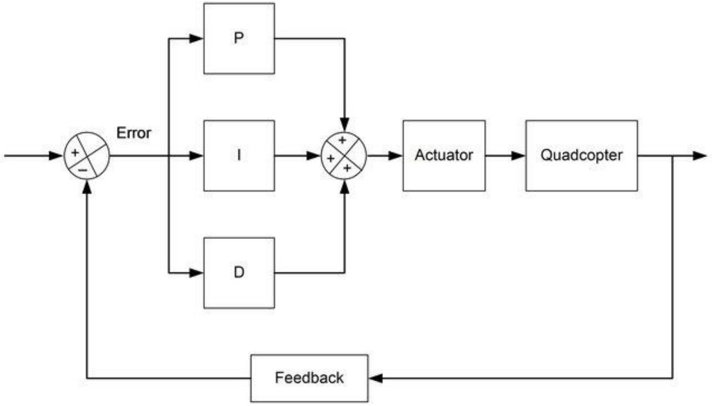
[1] http://ardupilot.org/copter/docs/ac2_guidedmode.html
[2] https://www.heliguy.com/blog/2017/10/04/benefits-of-rtk/



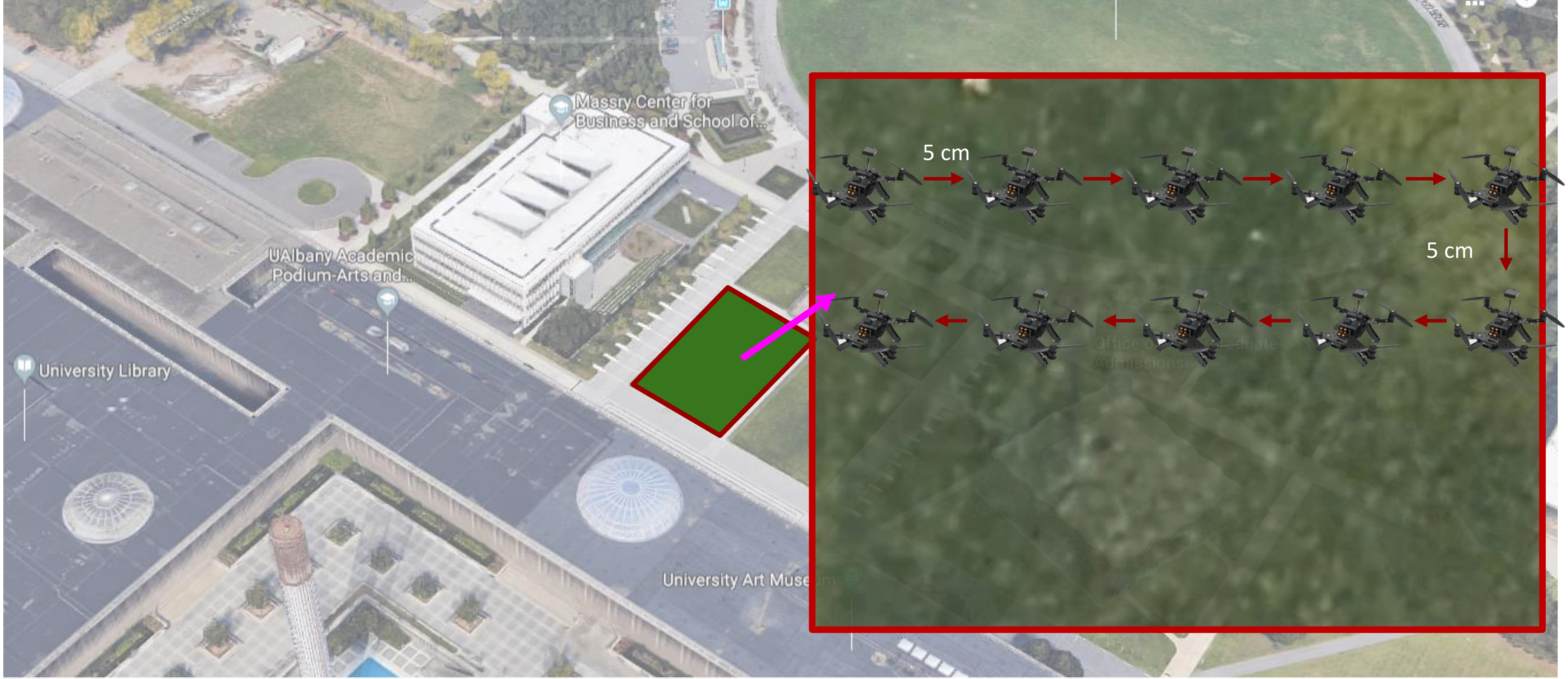
GNSS based Navigation



RTK



PID



Problem Statement

Facilitate Fine motion in Indoor and Outdoor Environments

Solution Space

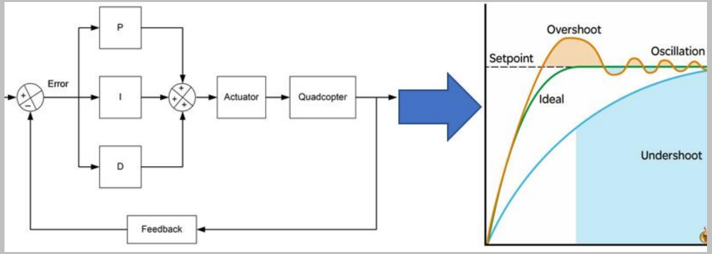
- Drones consist of various sensors (Cameras, Radios, IMUs, GPS, Barometer)

→ Improve Sensing

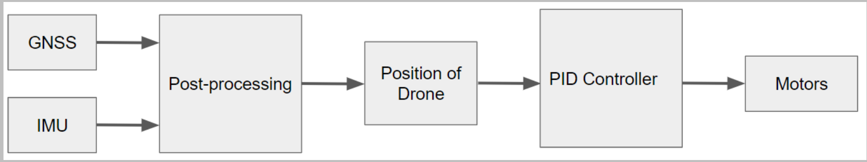
- Drone consists of complex controllers (**rate**, **attitude, altitude**, and **velocity & position controllers)**

→ Design / Modify & Tweak.

- Build a precise positioning mode with efficient PID controllers



PID

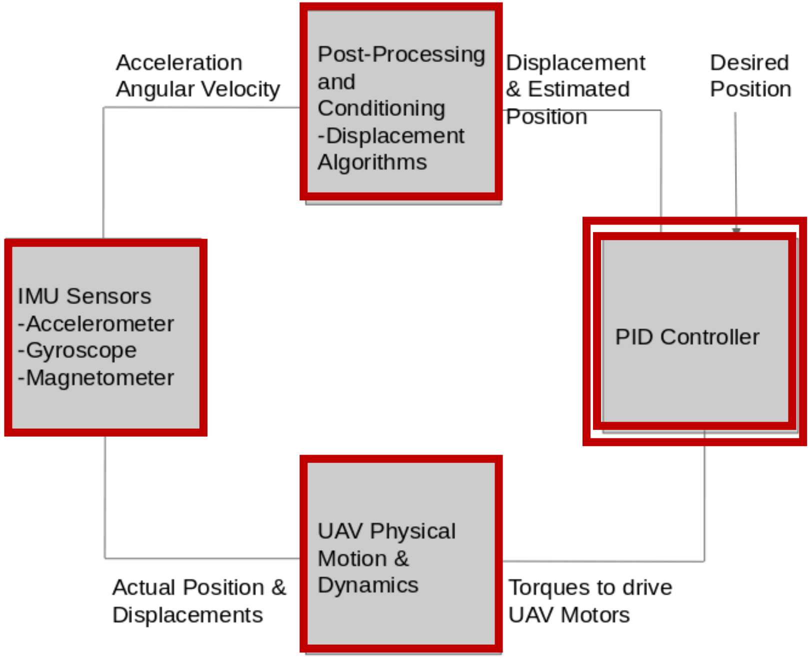


Overall System

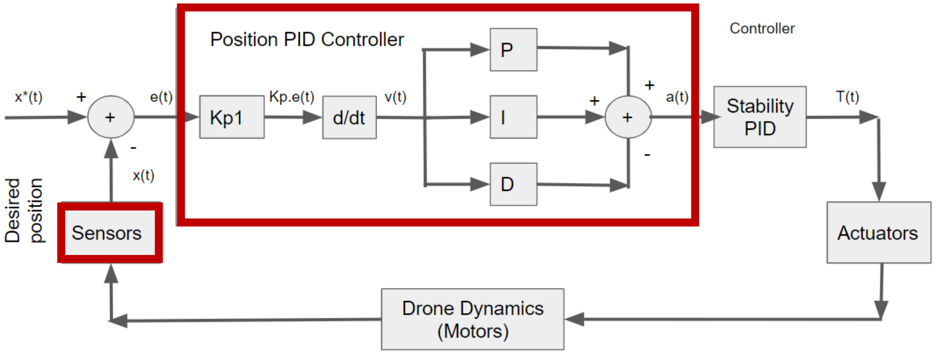
[1] https://oscarliang.com/quadcopter-pid-explained-tuning/

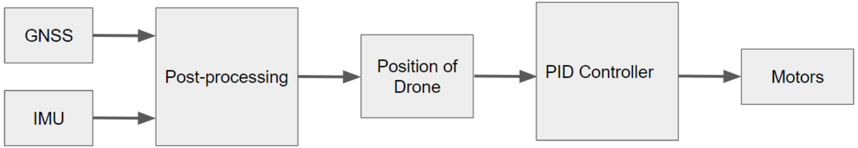
System Design

Conceptual Design



System Design





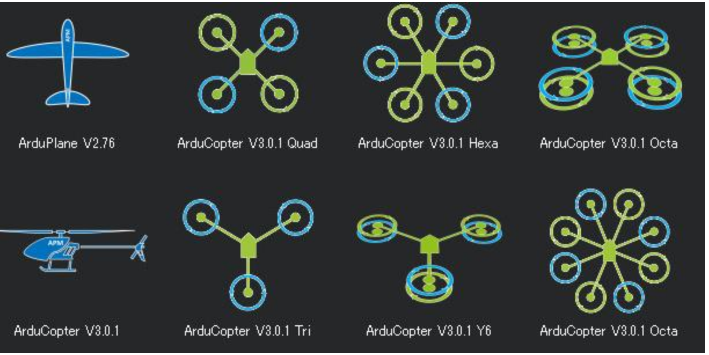
Implementation

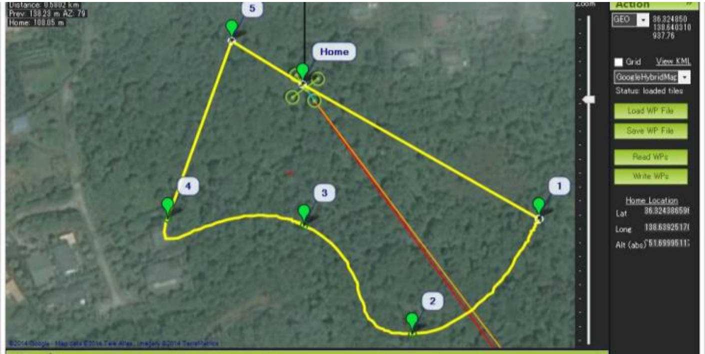
- Hardware (Drone (MCU + Flight Controller) + Sensors)

- Firmware (Ardućopter above PX4)

- Software (Libraries + GUI)

1. Intel® Aero Compute Board
2. Intel® Aero Flight Controller, preprogrammed with Dronecode* PX4* autopilot
3. Intel® RealSense R200 Camera for 3D depth sensing
4. 8 MP RGB camera (front-facing)
5. VGA camera, global shutter, monochrome (down-facing) (not visible in photo)
6. GPS and Compass
7. Four ESCs, Motors, Propellers
8. Carbon Fiber Chassis (Fully Assembled)
9. Radio Control Transmitter and Reciever





[1] http://ardupilot.org/planner/

Hardware

1. Intel® Aero Compute Board
2. Intel® Aero Flight Controller, preprogrammed with Dronecode* PX4* autopilot
3. Intel® RealSense R200 Camera for 3D depth sensing
4. 8 MP RGB camera (front-facing)
5. VGA camera, global shutter, monochrome (down-facing) (not visible in photo)
6. GPS and Compass
7. Four ESCs, Motors, Propellers
8. Carbon Fiber Chassis (Fully Assembled)
9. Radio Control Transmitter and Reciever

Connector Locations and Pin Orientation

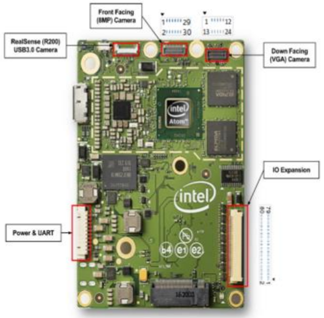


Figure 2. Connector Locations and Pin Orientation

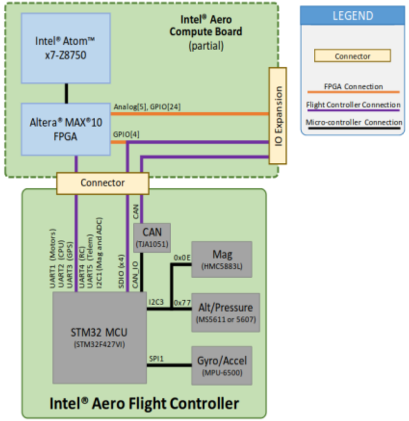


Figure 11. Hardware Block Diagram – Aero Flight Controller


Linux yocto PROJECT

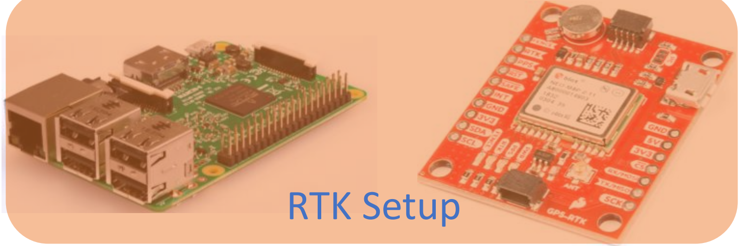
Dronecode PX4 inside

Powerful Compute

Flexible I/O, Wireless Comms

Open Source Development

Additional Payload



RTK Setup

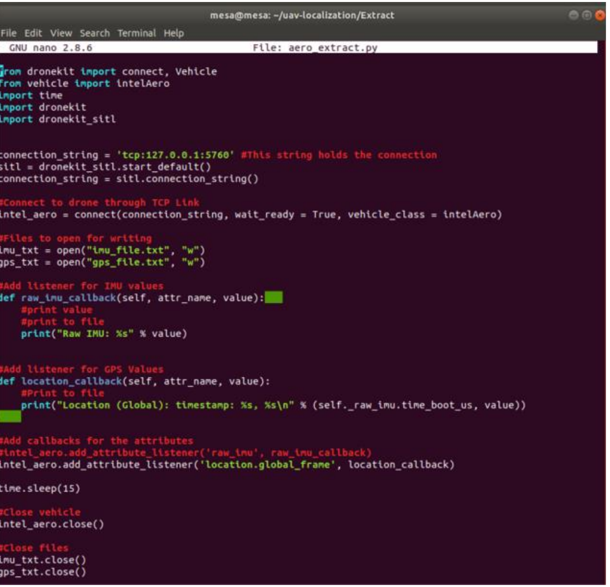
Radio

Base

[1] https://www.intel.com/content/www/us/en/products/drones/aero-ready-to-fly

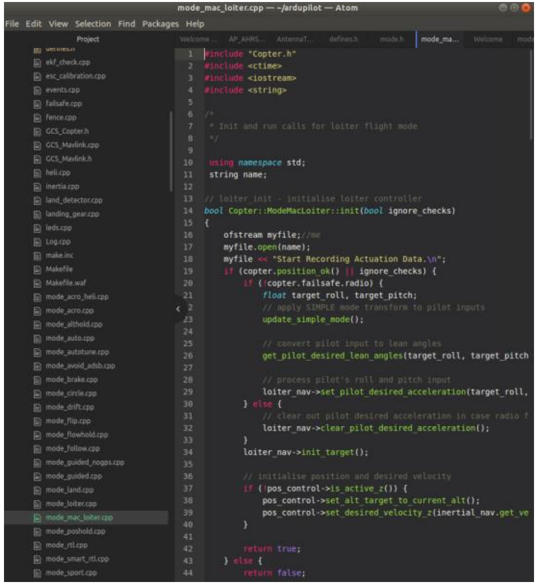
Software Implementation

Get Sensor Outputs



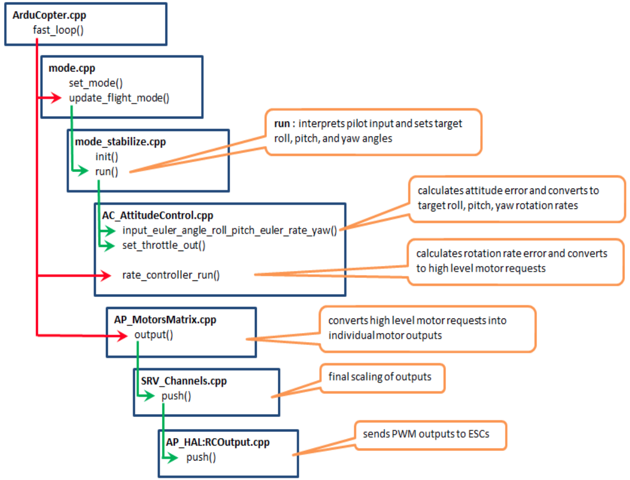
```
rom dronekit import connect, Vehicle
from vehicle import intelAero
import time
import dronekit
import dronekit_sitl


connection_string = 'tcp:127.0.0.1:5760' #This string holds the connection
sitl = dronekit_sitl.start_default()
connection_string = sitl.connection_string()

#Connect to drone through TCP Link
intel_aero = connect(connection_string, wait_ready = True, vehicle_class = intelAero)

#Files to open for writing
imu_txt = open("imu_file.txt", "w")
gps_txt = open("gps_file.txt", "w")

#Add listener for IMU values
def raw_imu_callback(self, attr_name, value):
    #print value
    #print to file
    print("Raw IMU: %s" % value)


#Add listener for GPS Values
def location_callback(self, attr_name, value):
    #Print to file
    print("Location (Global): timestamp: %s, %s\n" % (self._raw_imu.time_boot_us, value))


#Add callbacks for the attributes
#intel_aero.add_attribute_listener('raw_imu', raw_imu_callback)
intel_aero.add_attribute_listener('location.global_frame', location_callback)

time.sleep(15)

#Close vehicle
intel_aero.close()

#Close files
imu_txt.close()
gps_txt.close()
```

Design Position PID



```
#include "Copter.h"
#include <ctime>
#include <iostream>
#include <string>

/*
 * Init and run calls for loiter flight mode
 */

 using namespace std;
 string name;

// loiter_init - initialise loiter controller
bool Copter::ModeMacLoiter::init(bool ignore_checks)
{
    ofstream myfile;//me
    myfile.open(name);
    myfile << "Start Recording Actuation Data.\n";
    if (copter.position_ok() || ignore_checks) {
        if (!copter.failsafe.radio) {
            float target_roll, target_pitch;
            // apply SIMPLE mode transform to pilot inputs
            update_simple_mode();

            // convert pilot input to lean angles
            get_pilot_desired_lean_angles(target_roll, target_pitch

            // process pilot's roll and pitch input
            loiter_nav->set_pilot_desired_acceleration(target_roll,
        } else {
            // clear out pilot desired acceleration in case radio f
            loiter_nav->clear_pilot_desired_acceleration();
        }
        loiter_nav->init_target();

        // initialise position and desired velocity
        if (!pos_control->is_active_z()) {
            pos_control->set_alt_target_to_current_alt();
            pos_control->set_desired_velocity_z(inertial_nav.get_ve
        }

        return true;
    } else {
        return false;
```

Integrating in Ardupilot Hierarchy



ArduCopter.cpp
fast_loop()

mode.cpp
set_mode()
update_flight_mode()

mode_stabilize.cpp
init()
run()

run : interprets pilot input and sets target roll, pitch, and yaw angles

AC_AttitudeControl.cpp
input_euler_angle_roll_pitch_euler_rate_yaw()
set_throttle_out()
rate_controller_run()

calculates attitude error and converts to target roll, pitch, yaw rotation rates

calculates rotation rate error and converts to high level motor requests

AP_MotorsMatrix.cpp
output()

converts high level motor requests into individual motor outputs

SRV_Channels.cpp
push()

final scaling of outputs

AP_HAL:RCOutput.cpp
push()

sends PWM outputs to ESCs

[1] http://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html

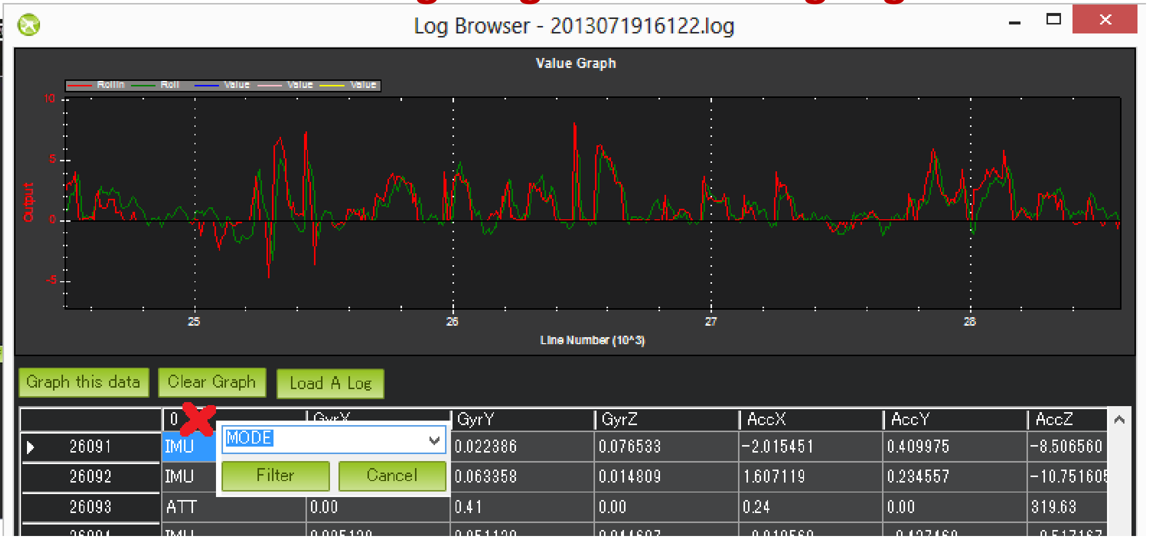
Controlling via the GUI

Defining Missions

WP Radius 2
Default Alt 100
☐ Absolute Alt
☐ Verify Height
Alt Warn 20

	Command					Lat	Long	Alt	Delete	Up	Down	Grad %	Dist	AZ
1	WAYPOINT	0	0	0	0	-35.0407928	117.8277898	100	X			95.7	104.5	1
2	WAYPOINT	0	0	0	0	-35.0406786	117.8260410	100	X			0.0	159.7	275
3	WAYPOINT	0	0	0	0	-35.0417239	117.8251612	100	X			0.0	141.2	215
4	WAYPOINT	0	0	0	0	-35.0428395	117.8259873	100	X			0.0	145.1	149
5	WAYPOINT	0	0	0	0	-35.0427165	117.8274572	100	X			0.0	134.5	84

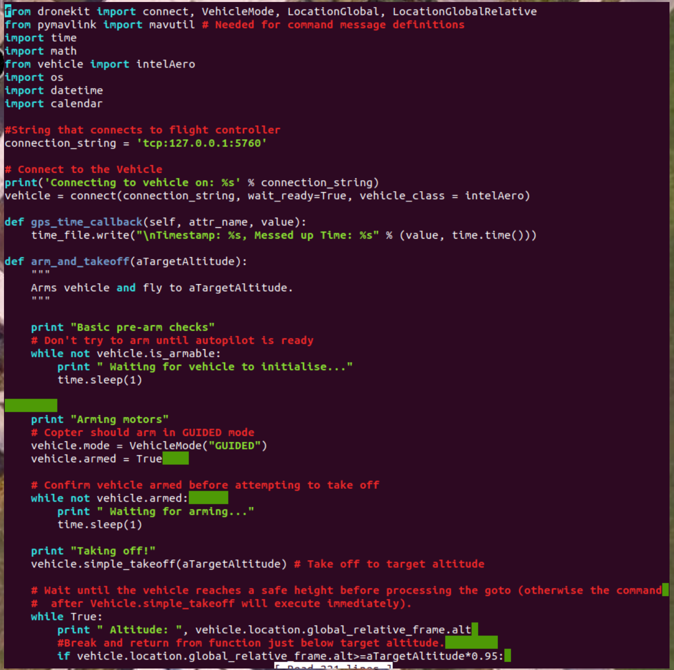
Configuring and Tweaking Logs



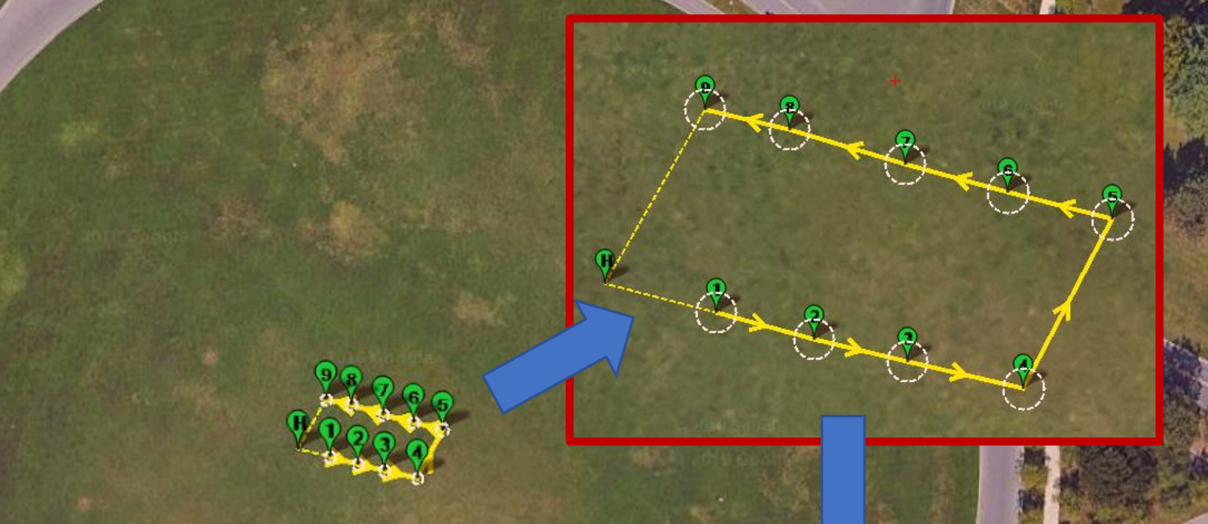
MODE

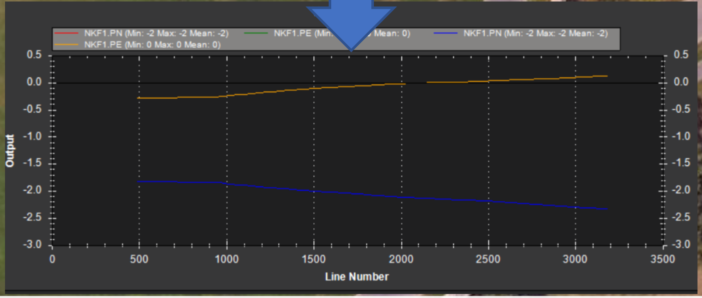
	0	GyrX	GyrY	GyrZ	AccX	AccY	AccZ
26091	IMU		0.022386	0.076533	−2.015451	0.409975	−8.506560
26092	IMU		0.063358	0.014809	1.607119	0.234557	−10.75160
26093	ATT	0.00	0.41	0.00	0.24	0.00	319.63

Controlling via the Script



```
from dronekit import connect, VehicleMode, LocationGlobal, LocationGlobalRelative
from pymavlink import mavutil # Needed for command message definitions
import time
import math
from vehicle import intelAero
import os
import datetime
import calendar

#String that connects to flight controller
connection_string = 'tcp:127.0.0.1:5760'

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True, vehicle_class = intelAero)

def gps_time_callback(self, attr_name, value):
    time_file.write("\nTimestamp: %s, Messed up Time: %s" % (value, time.time()))

def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print "Basic pre-arm checks"
    # Don't try to arm until autopilot is ready
    while not vehicle.is_armable:
        print " Waiting for vehicle to initialise..."
        time.sleep(1)


    print "Arming motors"
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    # Confirm vehicle armed before attempting to take off
    while not vehicle.armed:
        print " Waiting for arming..."
        time.sleep(1)

    print "Taking off!"
    vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude

    # Wait until the vehicle reaches a safe height before processing the goto (otherwise the command
    #  after Vehicle.simple_takeoff will execute immediately).
    while True:
        print " Altitude: ", vehicle.location.global_relative_frame.alt
        #Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
```





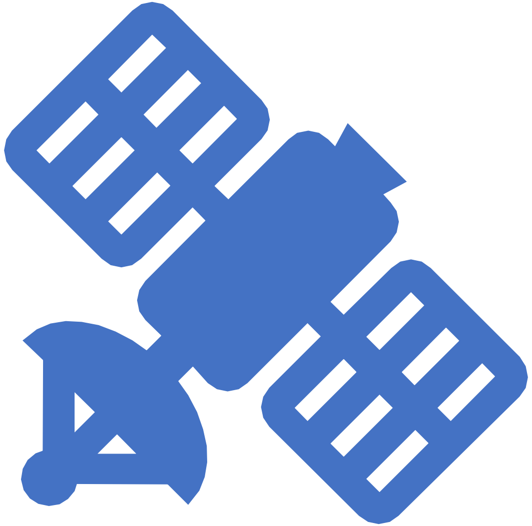
Sensors

POSITION SENSORS

ATTITUDE SENSORS

ALTITUDE SENSORS

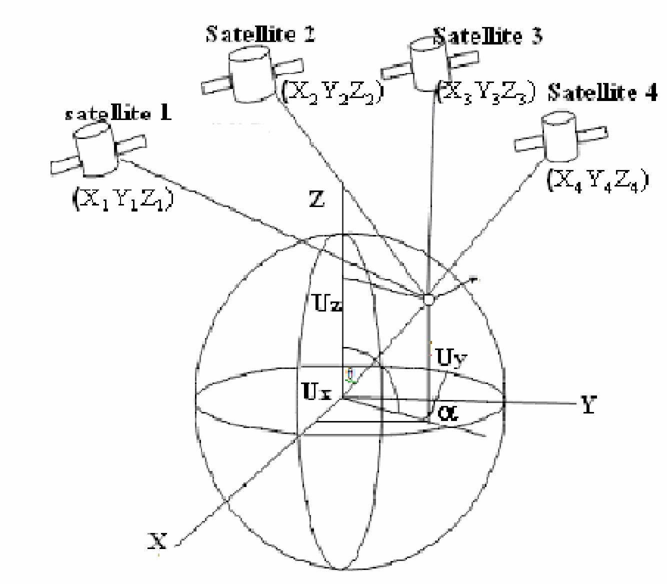
- IMU (Accelerometer, Gyroscope, Magnetometer)
- GPS
- Barometer
- Camera
- Radio

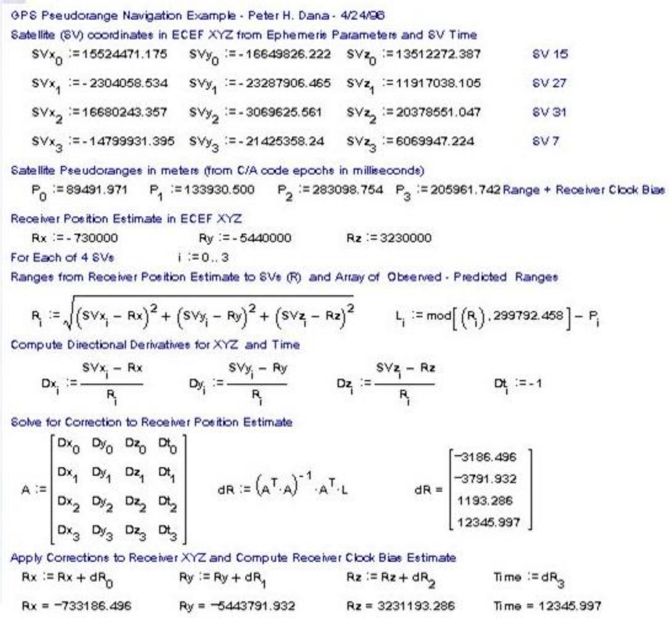


Position Sensing

- GNSS
- INS
- EKF (GNSS + INS)
- RTK-GPS

GNSS





GPS Pseudorange Navigation Example - Peter H. Dana - 4/24/96

Satellite (SV) coordinates in ECEF XYZ from Ephemeris Parameters and SV Time

$SVx_0 := 15524471.175$ $SVy_0 := -16649826.222$ $SVz_0 := 13512272.387$ SV 15

$SVx_1 := -2304058.534$ $SVy_1 := -23287906.465$ $SVz_1 := 11917038.105$ SV 27

$SVx_2 := 16680243.357$ $SVy_2 := -3069625.561$ $SVz_2 := 20378551.047$ SV 31

$SVx_3 := -14799931.395$ $SVy_3 := -21425358.24$ $SVz_3 := 6069947.224$ SV 7

Satellite Pseudoranges in meters (from C/A code epochs in milliseconds)

$P_0 := 89491.971$ $P_1 := 133930.500$ $P_2 := 283098.754$ $P_3 := 205961.742$ Range + Receiver Clock Bias

Receiver Position Estimate in ECEF XYZ

$Rx := -730000$ $Ry := -5440000$ $Rz := 3230000$

For Each of 4 SVs $i := 0..3$

Ranges from Receiver Position Estimate to SVs (R) and Array of Observed - Predicted Ranges

$$R_i := \sqrt{(SVx_i - Rx)^2 + (SVy_i - Ry)^2 + (SVz_i - Rz)^2} \qquad L_i := \mathrm{mod}\left[(R_i), 299792.458\right] - P_i$$

Compute Directional Derivatives for XYZ and Time

$$Dx_i := \frac{SVx_i - Rx}{R_i} \qquad Dy_i := \frac{SVy_i - Ry}{R_i} \qquad Dz_i := \frac{SVz_i - Rz}{R_i} \qquad Dt_i := -1$$

Solve for Correction to Receiver Position Estimate

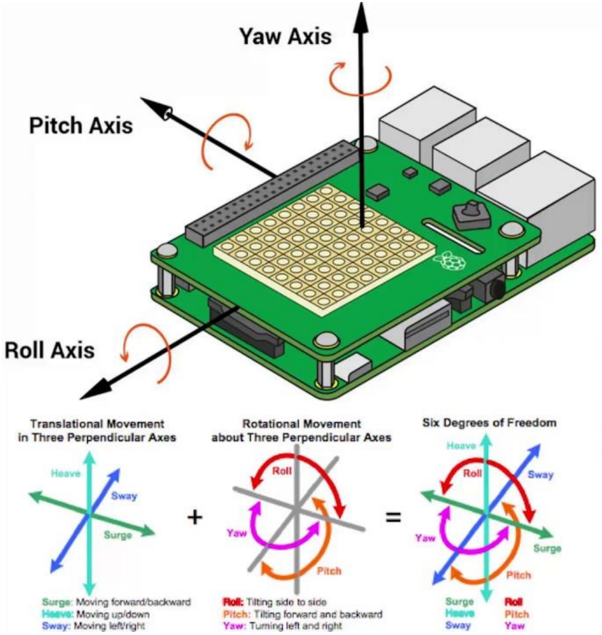
$$A := \begin{bmatrix} Dx_0 & Dy_0 & Dz_0 & Dt_0 \\ Dx_1 & Dy_1 & Dz_1 & Dt_1 \\ Dx_2 & Dy_2 & Dz_2 & Dt_2 \\ Dx_3 & Dy_3 & Dz_3 & Dt_3 \end{bmatrix} \qquad dR := \left(A^T \cdot A\right)^{-1} \cdot A^T \cdot L \qquad dR = \begin{bmatrix} -3186.496 \\ -3791.932 \\ 1193.286 \\ 12345.997 \end{bmatrix}$$

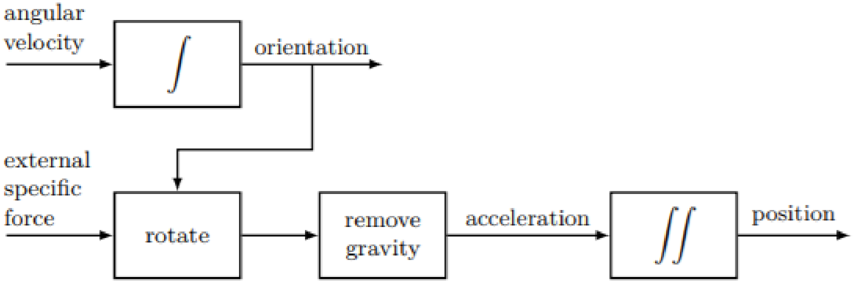
Apply Corrections to Receiver XYZ and Compute Receiver Clock Bias Estimate

$Rx := Rx + dR_0$ $Ry := Ry + dR_1$ $Rz := Rz + dR_2$ $Time := dR_3$

$Rx = -733186.496$ $Ry = -5443791.932$ $Rz = 3231193.286$ $Time = 12345.997$

INS





angular velocity

$\int$

orientation

external specific force

rotate

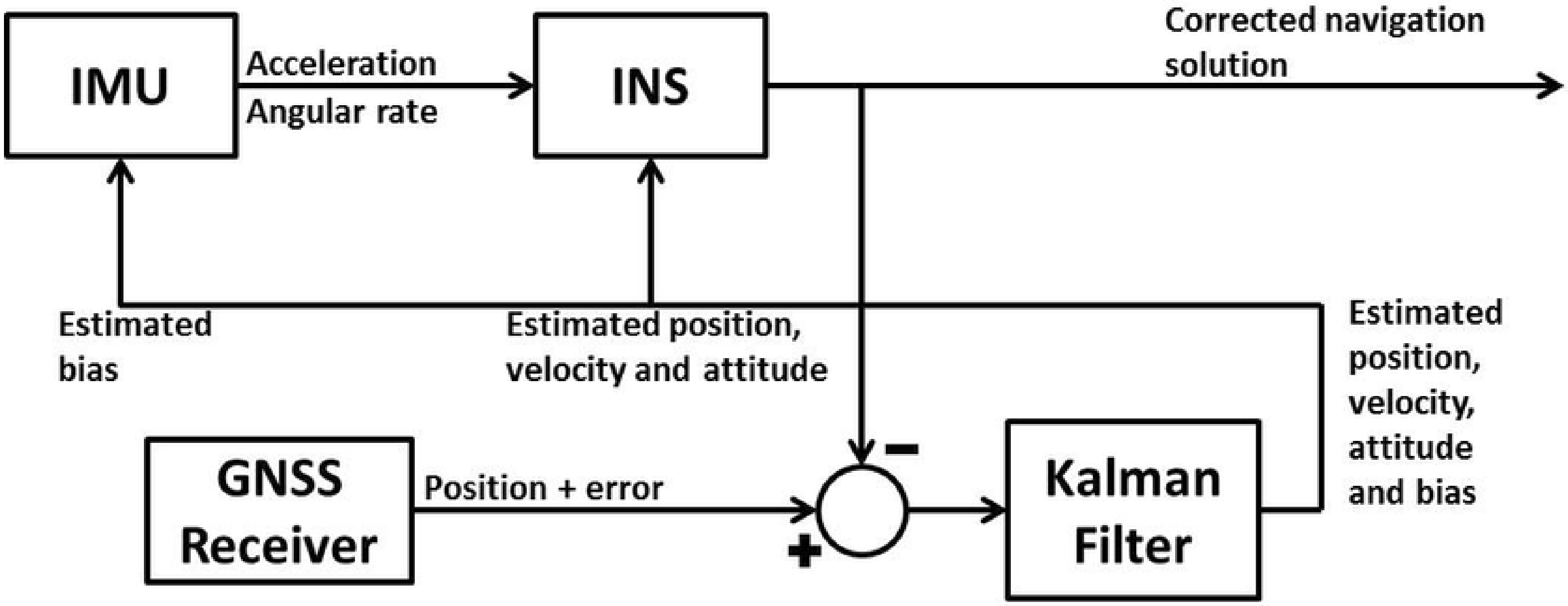
remove gravity

acceleration

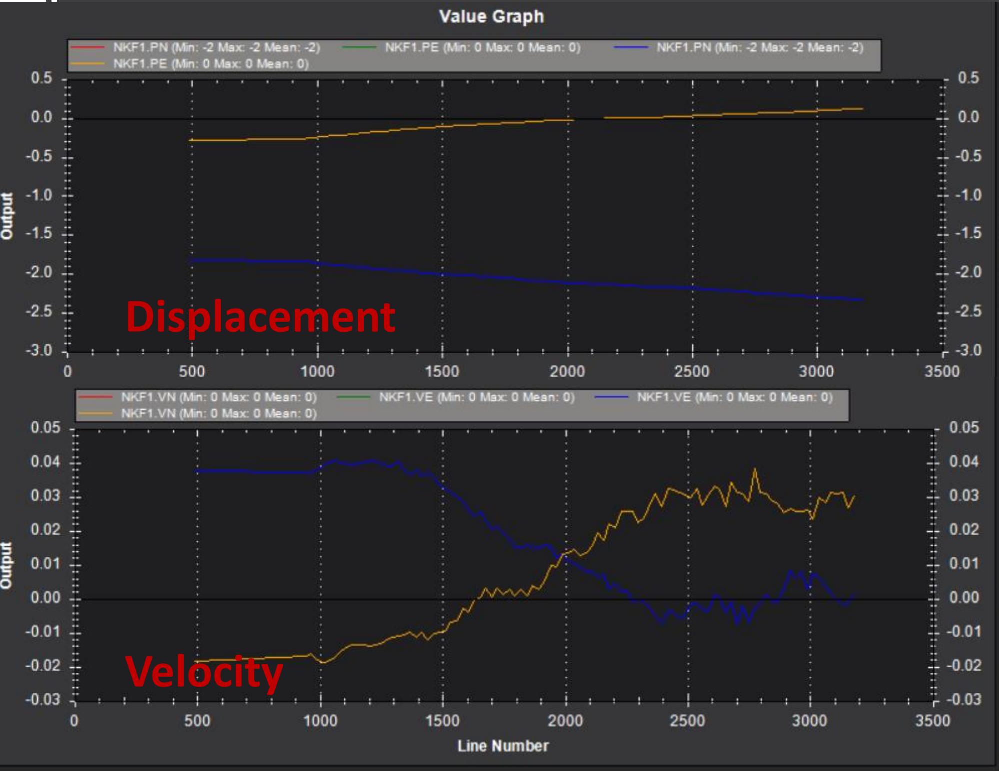
$\iint$

position

Extended Kalman Filtering (EKF)



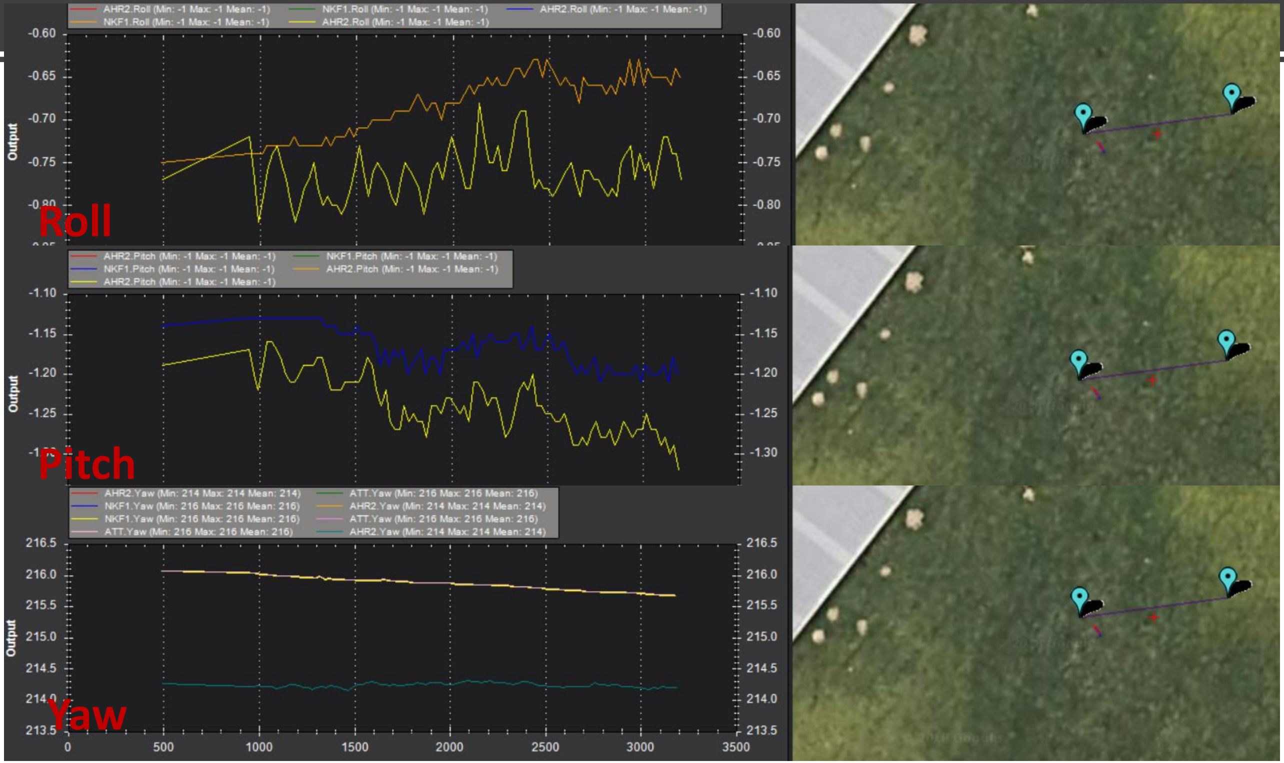
EKF Positions and Velocities



Displacement

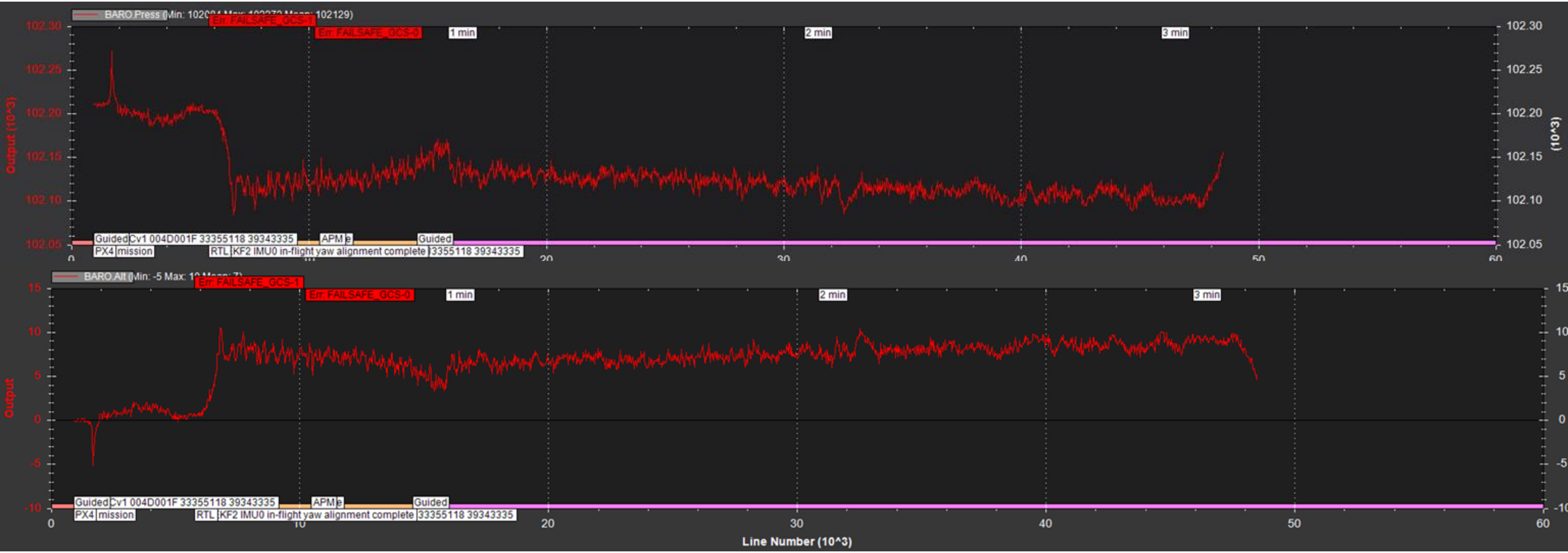
Velocity

EKF Attitude

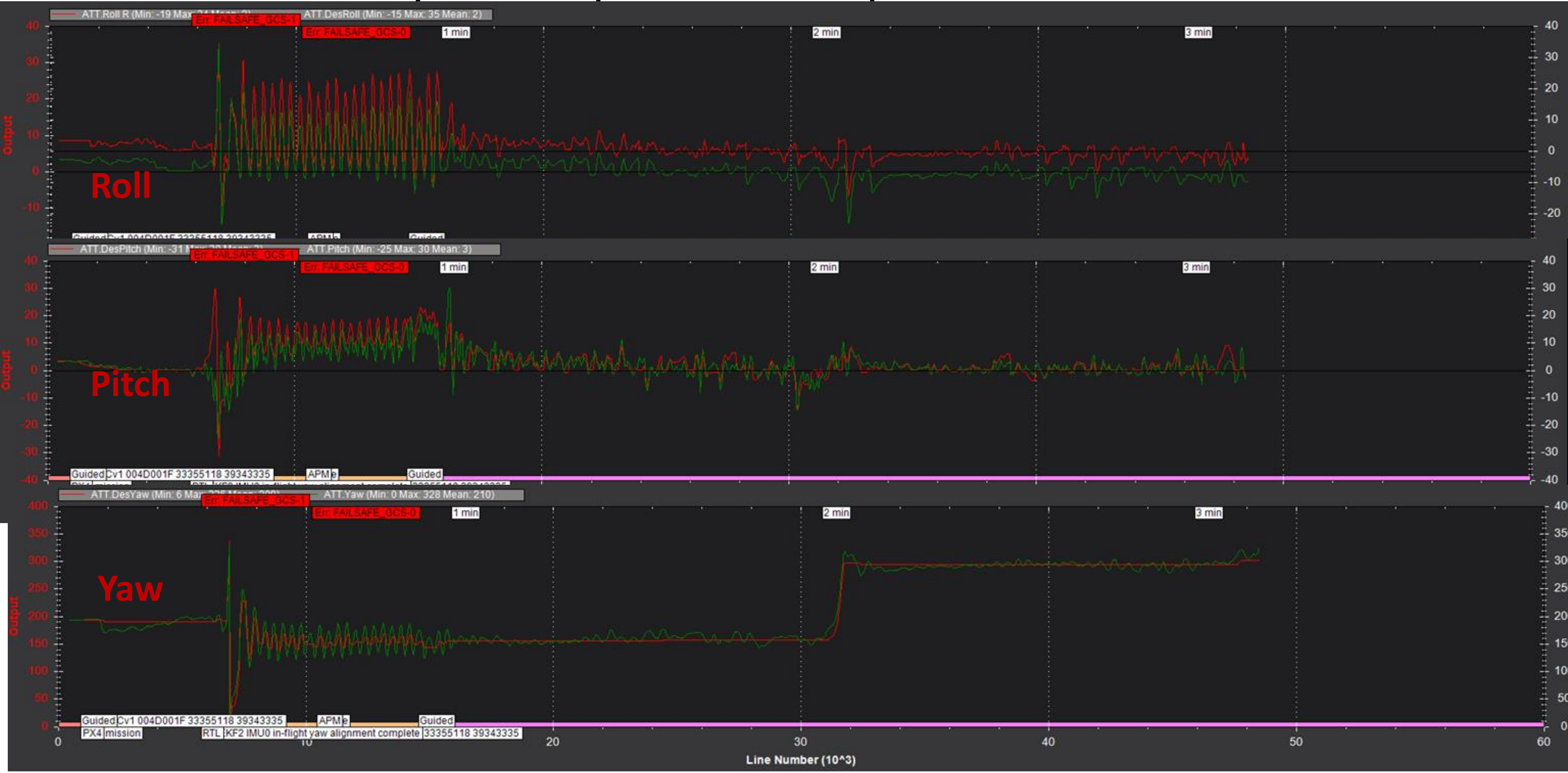


EKF Altitude

Sensitivity: 0.3m change in altitude

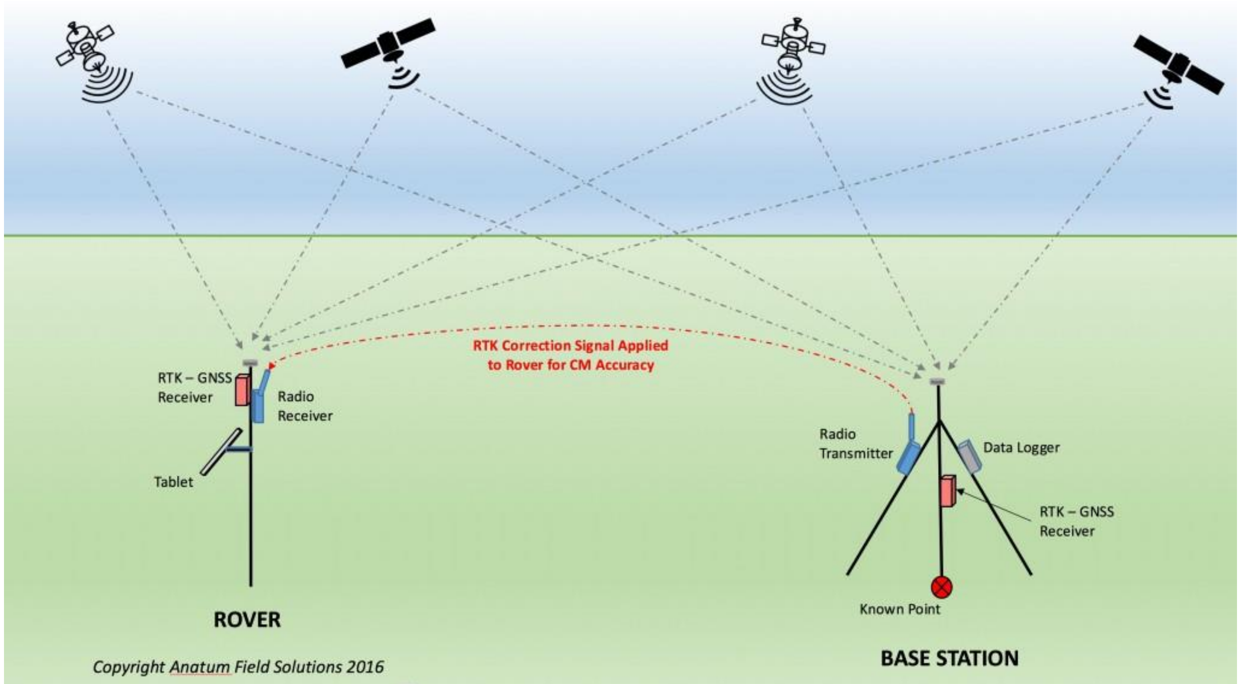


Attitude - Gyroscope + Compass



RTK GPS

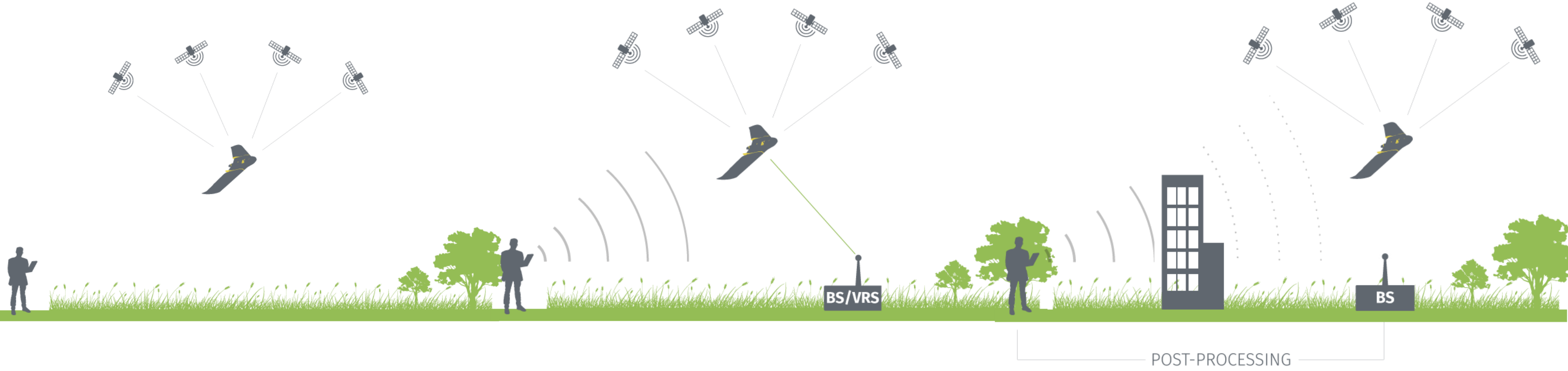
Precision: 0.1 mm → Accuracy: 2.5 cm (From 2.5 m!)



Copyright Anatum Field Solutions 2016

RTK, Penn State University, Dept. of Geography

RTK Advantage



Typical GNSS

2.5m Accuracy

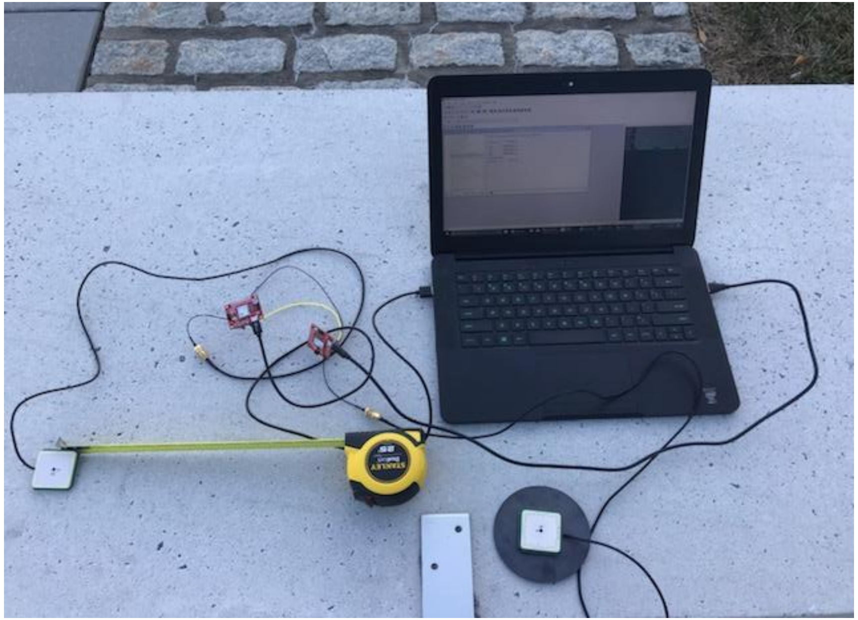
Real time Kinematics (RTK)

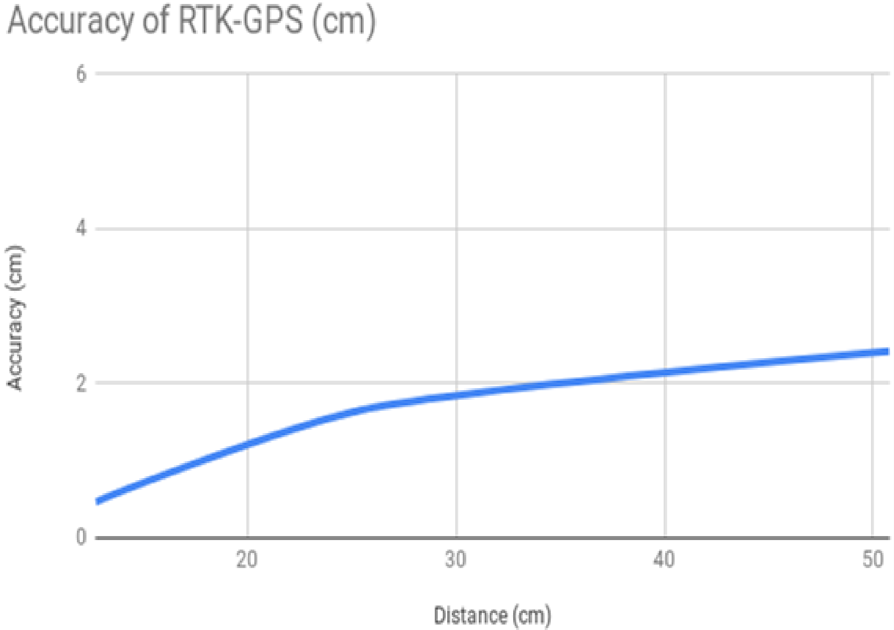
2.5cm Accuracy

Post-processing RTK

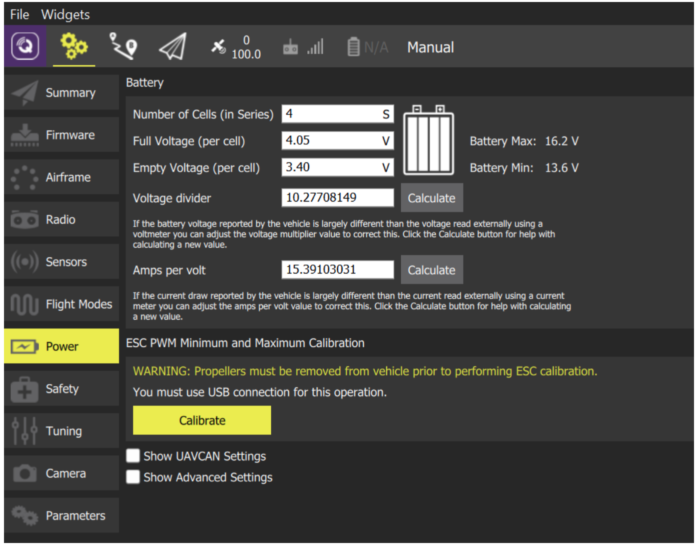
10cm Accuracy

RTK Accuracy





Calibration



Battery

Number of Cells (in Series): 4 S

Full Voltage (per cell): 4.05 V

Empty Voltage (per cell): 3.40 V

Battery Max: 16.2 V

Battery Min: 13.6 V

Voltage divider: 10.27708149

If the battery voltage reported by the vehicle is largely different than the voltage read externally using a voltmeter you can adjust the voltage multiplier value to correct this. Click the Calculate button for help with calculating a new value.

Amps per volt: 15.39103031

If the current draw reported by the vehicle is largely different than the current read externally using a current meter you can adjust the amps per volt value to correct this. Click the Calculate button for help with calculating a new value.

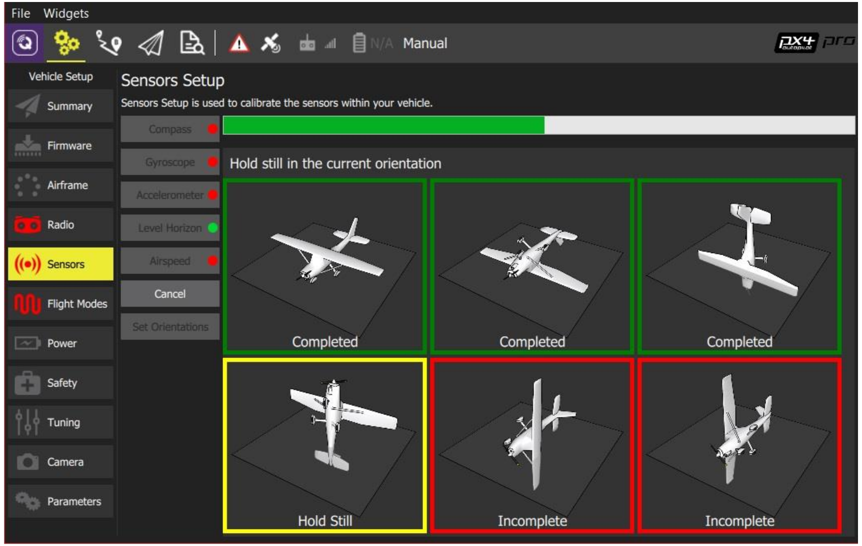
ESC PWM Minimum and Maximum Calibration

WARNING: Propellers must be removed from vehicle prior to performing ESC calibration.

You must use USB connection for this operation.

☐ Show UAVCAN Settings

☐ Show Advanced Settings



Sensors Setup

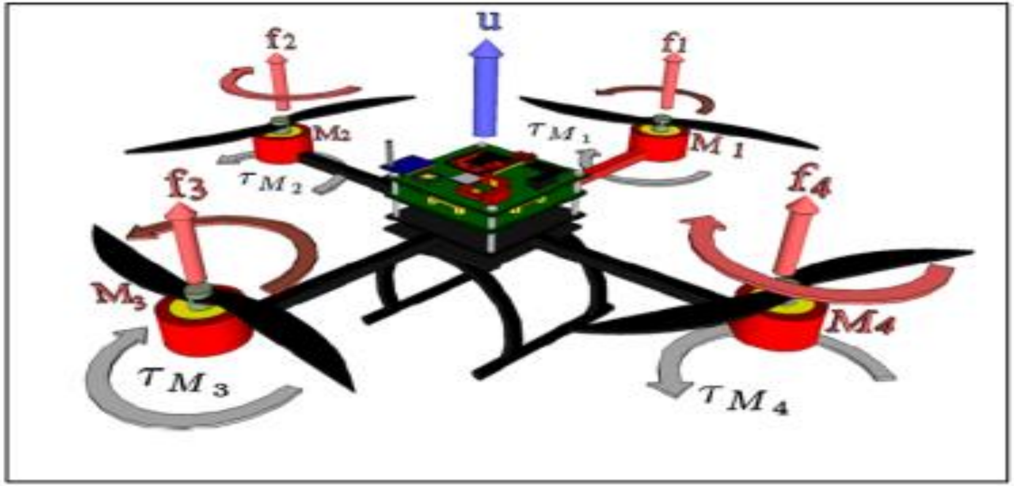
Sensors Setup is used to calibrate the sensors within your vehicle.

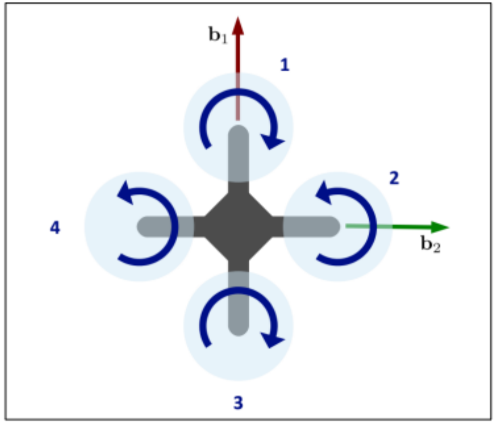
Hold still in the current orientation

Electronic Speed Control

Compass

Accelerometer





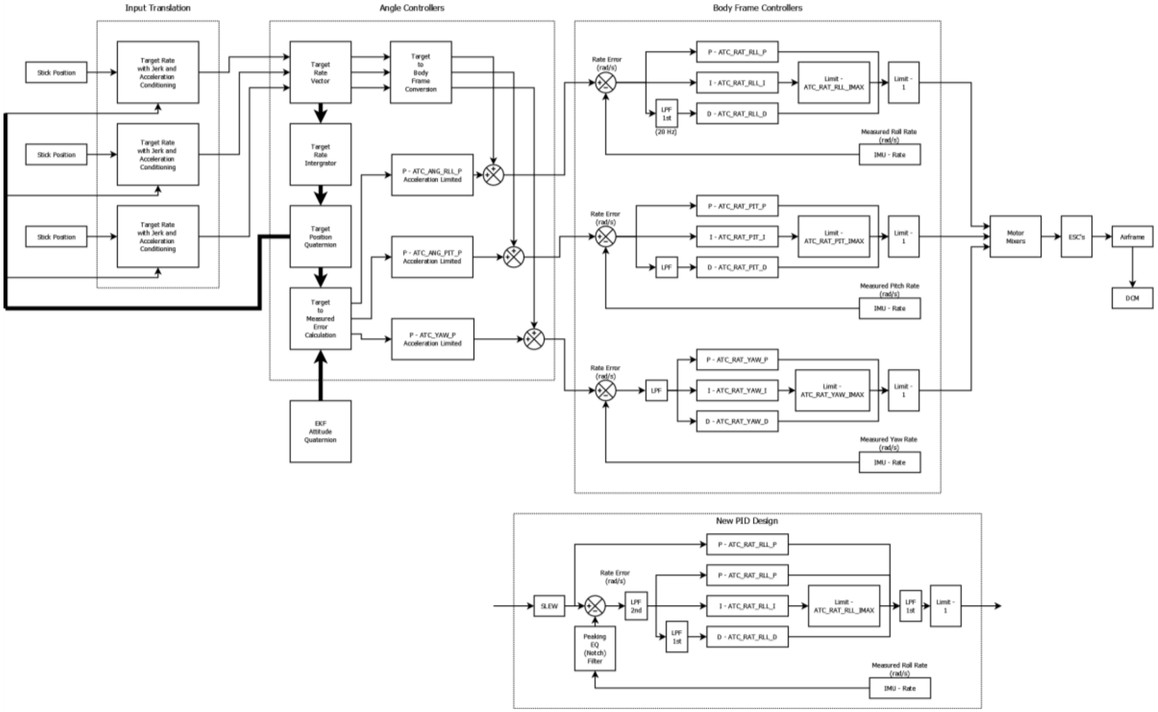
$$f_i = k\omega_i^2$$

Modeling the Dynamics

Controller

- The drone controller consists of a punch of different PID controllers.
- These PID controllers are responsible for stability, angle correction for manual control, displacement PID controller for navigation .
- The drone controller has a complex design in order to deal with different types of PID controllers.

Stabilize, Roll, Pitch & Yaw PID’s



[1] https://discuss.ardupilot.org

Controller

- The complexity of the controller design can be reduce by remodel the controller as:
  - Stabilize PID controllers (Row, Yaw and Pitch PID controllers)

  - Displacement PID controller

Stabilize PID controller

- It is responsible for maintaining the drone for a given position from the manual controller.
- It consists of three PID controllers for the three directions (Row, Yaw, Pitch).
- The input for the PID controllers is the manual input rate for the different directions as the output of the angular controllers.
- The angular controllers are fed from the pilot joystick by the target position.

Stabilize, Roll, Pitch & Yaw PID’s

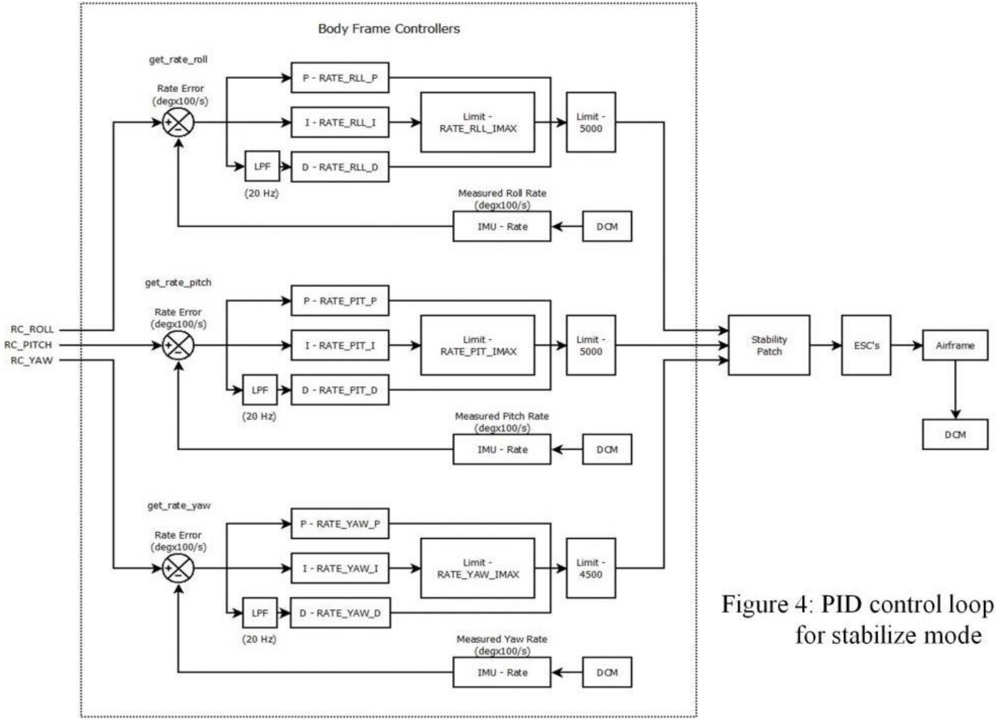
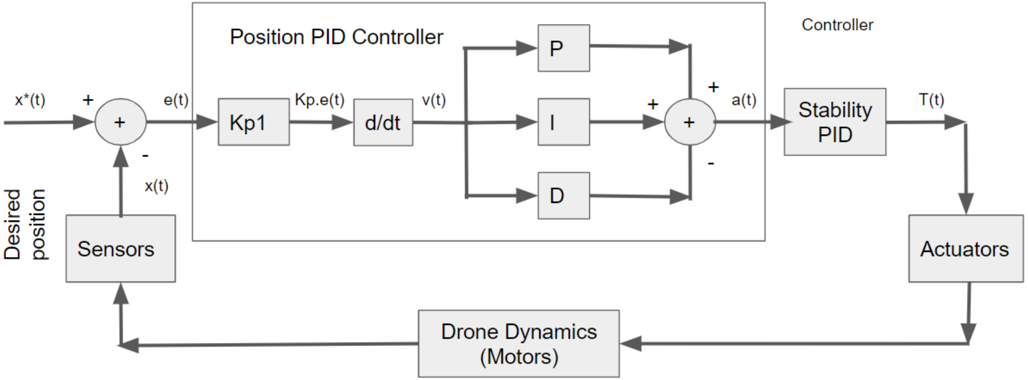


Figure 4: PID control loop for stabilize mode

[1] https://diydrones.com/forum/topics/stabilize-mode-like-loiter-mode

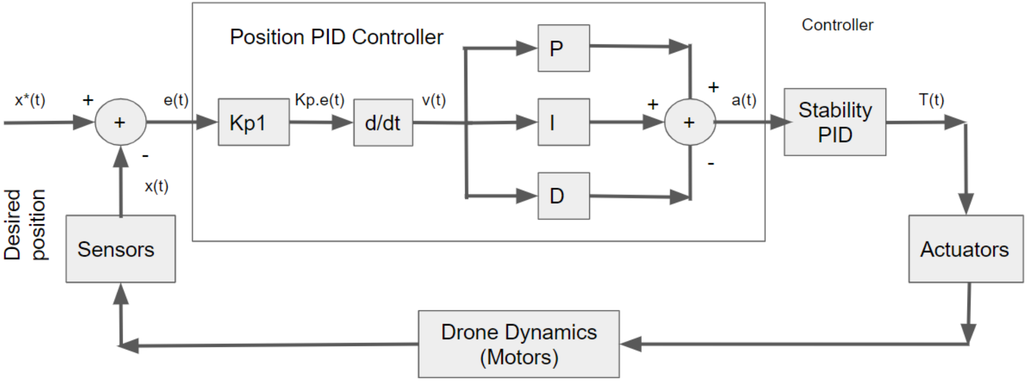
PID controller Design

- The Position PID controller consists of 2 cascade controllers one is proportional while the other is PID controller.
- The controller is fed by the desired position and the actual position of the drone.
- The first is proportional controller which uses the error in the position (i.e the difference between the desired and current locations) and converts it to desired speed.

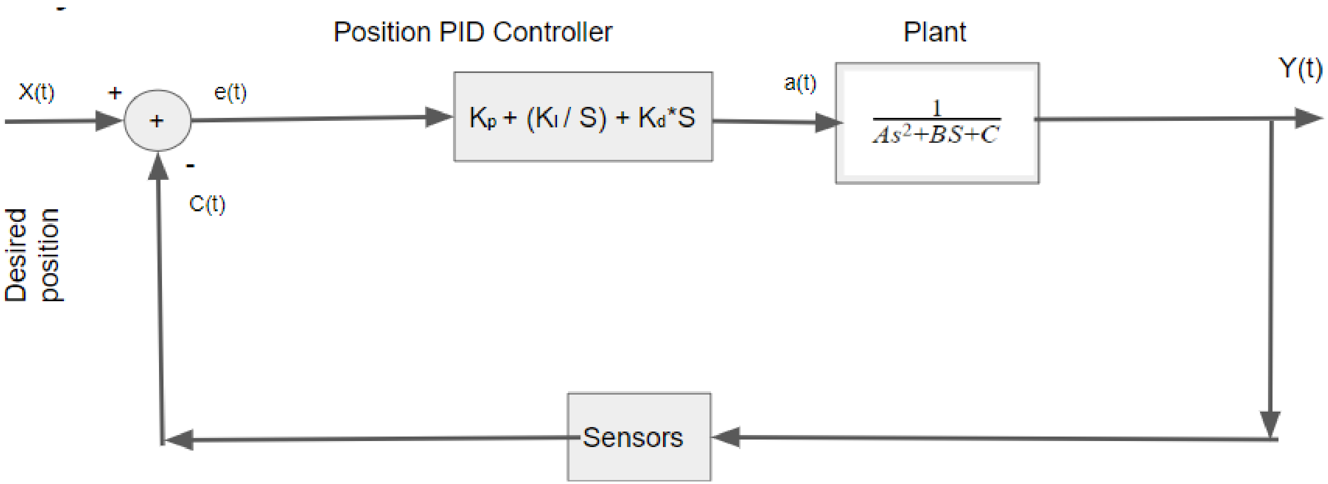


PID controller Design

- The desired speed is the input for the second PID controller which is converts it to desired acceleration.
- The resulting desired acceleration becomes a lean angle which is then passed to the stability PID controller to regulate the angles.
- The output from the stability PID controllers is the torques to drive the actuators (motors).



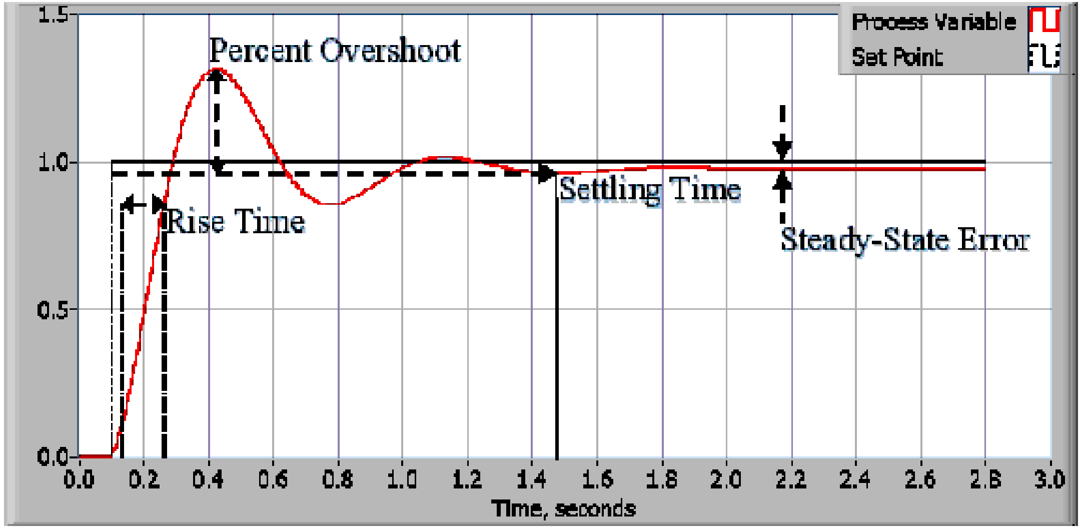
System Model



$$\frac{Y(s)}{X(s)} = \frac{K_d s^2+K_p S+K_I}{As^3+(B+K_d)s^2+(C+K_p)S+K_I}$$

PID Step Response

- Stability
- Responsiveness



[] http://www.ni.com/white-paper/3782/en/

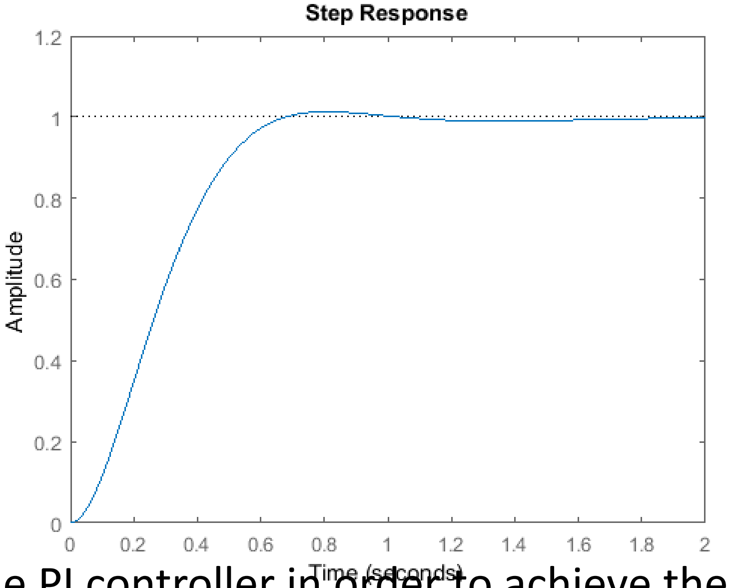
PID Tuning

- Proportional part: It makes fast responsiveness for errors, but as much it is increase, the system suffers from **Overshooting**.

- Derivative part: Prediction of the future through the derivative of the current error, however it is **Noise sensitive**.

- Integral Part: it make the system stable, however it is **Slow**.

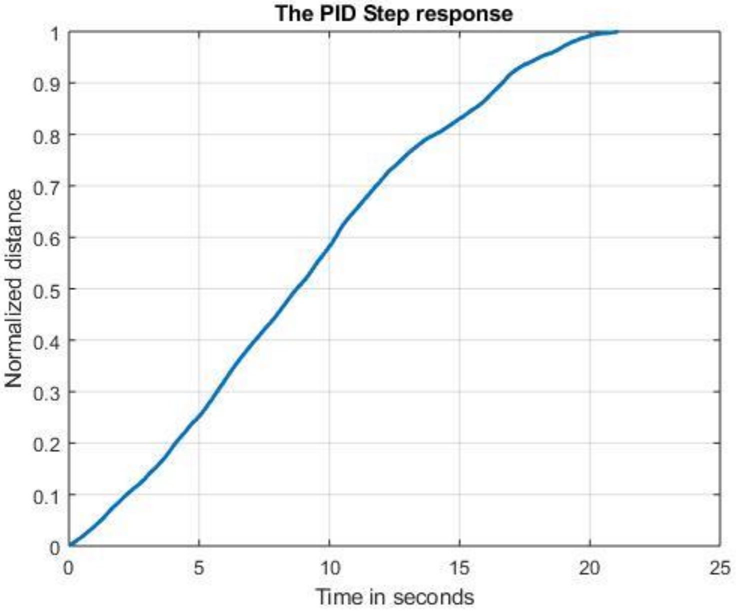
PID Tuning



- We will use PI controller in order to achieve the stability rather that the responsiveness.

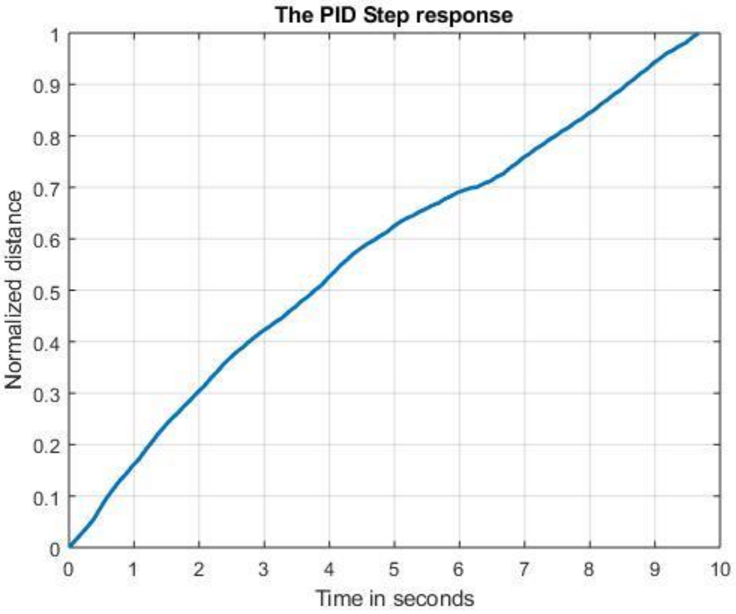
PID Tuning

- We Evaluate the auto tuning of the PID controller.
- Kp=1
- KI=0.5
- Kd=0



PID Tuning

- We Evaluate the auto tuning of the PID controller.
- Kp=1
- KI=0.5
- Kd=0



Communication architecture

- There are different communication protocols to transfer the data between the different modules.
- The flight controller captures the data of the position from the IMU using SPI protocol
- Also the motor outputs transferred by UART protocol to the FBGA which fed the motor by PWM input.

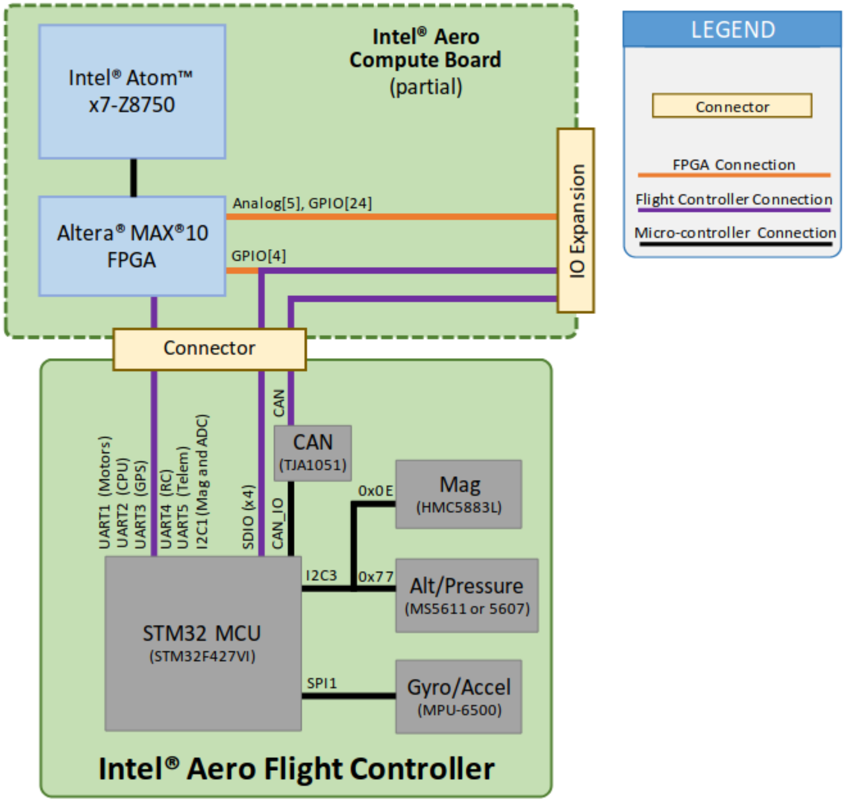


Figure 11. Hardware Block Diagram – Aero Flight Controller

[1]https://www.intel.com/content/dam/support/us/en/documents/drones/development-drones/intel-aero-compute-board-guide.pdf

Analysis Framework

MODELING

DESIGN (EXPERIMENTS, MODIFYING EXISTING SCRIPTS, ADDING NEW SCRIPTS)

TESTING AND VERIFYING

Limitations

- Weather.
- Battery lifetime.

Future Work

- Try to see the effect of changing the proportional gain of the PID controller and use cause and effect analysis to analyze the PID controller.

Conclusions

- Achieving Cm level Sensing.
- Control the drone movement within Cm Accuracy.
- Analyze the behavior of the PID controller, and the way we use for tuning.

Demo