documentation\digitalDevLogs\IMU.cpp

```cpp
//includes
#include <main.h>
#include <math.h>
#include <iomanip>
#include <array>
#define gravity 32.147 //ft/sec^2
#define imu_port 3

//namespaces
using std::array;
using std::string;
using std::stringstream;
using std::setprecision;
using std::fixed;

//global variables for storage between functions
float x = 0; //ft
float y = 0; //ft
float vx = 0; //ft per second
float vy = 0; //ft per second
float prev_vx = 0; //ft per second
float prev_vy = 0; //ft per second
float prev_ax = 0; //ft per second^2
float prev_ay = 0; //ft per second^2
lv_obj_t *statusLabel;
lv_obj_t *imuButton;


//inertial function
array<float, 8> inertialFunction(pros::IMU imu, int time_delay){
    //calculate delta time in seconds using time delay in milliseconds
    float delta_time = time_delay/1000;

    //get acceleration data
    pros::c::imu_accel_s_t accel = imu.get_accel();
    float ax = accel.x;
    float ay = accel.y;

    //recalculate accelerations in ft/sec^2
    ax *= gravity;
    ay *= gravity;

    //update velocities using trapezoidal riemann sum by:
    //   adding previous acceleration(changing it from f(x(n)) to 2f(x(n-1)))
    //   current acceleration
    //   multiplied both by delta x/2 before addition
    //   with the formula: F(x) = ▲x(f(x(0))+2f(x1)+...+2f(x(n-1))+f(x(n)))/2
    vx += delta_time*(prev_ax+ax)/2;
    vy += delta_time*(prev_ax+ax)/2;

    //update previous accelerations
    prev_ax = ax;
```

```cpp
    prev_ay = ay;

    //update positions using trapezoidal riemann sum by:
    //   adding previous velocity(changing it from f(x(n)) to 2f(x(n-1)))
    //   current velocity
    //   multiplied both by delta x/2 before addition
    //   with the formula: F(x) = ▲x(f(x(0))+2f(x1)+...+2f(x(n-1))+f(x(n)))/2
    x += delta_time*(prev_vx+vx)/2;
    y += delta_time*(prev_vy+vy)/2;

    //calculate overall velocity/acceleration using the following formula:
    //   p^2=x^2+y^2 -> 2p*dp/dt = 2x*dx/dt + 2y*dy/dt -> 2p*(d^2p/(dt^2)) + 2*(dp/dt)*(dp/dt) = 2x*(d^2x/(dt^2)) + 2*(dx/dt)*(dx/dt) + 2y*(d^2y/(dt^2)) + 2*(dy/dt)*(dy/dt)
    //   In this case we don't need absolute position(or displacement from (0,0)), but dp/dt is our overall velocity, and d^2p/dt^2 is our overall acceleration
    //   For ease:
    //   p = displacement, dp/dt = velocity or v, d^2*p/dt^2 = acceleration or a
    //   x = displacementX, dx/dt = velocityX or vx, d^2*x/dt^2 = accelerationX or ax
    //   y = displacementY, dy/dt = velocityY or vy, d^2*y/dt^2 = accelerationY or ay,
    float p = pow(pow(x,2)+pow(y,2),0.5);
    float v = (x*vx + y*vy)/p;
    float a = (pow(vx,2) + x*ax + pow(vy,2) + y*ay - pow(v,2))/p;

    //create array and return array
    array<float, 8> va_pvax_pvay = {v, a, x, vx, ax, y, vy, ay};
    return va_pvax_pvay;
}
```