```cpp
#include "UserCode.hpp"
#include "UtilityFunctions.hpp"
#include "Vec3f.hpp"
#include <cmath>
#include <stdio.h> //for printf
#include <Eigen/Dense>

using namespace Eigen;

//We keep the last inputs and outputs around for debugging:
MainLoopInput lastMainLoopInputs;
MainLoopOutput lastMainLoopOutputs;

//Some constants that we may use:
const float mass = 32e-3f;  // mass of the quadcopter [kg]
const float gravity = 9.81f;  // acceleration of gravity [m/s^2]
const float inertia_xx = 16e-6f;  //MMOI about x axis [kg.m^2]
const float inertia_yy = inertia_xx;  //MMOI about y axis [kg.m^2]
const float inertia_zz = 29e-6f;  //MMOI about z axis [kg.m^2]

const float dt = 1.0f / 500.0f;  //[s] period between successive calls to MainLoop

int t = 0; //counter for flight control

//Added initialization of gyro bias
Vec3f estGyroBias = Vec3f(0,0,0);

//Added initialization of integrator variables
float estRoll = 0; //Roll estimate
float estPitch = 0; //Pitch estimate
float estYaw = 0; //Yaw estimate
float phi_meas = 0; //Roll angle
float theta_meas = 0; //Pitch angle
float estHeight = 0; //Height Estimate
float desHeight = 0; //Desired height
float estVelocity_1 = 0; // Velocity in 1 Direction
float estVelocity_2 = 0; // Velocity in 2 Direction
float estVelocity_3 = 0; // Velocity in 3 Direction
float estPos_1 = 0; //Position in 1 Direction
float estPos_2 = 0; //Position in 2 Direction
float lastHeightMeas_meas = 0; //measurement of last height value
float lastHeightMeas_time = 0; //time when last height meas was taken

// Input for desired attitude and position
Vec3f des_attitude = Vec3f(0,0,0);
Vec3f desPos = Vec3f(0,0,0);
float desAcc1 = 0;
```

```cpp
float desAcc2 = 0;

//Time Constants
float const timeConstant_rollAngle = 0.10f;                     //roll angle
float const timeConstant_pitchAngle = timeConstant_rollAngle;   //pitch angle
float const timeConstant_yawAngle = 0.2f;                       //yaw angle
float const timeConst_horizVel = 100.0f;                        //horizontal velocity
float const timeConstant_rollrate = 0.02f;                      //roll rate
float const timeConstant_pitchrate = timeConstant_rollrate;     //pitch rate
float const timeConstant_yawrate = 0.1f;                        //yaw rate

const float natFreq_height = 2.0f;      //height natural frequency
const float dampingRatio_height = 0.7f; //height damping ratio

MainLoopOutput MainLoop(MainLoopInput const &in) {

//Define the output numbers (in the struct outVals):
  MainLoopOutput outVals;

//  motorCommand1 -> located at body +x +y
//  motorCommand2 -> located at body +x -y
//  motorCommand3 -> located at body -x -y
//  motorCommand4 -> located at body -x +y

//Added calculation for rate gyro bias and correction
  if (in.currentTime < 1.0f) {
    estGyroBias = estGyroBias + (in.imuMeasurement.rateGyro/500.0f); //bias calculation
  }
  Vec3f rateGyro_corr = in.imuMeasurement.rateGyro - estGyroBias; // correction calculation

//Added Integrator Estimations
  int p = 0.05; //Assign trade-off scalar
  phi_meas = in.imuMeasurement.accelerometer.y/gravity; //Roll angle calculation
  theta_meas = -in.imuMeasurement.accelerometer.x/gravity; //Pitch angle calculation
  estRoll = (1-p)*(estRoll + dt*rateGyro_corr.x) + p*phi_meas; //Roll estimate calculation
  estPitch = (1-p)*(estPitch + dt*rateGyro_corr.y) + p*theta_meas; //Pitch estimate calculation
  estYaw = estYaw + dt*rateGyro_corr.z; //Yaw estimate calculation

  outVals.motorCommand1 = 0;
  outVals.motorCommand2 = 0;
  outVals.motorCommand3 = 0;
```

```
  outVals.motorCommand4 = 0;

//Added assignment of rate gyro measurements to telemetry outputs
  outVals.telemetryOutputs_plusMinus100[10]=lastMainLoopInputs.\
      imuMeasurement.rateGyro.x; //roll angular velocity
  outVals.telemetryOutputs_plusMinus100[11]=lastMainLoopInputs.\
      imuMeasurement.rateGyro.y; //pitch angular velocity

//Start of Angle Control---------------------------------------------------

  //Log desired pitch angle in telemetry outputs.
  outVals.telemetryOutputs_plusMinus100[9] = 0;

  //Calculate angular velocity commands to be fed is des_ang_vel in rate control calculation
  float cmdAngVelRoll = (-1.0f/timeConstant_rollAngle)*(estRoll - des_attitude.x);
  float cmdAngVelPitch = (-1.0f/timeConstant_pitchAngle)*(estPitch - des_attitude.y);
  float cmdAngVelYaw = (-1.0f/timeConstant_yawAngle)*(estYaw - des_attitude.z);



//End of Angle Control-----------------------------------------------------

//Start of Rate Control----------------------------------------------------

  //Added input for desired angular velocity
  Vec3f des_ang_vel = Vec3f(cmdAngVelRoll,cmdAngVelPitch,cmdAngVelYaw);

  //Calculate angular acceleration commands to be fed is des_ang_accel in mixer calculation
  float cmdAngAcclRoll = (-1.0f/timeConstant_rollrate)*(rateGyro_corr.x - des_ang_vel.x);
  float cmdAngAcclPitch = (-1.0f/timeConstant_pitchrate)*(rateGyro_corr.y - des_ang_vel.y);
  float cmdAngAcclYaw = (-1.0f/timeConstant_yawrate)*(rateGyro_corr.z - des_ang_vel.z);

//End of Rate Control------------------------------------------------------

// Vertical Estimation Control -----------------------------------------------

  // Prediction Step
  estHeight = estHeight + estVelocity_3 * dt; //height change with vertical velocity
```

```
  estVelocity_3 = estVelocity_3 + 0*dt; //assumed constant

  //Correction Step
  float const mixHeight = 0.3f;
  if (in.heightSensor.updated) {
    //check that measurement is reasonable
    if (in.heightSensor.value < 5.0f) {
      float hMeas = in.heightSensor.value * cosf(estRoll) * cosf(estPitch); //cosf gives float value of cosine
      estHeight = (1-mixHeight) * estHeight + mixHeight * hMeas;

      float v3Meas = (hMeas - lastHeightMeas_meas) / (in.currentTime - lastHeightMeas_time);

      estVelocity_3 = (1-mixHeight) * estVelocity_3 + mixHeight * v3Meas; //***vertical velocity value
      lastHeightMeas_meas = hMeas;
      lastHeightMeas_time = in.currentTime;
    }
  }

//-------------------------------------------------------------------------------

//Horizontal Estimation Control

//  //Prediction Step
  estPos_1 = estPos_1 + estVelocity_1*dt;
  estPos_2 = estPos_1 + estVelocity_2*dt;

  //velocity feedback with desired acceleration
  estVelocity_1 = estVelocity_1 + (desPos.x-estPos_1)*desAcc1*dt; // 1B velocity
  estVelocity_2 = estVelocity_2 + (desPos.y-estPos_2)*desAcc2*dt; // 2B velocity

  //Correction Step
  float const mixHorizVel = 0.1f; //mixing constant
  if (in.opticalFlowSensor.updated) {
    float sigma_1 = in.opticalFlowSensor.value_x;
    float sigma_2 = in.opticalFlowSensor.value_y;
    float div = (cosf(estRoll)*cosf(estPitch));

    if (div > 0.5f) {
      float deltaPredict = estHeight / div; //delta for velocity measurements

      float v1Meas = (-sigma_1 + in.imuMeasurement.rateGyro.y)*deltaPredict;
      float v2Meas = (-sigma_2 - in.imuMeasurement.rateGyro.x)*deltaPredict;
```

```

      estVelocity_1 = (1-mixHorizVel) * estVelocity_1 + mixHorizVel * v1Meas;
      estVelocity_2 = (1-mixHorizVel) * estVelocity_2 + mixHorizVel * v2Meas;
    }
  }
//---------------------------------------------------------------------------

// Horizontal Control -------------------------------------------------------

  desAcc1 = -(1/timeConst_horizVel) * (estVelocity_1-100*(desPos.x-estPos_1)*dt)/dt;
  desAcc2 = -(1/timeConst_horizVel) * (estVelocity_2-100*(desPos.y-estPos_2)*dt)/dt;

  float desRoll = -desAcc2/gravity;   // Desired Roll Angle
  float desPitch = desAcc1/gravity;   // Desired Pitch Angle
  float desYaw = 0;                   // Desired Yaw Angle

  des_attitude.x = desRoll; //set des pitch and roll
  des_attitude.y = desPitch;
//  cmdAngAcclRoll = -0.1;
//  cmdAngAcclPitch = 0.1;
//---------------------------------------------------------------------------


// Vertical Control ---------------------------------------------------------

  if (in.userInput.buttonBlue == true) //flight trajectory
    desHeight = 1.5f;
    desPos.x = 1; //desired x coordinate
    desPos.y = 1; //desired y coordinate
  if (t > 9000)
    desHeight = 0.04f; //land before dropping out of the sky (competition)
  if (in.userInput.buttonRed == false)
    t = 0; //reset flight timer
//  printf("%6.3d",t);
  const float desAcc3 = -2 * dampingRatio_height * natFreq_height * estVelocity_3 \
                        - natFreq_height * natFreq_height * (estHeight - desHeight);

  float desNormalizedAcceleration = (gravity + desAcc3) / (cosf(estRoll) * cosf(estPitch));

//---------------------------------------------------------------------------


//Start of our mixer calculation------------------------------------------
```

```cpp

float length = 33e-3f; //Size parameter constant
float kappa = 0.01f; //Propeller thrust to torque coefficient

//Added input for desired angular acceleration
Vector3f des_ang_accel(cmdAngAcclRoll,cmdAngAcclPitch,cmdAngAcclYaw);

//Create Inertia Tensor
Matrix3f inertia_tensor;
inertia_tensor << inertia_xx, 0, 0,
                  0, inertia_yy, 0,
                  0, 0, inertia_zz;

//Calculate desired thrust from normalized thrust
float des_thrust = mass*desNormalizedAcceleration;

//Calculate desired torque from desired angular acceleration
Vector3f des_torque = inertia_tensor*des_ang_accel;

//Assigned values to desired thrust and torque matrix
Vector4f desired_output(des_thrust,des_torque(0),des_torque(1),des_torque(2));

Matrix4f mixer; //Initialize mixer matrix

//Assign values to mixer matrix
mixer(0,0) = 1.0f;
mixer(0,1) = 1.0f/length;
mixer(0,2) = -1.0f/length;
mixer(0,3) = 1.0f/kappa;

mixer(1,0) = 1.0f;
mixer(1,1) = -1.0f/length;
mixer(1,2) = -1.0f/length;
mixer(1,3) = -1.0f/kappa;

mixer(2,0) = 1.0f;
mixer(2,1) = -1.0f/length;
mixer(2,2) = 1.0f/length;
mixer(2,3) = 1.0f/kappa;

mixer(3,0) = 1.0f;
mixer(3,1) = 1.0f/length;
mixer(3,2) = 1.0f/length;
mixer(3,3) = -1.0f/kappa;

//Calculate desired propeller forces
Vector4f des_prop_force = 0.25f*mixer*desired_output;
```

```

//End of mixer calculations---------------------------------------------

//Map propeller forces to PWM signals
  //Propeller 1
  int force1 = speedFromForce(des_prop_force(0));
  int speed1 = pwmCommandFromSpeed(force1);

  //Propeller 2
  int force2 = speedFromForce(des_prop_force(1));
  int speed2 = pwmCommandFromSpeed(force2);

  //Propeller 3
  int force3 = speedFromForce(des_prop_force(2));
  int speed3 = pwmCommandFromSpeed(force3);

  //Propeller 4
  int force4 = speedFromForce(des_prop_force(3));
  int speed4 = pwmCommandFromSpeed(force4);

//Assign PWM signals to each motor when red button is pressed
  if (in.userInput.buttonRed == true) {
    outVals.motorCommand1 = speed1;
    outVals.motorCommand2 = speed2;
    outVals.motorCommand3 = speed3;
    outVals.motorCommand4 = speed4;
    if (t > 9850) {
      outVals.motorCommand1 = 0;
      outVals.motorCommand2 = 0;
      outVals.motorCommand3 = 0;
      outVals.motorCommand4 = 0;
    }
  }

  //Console Output Values for Debugging -------------------------------

  //Attitude Estimation for Telemetry
    outVals.telemetryOutputs_plusMinus100[0]=estRoll;       //Roll estimate
    outVals.telemetryOutputs_plusMinus100[1]=estPitch;      //Pitch estimate
    outVals.telemetryOutputs_plusMinus100[2]=estYaw;        //Yaw estimate

  //Velocity Estimation for Telemetry
    outVals.telemetryOutputs_plusMinus100[3]=estVelocity_1; //1B velocity estimate
    outVals.telemetryOutputs_plusMinus100[4]=estVelocity_2; //2B velocity estimate
    outVals.telemetryOutputs_plusMinus100[5]=estVelocity_3; //3B velocity estimate
```

```
    outVals.telemetryOutputs_plusMinus100[6]=estHeight;     //height estimate

  //Desired Pitch and Roll Angles for Telemetry
    outVals.telemetryOutputs_plusMinus100[7] = estPos_1;     //Desired roll angle
    outVals.telemetryOutputs_plusMinus100[8] = estPos_2;    //Desired pitch angle

  //Desired Normalized Thrust
    outVals.telemetryOutputs_plusMinus100[9] = desNormalizedAcceleration; // desired thrust

//  //Commanded Angular Acceleration for Telemetry
//    outVals.telemetryOutputs_plusMinus100[3]=cmdAngAcclRoll;
//    outVals.telemetryOutputs_plusMinus100[4]=cmdAngAcclPitch;
//    outVals.telemetryOutputs_plusMinus100[5]=cmdAngAcclYaw;
//
//  //Commanded Angular Velocity for Telemetry
//    outVals.telemetryOutputs_plusMinus100[6]=cmdAngVelRoll;
//    outVals.telemetryOutputs_plusMinus100[7]=cmdAngVelPitch;
//    outVals.telemetryOutputs_plusMinus100[8]=cmdAngVelYaw;

//-----------------------------------------------------------------------
  t++;
  //copy the inputs and outputs:
  lastMainLoopInputs = in;
  lastMainLoopOutputs = outVals;
  return outVals;
}

void PrintStatus() {

//Added printing the accelerometer measurements
  printf("Acc:");
  printf("x=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.x)); //Accel. x
  printf("y=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.y)); //Accel. y
  printf("z=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.z)); //Accel. z

//Added printing of the raw rate gyro measurements
  printf("Gyro:");
  printf("x=%6.3f, ", double(lastMainLoopInputs.imuMeasurement.rateGyro.x)); //Gyro x
```

```
  printf("y=%6.3f, ", double(lastMainLoopInputs.imuMeasurement.rateGyro.y)); //Gyro y
  printf("z=%6.3f, ", double(lastMainLoopInputs.imuMeasurement.rateGyro.z)); //Gyro z
  printf("\n\n");

//Added printing of commanded angular velocities
  printf("Commanded Angular Velocities = (%6.3f %6.3f, %6.3f)\n\n",
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[6]), // Bias for x gyro
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[7]), // Bias for y gyro
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[8])); // Bias for z gyro

//Added printing of commanded angular accelerations
  printf("Commanded Angular Accelerations = (%6.3f %6.3f, %6.3f)\n\n",
         double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[3]), // Corrected x gyro
         double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[4]), // Corrected y gyro
         double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[5])); // Corrected z gyro

//Added printing of the estimated angles
  printf("Estimated Attitude = (%6.3f %6.3f, %6.3f)\n\n",
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[0]), // Estimated roll
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[1]), // Estimated pitch
           double(lastMainLoopOutputs.telemetryOutputs_plusMinus100[2])); // Estimated yaw

//Range and Flow Sensor Printing
  printf("Last range = %6.3fm, ", \
         double(lastMainLoopInputs.heightSensor.value)); // Range sensor value
  printf("Last Flow: x = %6.3f, y = %6.3f\n", \
         double(lastMainLoopInputs.opticalFlowSensor.value_x), \
         double(lastMainLoopInputs.opticalFlowSensor.value_y)); // x and y position values



  //just an example of how we would inspect the last main loop inputs and outputs:
//  printf("Last main loop inputs:\n");
//  printf("  batt voltage = %6.3f\n",
```

```
//         double(lastMainLoopInputs.batteryVoltage.value));
//  printf("  JS buttons: ");
//  if (lastMainLoopInputs.userInput.buttonRed)
//    printf("buttonRed ");
//  if (lastMainLoopInputs.userInput.buttonGreen)
//    printf("buttonGreen ");
//  if (lastMainLoopInputs.userInput.buttonBlue)
//    printf("buttonBlue ");
//  if (lastMainLoopInputs.userInput.buttonYellow)
//    printf("buttonYellow ");
//  if (lastMainLoopInputs.userInput.buttonArm)
//    printf("buttonArm ");
  printf("\n");
  printf("Last main loop outputs:\n");
  printf("  motor commands: = %6.3f\t%6.3f\t%6.3f\t%6.3f\t\n",
         double(lastMainLoopOutputs.motorCommand1),
         double(lastMainLoopOutputs.motorCommand2),
         double(lastMainLoopOutputs.motorCommand3),
         double(lastMainLoopOutputs.motorCommand4));
}

```