```
#include "UserCode.hpp"
#include "UtilityFunctions.hpp"
#include "Vec3f.hpp"

#include <stdio.h> //for printf

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

//Some constants that we may use:
const float mass = 30e-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

Vec3f estGyroBias = Vec3f(0, 0, 0);  // declare x,y,z variable for bias
Vec3f rateGyro_corr = Vec3f(0, 0, 0);  // declare x,y,z variable for corrected Gyro measurements

float estRoll = 0.0f;  //declare variable for Roll
float estPitch = 0.0f;  //declare variable for Pitch
float estYaw = 0.0f;  //declare variable for Yaw

const float rho = 0.001f;
float phiK = 0.0f;
float thetaK = 0.0f;
```

```
float newRoll = 0.0f;
float newPitch = 0.0f;

float phiK_nan = 0;
bool rateGyro_x_nan = false;

// lab04
float l = 33e-3f;
float mB = 32e-3f;
float k = 0.01f;

float Jxx = 16.0e-6f;
float Jzz = 29.0e-6f;

float cp1 = 0.0f;
float cp2 = 0.0f;
float cp3 = 0.0f;
float cp4 = 0.0f;
float cSigma = 0.0f;

float n1 = 0.0f;
float n2 = 0.0f;
float n3 = 0.0f;

float pDes = 0.0f;
float qDes = 0.0f;
float rDes = 0.0f;

float dpCmd = 0.0f;
```

```
float dqCmd = 0.0f;
float drCmd = 0.0f;

Vec3f cmdAngAcc = Vec3f(0, 0, 0);


float phiDes = 0.0f;
float thetaDes = 0.0f;
float psiDes = 0.0f;

float pCmd = 0.0f;
float qCmd = 0.0f;
float rCmd = 0.0f;

Vec3f cmdAngVel = Vec3f (0,0,0);

// LAB05
float estHeight = 0;
float estVelocity_1 = 0;
float estVelocity_2 = 0;
float estVelocity_3 = 0;

float lastHeightMeas_meas = 0;
float lastHeightMeas_time = 0;

const float timeConstant_rollAngle = 0.12f; //[s]
const float timeConstant_pitchAngle = timeConstant_rollAngle; //[s]
const float timeConstant_yawAngle = 0.2f; //[s]
```

```
const float timeConstant_rollRate = 0.04f; //[s]
const float timeConstant_pitchRate = timeConstant_rollRate; //[s]
const float timeConstant_yawRate = 0.1f; //[s]

// Horizontal Alt Control
const float timeConst_horizVel = 0.5f;
const float timeConst_horizPos = 1.0f;

//Vertical Control
const float natFreq_height = 2.2f;
const float dampingRatio_height = 0.7f;

const float mixHeight = 0.3f;
const float mixHorizVel = 0.1f;

float desHeight = 0.0f;
float desPos1 = 0;
float desPos2 = 0;

float estPos1 = 0;
float estPos2 = 0;
bool greenButtonPressed = false;
bool redButtonPressed = false;
bool yellowButtonPressed = false;
float greenButtonPressedTime = 0;
float redButtonPressedTime = 0;
float ramp1 = 0;
float ramp3 = 0;
bool powerOff = false;
```

```
float pos1integral = 0;
float pos2integral = 0;
const float timeConst_horizPosIntegral = 500.0f;

MainLoopOutput MainLoop(MainLoopInput const &in) {
  if(in.joystickInput.buttonYellow == true) {
    if(yellowButtonPressed == false){
      estGyroBias.x = 0;
      estGyroBias.y = 0;
      estGyroBias.z = 0;
      yellowButtonPressed = true;
    }
    estYaw = 0;
    newRoll = 0;
    newPitch = 0;
    estHeight = 0;
    estPos1 = 0;
    estPos2 = 0;
    pos1integral = 0;
    pos2integral = 0;
    estVelocity_3 = 0;
    estVelocity_2 = 0;
    estVelocity_1 = 0;
    desHeight = 0;
    desPos1 = 0;
    desPos2 = 0;
    ramp1 = 0;
    ramp3 = 0;
```

```
  greenButtonPressed = false;
  redButtonPressed = false;
  redButtonPressedTime = 0;
  greenButtonPressedTime = 0;
  estGyroBias = estGyroBias + (in.imuMeasurement.rateGyro / 500.0f);  //500 measurements
 }
 else if(in.joystickInput.buttonBlue == true && in.joystickInput.buttonRed == true) {
  desHeight = 1.0;
  desPos1 = 0;
  desPos2 = 0;
 }
 else if(in.joystickInput.buttonRed == true && in.joystickInput.buttonGreen == false) {
  if(redButtonPressed == false){
    redButtonPressedTime = in.currentTime;
    redButtonPressed = true;
  }
  ramp1 = (in.currentTime - redButtonPressedTime)*0.2f;
  if(1.0f > ramp1){
    desPos1 = ramp1;
    desPos2 = 0;
  }
  else{
    desPos1 = 1.0;
    desPos2 = 0;
  }
 }
 else if(in.joystickInput.buttonGreen == true) {
  if(greenButtonPressed == false){
    greenButtonPressedTime = in.currentTime;
```

```
    greenButtonPressed = true;
  }
  ramp3 = 1.0f - (in.currentTime - greenButtonPressedTime);
  if(estHeight > 0.1f){
    desHeight = ramp3;
    desPos1 = 1.0;
    desPos2 = 0;
  }
  else{
    //desHeight = 0; turn off the motors
    powerOff = true;
  }
}
if(in.joystickInput.buttonYellow == false) {
// Rate Gyro bias
  if (in.currentTime < 1.0f) {
    estGyroBias = estGyroBias + (in.imuMeasurement.rateGyro / 500.0f);  //500 measurements
  }
  rateGyro_corr = in.imuMeasurement.rateGyro - estGyroBias;  //Correcting bias


  estYaw = estYaw + dt * rateGyro_corr.z;


  // To reduce drift
  phiK = (lastMainLoopInputs.imuMeasurement.accelerometer.y) / gravity;


  //make sure phiK is not NAN
  if (phiK == phiK) {
    newRoll = (1 - rho) * (newRoll + (dt * rateGyro_corr.x)) + (rho * phiK);
  } else {
```

```
  //    newRoll = newRoll;  //do nothing
}
thetaK = -(lastMainLoopInputs.imuMeasurement.accelerometer.x) / gravity;

if (thetaK == thetaK) {
  newPitch = (1 - rho) * (newPitch + (dt * rateGyro_corr.y)) + (rho * thetaK);
} else {
  //    newPitch = newPitch;  //do nothing
}



//LAB05 height estimator:
estHeight = estHeight + estVelocity_3 * dt;
estVelocity_3 = estVelocity_3 + 0 * dt; //assume constant (!)

//correction step, directly after the prediction step:
if (in.heightSensor.updated){
  //check that the measurement is reasonable
  if (in.heightSensor.value < 5.0f){
    float hMeas = in.heightSensor.value * cosf(newRoll) * cosf(newPitch);
    estHeight = (1-mixHeight) * estHeight + mixHeight *hMeas;

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

    estVelocity_3 = (1-mixHeight) * estVelocity_3 + mixHeight * v3Meas;
    //store this measurement for the next velocity update
    lastHeightMeas_meas = hMeas;
    lastHeightMeas_time = in.currentTime;
  }
```

```
}

//prediction
//(just assume velocity is constant):
estVelocity_1 = estVelocity_1 + 0 * dt;
estVelocity_2 = estVelocity_2 + 0 * dt;



//correction step:
if (in.opticalFlowSensor.updated){
  float sigma_1 = in.opticalFlowSensor.value_x;
  float sigma_2 = in.opticalFlowSensor.value_y;

  float div = (cosf(newRoll) * cosf(newPitch));
  if(div > 0.5f){
    float deltaPredict = estHeight / div;//this is delta in the equation

    float v1Meas = (-sigma_1 + rateGyro_corr.y) * deltaPredict;
    float v2Meas = (-sigma_2 - rateGyro_corr.x) * deltaPredict;

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

//position integrator
estPos1 = estPos1 + estVelocity_1*dt;
estPos2 = estPos2 + estVelocity_2*dt;
```

```
    pos1integral = pos1integral + (estPos1 - desPos1);
    pos2integral = pos2integral + (estPos2 - desPos2);

    float desAcc1 = -(1 / timeConst_horizVel) * (estVelocity_1 - ((-1/timeConst_horizPos)*(estPos1 - desPos1) + (-1/timeConst_horizPosIntegral)*pos1integral));
    float desAcc2 = -(1 / timeConst_horizVel) * (estVelocity_2 - ((-1/timeConst_horizPos)*(estPos2 - desPos2) + (-1/timeConst_horizPosIntegral)*pos2integral));

    float desRoll = -1 / gravity * desAcc2;
    float desPitch = 1 / gravity * desAcc1;
    float desYaw = 0;

    // Angle Control

    pCmd = (-1/timeConstant_rollAngle) * (newRoll - desRoll);
    qCmd = (-1/timeConstant_pitchAngle) * (newPitch - desPitch);
    rCmd = (-1/timeConstant_yawAngle) * (estYaw - desYaw);

    cmdAngVel = Vec3f (pCmd,qCmd,rCmd);

    dpCmd = (-1/timeConstant_rollRate) * (rateGyro_corr.x - pCmd);
    dqCmd = (-1/timeConstant_pitchRate) * (rateGyro_corr.y - qCmd);
    drCmd = (-1/timeConstant_yawRate) * (rateGyro_corr.z - rCmd);

    cmdAngAcc = Vec3f (dpCmd,dqCmd,drCmd);

    n1 = dpCmd*Jxx + (Jxx - Jzz)*(-rateGyro_corr.y*rateGyro_corr.z);
    n2 = dqCmd*Jxx + (Jxx - Jzz)*(rateGyro_corr.x*rateGyro_corr.z);
    n3 = drCmd*Jzz;
```

```
  //Vertical
  float desAcc3 = -2 * dampingRatio_height * natFreq_height * estVelocity_3 - natFreq_height *
natFreq_height * (estHeight - desHeight);
  float desNormalizedAcceleration = (gravity + desAcc3) / (cosf(newRoll) * cosf(newPitch));
  cSigma = desNormalizedAcceleration*mB;

  // Mixer Matrix
  if(powerOff == false){
   cp1 = 0.25f*(cSigma + n1/l - n2/l + n3/k);
   cp2 = 0.25f*(cSigma - n1/l - n2/l - n3/k);
   cp3 = 0.25f*(cSigma - n1/l + n2/l + n3/k);
   cp4 = 0.25f*(cSigma + n1/l + n2/l - n3/k);
  }
  else{
   cp1 = 0;
   cp2 = 0;
   cp3 = 0;
   cp4 = 0;
  }

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

  outVals.motorCommand1 = pwmCommandFromSpeed(speedFromForce(cp1));
  outVals.motorCommand2 = pwmCommandFromSpeed(speedFromForce(cp2));
  outVals.motorCommand3 = pwmCommandFromSpeed(speedFromForce(cp3));
  outVals.motorCommand4 = pwmCommandFromSpeed(speedFromForce(cp4));
```

```
  //copy the inputs and outputs:
  lastMainLoopInputs = in;
  lastMainLoopOutputs = outVals;

  // Getting angles from outVals
  outVals.telemetryOutputs_plusMinus100[0] = newRoll;
  outVals.telemetryOutputs_plusMinus100[1] = newPitch;
  outVals.telemetryOutputs_plusMinus100[2] = estYaw;

  //LAB05
  outVals.telemetryOutputs_plusMinus100[3] = estVelocity_1;
  outVals.telemetryOutputs_plusMinus100[4] = estVelocity_2;
  outVals.telemetryOutputs_plusMinus100[5] = estVelocity_3;
  outVals.telemetryOutputs_plusMinus100[6] = estHeight;

  outVals.telemetryOutputs_plusMinus100[7] = lastMainLoopInputs.imuMeasurement.accelerometer.y
* dt;
  outVals.telemetryOutputs_plusMinus100[8] = lastMainLoopInputs.imuMeasurement.accelerometer.x
* dt;

  outVals.telemetryOutputs_plusMinus100[9] = estPos1;
  outVals.telemetryOutputs_plusMinus100[10] = estPos2;
  return outVals;
 }
 else{

  cp1 = 0;
  cp2 = 0;
  cp3 = 0;
```

```
  cp4 = 0;
  //Define the output numbers (in the struct outVals):
  MainLoopOutput outVals;

  outVals.motorCommand1 = pwmCommandFromSpeed(speedFromForce(cp1));
  outVals.motorCommand2 = pwmCommandFromSpeed(speedFromForce(cp2));
  outVals.motorCommand3 = pwmCommandFromSpeed(speedFromForce(cp3));
  outVals.motorCommand4 = pwmCommandFromSpeed(speedFromForce(cp4));

  //copy the inputs and outputs:
  lastMainLoopInputs = in;
  lastMainLoopOutputs = outVals;

  // Getting angles from outVals
  outVals.telemetryOutputs_plusMinus100[0] = newRoll;
  outVals.telemetryOutputs_plusMinus100[1] = newPitch;
  outVals.telemetryOutputs_plusMinus100[2] = estYaw;

  //LAB05
  outVals.telemetryOutputs_plusMinus100[3] = estVelocity_1;
  outVals.telemetryOutputs_plusMinus100[4] = estVelocity_2;
  outVals.telemetryOutputs_plusMinus100[5] = estVelocity_3;
  outVals.telemetryOutputs_plusMinus100[6] = estHeight;

  outVals.telemetryOutputs_plusMinus100[7] = lastMainLoopInputs.imuMeasurement.accelerometer.y
* dt;
  outVals.telemetryOutputs_plusMinus100[8] = lastMainLoopInputs.imuMeasurement.accelerometer.x
* dt;
```

```
    outVals.telemetryOutputs_plusMinus100[9] = estPos1;
    outVals.telemetryOutputs_plusMinus100[10] = estPos2;
    return outVals;
  }
}

void PrintStatus() {

  printf("Last main loop inputs:\n");
  printf("  batt voltage = %6.3f\n",
         double(lastMainLoopInputs.batteryVoltage.value));
  printf("  JS buttons: ");
  if (lastMainLoopInputs.joystickInput.buttonRed)
    printf("buttonRed ");
  if (lastMainLoopInputs.joystickInput.buttonGreen)
    printf("buttonGreen ");
  if (lastMainLoopInputs.joystickInput.buttonBlue)
    printf("buttonBlue ");
  if (lastMainLoopInputs.joystickInput.buttonYellow)
    printf("buttonYellow ");
  if (lastMainLoopInputs.joystickInput.buttonStart)


    printf("buttonStart ");
  if (lastMainLoopInputs.joystickInput.buttonSelect)
    printf("buttonSelect ");
  printf("\n");
  printf("Last main loop outputs:\n");
  printf("  motor command 1 = %6.3f\n",
```

```
      double(lastMainLoopOutputs.motorCommand1));
printf("  motor command 2 = %6.3f\n",
      double(lastMainLoopOutputs.motorCommand2));
printf("  motor command 3 = %6.3f\n",
      double(lastMainLoopOutputs.motorCommand3));
printf("  motor command 4 = %6.3f\n",
      double(lastMainLoopOutputs.motorCommand4));

if (phiK_nan > 0)
  printf("NAN AT phiK t=%6.3f\n", double(phiK_nan));
if (rateGyro_x_nan)
  printf("NAN AT rateGyro_x\n");

printf("cSimga = %6.3f\n",
      double(cSigma));
printf("cp1 = %6.3f\n",
      double(cp1));
printf("cp2 = %6.3f\n",
      double(cp2));
printf("cp3 = %6.3f\n",
      double(cp3));
printf("cp4 = %6.3f\n",
      double(cp4));

// LAB05
printf("Last range = %6.3f\n ",
      double (lastMainLoopInputs.heightSensor.value));
printf("Last flow : x=%6.3f , y=%6.3 f\n",
      double (lastMainLoopInputs.opticalFlowSensor.value_x),
```

```
        double (lastMainLoopInputs.opticalFlowSensor.value_y));




}
```