```
// Start the robot flat on the ground
// compile and load the code
// wait for code to load (look for "done uploading" in the Arduino IDE)
// wait for red LED to flash on board
// gently lift body of rocky to upright position
// this will enable the balancing algorithm
// wait for the buzzer
// let go
//
// The balancing algorithm is implemented in BalanceRocky()
// which you should modify to get the balancing to work
//


#include <Balboa32U4.h>
#include <Wire.h>
#include <LSM6.h>
#include "Balance.h"


extern int32_t angle_accum;
extern int32_t speedLeft;
extern int32_t driveLeft;
extern int32_t distanceRight;
extern int32_t speedRight;
extern int32_t distanceLeft;
extern int32_t distanceRight;
float speedCont = 0;
float displacement_m = 0;
int16_t limitCount = 0;
uint32_t cur_time = 0;
float distLeft_m;
float distRight_m;


extern uint32_t delta_ms;
float measured_speedL = 0;
float measured_speedR = 0;
float desSpeedL=0;
float desSpeedR =0;
float dist_accumL_m = 0;
float dist_accumR_m = 0;
float dist_accum = 0;
float speed_err_left = 0;
float speed_err_right = 0;
float speed_err_left_acc = 0;
float speed_err_right_acc = 0;
float errAccumRight_m = 0;
float errAccumLeft_m = 0;
float prevDistLeft_m = 0;
float prevDistRight_m = 0;
float angle_rad_diff = 0;
float angle_rad;                // this is the angle in radians
float angle_rad_accum = 0;      // this is the accumulated angle in radians
float angle_prev_rad = 0; // previous angle measurement
extern int32_t displacement;
int32_t prev_displacement=0;
uint32_t prev_time;

#define G_RATIO (162.5)



LSM6 imu;
Balboa32U4Motors motors;
Balboa32U4Encoders encoders;
Balboa32U4Buzzer buzzer;
Balboa32U4ButtonA buttonA;


#define FIXED_ANGLE_CORRECTION (0.35)  // ***** Replace the value 0.25 with the value you obtained from the Gyro calibration procedure





//////////////////////////////////////////////////////////////////////////////////////////////
// This is the main function that performs the balancing
// It gets called approximately once every 10 ms  by the code in loop()
// You should make modifications to this function to perform your
// balancing
//////////////////////////////////////////////////////////////////////////////////////////////

void BalanceRocky()
{

    // **************Enter the control parameters here

  //CONTROL PARAMETERS FOR INITIAL SYSTEM
  float Kp = 1.2151e+06;
  float Ki = 6.3253e+06;
  float Ji = -1.2080e+06;
  float Jp = 1.6871e+05;
  float Ci = -1.3269e+06;

  //CONTROL PARAMETERS FOR STATIONARY SYSTEM
  float Kp = 1.2151e+06;
  float Ki = 6.3253e+06;
  float Ji = -1.2080e+06;
  float Jp = 1.6871e+05;
  float Ci = -1.3269e+06;


    float v_c_L, v_c_R; // these are the control velocities to be sent to the motors
    float v_d = 0; // this is the desired speed produced by the angle controller


   // Variables available to you are:
   // angle_rad  - angle in radians
   // angle_rad_accum - integral of angle
   // measured_speedR - right wheel speed (m/s)
   // measured_speedL - left wheel speed (m/s)
   // distLeft_m - distance traveled by left wheel in meters
   // distRight_m - distance traveled by right wheel in meters  (this is the integral of the velocities)
   // dist_accum - integral of the distance

   // *** enter an equation for v_d in terms of the variables available ****
    v_d =  Kp * angle_rad + Ki * angle_rad_accum; // this is the desired velocity from the angle controller


  // The next two lines implement the feedback controller for the motor. Two separate velocities are calculated.
  //
  //
  // We use a trick here by criss-crossing the distance from left to right and
  // right to left. This helps ensure that the Left and Right motors are balanced

  // *** enter equations for input signals for v_c (left and right) in terms of the variables available ****
    v_c_R = v_d - (Jp * measured_speedR + Ji * distLeft_m) - (Ci * dist_accum);
    v_c_L = v_d - (Jp * measured_speedL + Ji * distRight_m) - (Ci * dist_accum);



    // save desired speed for debugging
    desSpeedL = v_c_L;
    desSpeedR = v_c_R;

    // the motor control signal has to be between +- 300. So clip the values to be within that range
    // here
    if(v_c_L > 300) v_c_L = 300;
    if(v_c_R > 300) v_c_R = 300;
    if(v_c_L < -300) v_c_L = -300;
    if(v_c_R < -300) v_c_R = -300;

    // Set the motor speeds
    motors.setSpeeds((int16_t) (v_c_L), (int16_t)(v_c_R));
    // Serial.println("set!");

}



void setup()
{
  // Uncomment these lines if your motors are reversed.
  // motors.flipLeftMotor(true);
  // motors.flipRightMotor(true);

  Serial.begin(9600);
  prev_time = 0;
  displacement = 0;
  ledYellow(0);
  ledRed(1);
  balanceSetup();
  ledRed(0);
  angle_accum = 0;

  ledGreen(0);
  ledYellow(0);
}



int16_t time_count = 0;
extern int16_t angle_prev;
int16_t start_flag = 0;
int16_t start_counter = 0;
void lyingDown();
extern bool isBalancingStatus;
extern bool balanceUpdateDelayedStatus;

void UpdateSensors()
{
  static uint16_t lastMillis;
  uint16_t ms = millis();

  // Perform the balance updates at 100 Hz.
  balanceUpdateDelayedStatus = ms - lastMillis > UPDATE_TIME_MS + 1;
  lastMillis = ms;

  // call functions to integrate encoders and gyros
  balanceUpdateSensors();

  if (imu.a.x < 0)
  {
    lyingDown();
    isBalancingStatus = false;
  }
  else
  {
    isBalancingStatus = true;
  }
}



void GetMotorAndAngleMeasurements()
{
    // convert distance calculation into meters
    // and integrate distance
    distLeft_m = ((float)distanceLeft)/((float)G_RATIO)/12.0*80.0/1000.0*3.14159;
    distRight_m = ((float)distanceRight)/((float)G_RATIO)/12.0*80.0/1000.0*3.14159;
    dist_accum += (distLeft_m+distRight_m)*0.01/2.0;

    // compute left and right wheel speed in meters/s
    measured_speedL = speedLeft/((float)G_RATIO)/12.0*80.0/1000.0*3.14159*100.0;
    measured_speedR = speedRight/((float)G_RATIO)/12.0*80.0/1000.0*3.14159*100.0;

    prevDistLeft_m = distLeft_m;
    prevDistRight_m = distRight_m;


    // this integrates the angle
    angle_rad_accum += angle_rad*0.01;
    // this is the derivative of the angle
    angle_rad_diff = (angle_rad-angle_prev_rad)/0.01;
    angle_prev_rad  = angle_rad;

}

void balanceResetAccumulators()
{
    errAccumLeft_m = 0.0;
    errAccumRight_m = 0.0;
    speed_err_left_acc = 0.0;
    speed_err_right_acc = 0.0;
}



void loop()
{
  static uint32_t prev_print_time = 0;   // this variable is to control how often we print on the serial monitor
  int16_t distanceDiff;    // this stores the difference in distance in encoder clicks that was traversed by the right vs the left wheel
  static float del_theta = 0;
  char enableLongTermGyroCorrection = 1;

  cur_time = millis();                   // get the current time in miliseconds


  if((cur_time - prev_time) > UPDATE_TIME_MS){
    UpdateSensors();                    // run the sensor updates.

    // calculate the angle in radians. The FIXED_ANGLE_CORRECTION term comes from the angle calibration procedure (separate sketch available for this)
    // del_theta corrects for long-term drift
    angle_rad = ((float)angle)/1000/180*3.14159 - FIXED_ANGLE_CORRECTION - del_theta;

    if(angle_rad > 0.1 || angle_rad < -0.1)      // If angle is not within +- 6 degrees, reset counter that waits for start
    {
      start_counter = 0;
   }


  if(angle_rad > -0.1 && angle_rad < 0.1 && ! start_flag)
  {
    // increment the start counter
    start_counter++;
    // If the start counter is greater than 30, this means that the angle has been within +- 6 degrees for 0.3 seconds, then set the start_flag
    if(start_counter > 30)
    {
      balanceResetEncoders();
      start_flag = 1;
      buzzer.playFrequency(DIV_BY_10 | 445, 1000, 15);
      // Serial.println("Starting");
      ledYellow(1);
    }
  }


  // every UPDATE_TIME_MS, if the start_flag has been set, do the balancing
  if(start_flag)
  {
    // Serial.println("check1");
    GetMotorAndAngleMeasurements();
    if(enableLongTermGyroCorrection)
      del_theta = 0.999*del_theta + 0.001*angle_rad;  // assume that the robot is standing. Smooth out the angle to correct for long-term gyro drift

    // Control the robot
    BalanceRocky();
    // Serial.println("balancing");
  }
  prev_time = cur_time;
  }
// if the robot is more than 45 degrees, shut down the motor
  if(start_flag && angle_rad > .78)
  {
    motors.setSpeeds(0,0);
    start_flag = 0;
    // Serial.println("shut1");
  }
  else if(start_flag && angle < -0.78)
  {
    motors.setSpeeds(0,0);
    start_flag = 0;
    // Serial.println("shut2");
  }

// kill switch
  if(buttonA.getSingleDebouncedPress())
  {
      motors.setSpeeds(0,0);
      while(!buttonA.getSingleDebouncedPress());
  }

if(cur_time - prev_print_time > 103)   // do the printing every 105 ms. Don't want to do it for an integer multiple of 10ms to not hog the processor
  {
        Serial.print(angle_rad);
        Serial.print("\t");
        Serial.print(distLeft_m);
        Serial.print("\t");
        Serial.print(measured_speedL);
        Serial.print("\t");
        Serial.print(measured_speedR);
        Serial.print("\t");
       Serial.println(speedCont);
       prev_print_time = cur_time;
  }

}
```