California Polytechnic University of San Luis Obispo Department of Electrical Engineering

Spring EE 329 Section: 7 – Microcontroller-Based Systems Design

Flight Stabilizer - SteadiFly

Professor:

John Penvenne

Authors:

Dante Benedetti

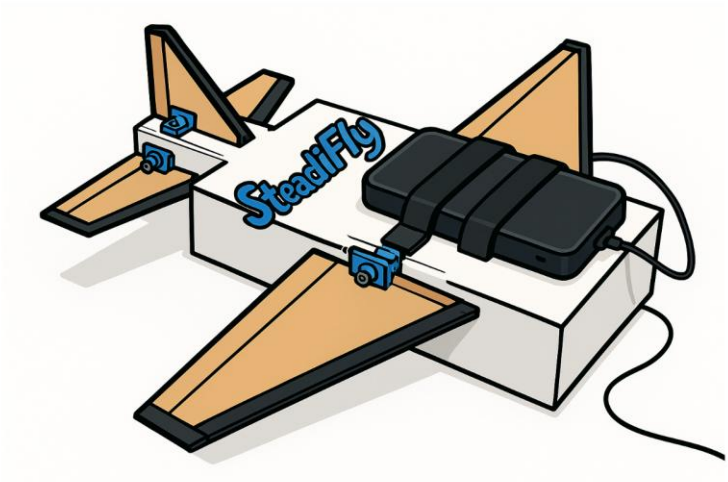
Nathan Heil

Benjamin Tavares

Gulianna Corteguera

Date:

June 6, 2025



Behavior Description

Project description: SteadiFly is an advanced flight stabilizer system powered by a Nucleo development board. Designed for aircraft stability, SteadiFly integrates key components including servos, a gyroscope/accelerometer, and a dedicated power supply. The system continuously monitors the aircraft's orientation and motion through real-time sensor data. Based on this input, the Nucleo board intelligently adjusts the wing flaps via servos to counteract disturbances and maintain level flight. Along with that the data collected is also displayed on a serial monitor via PuTTY and MatLab.

System Specification

Table 1. System Specifications

Size	324mm x 96mm x 50.3mm
Weight	1360.7g
Battery Life	>18 Hours
SG90 Servo	5 V
MPU6050	3.3V, Sample Rate of 1k Hz, 16 bit resolution
LUPART	Asynchronous serial interface, 115200 baud, 8N1
I2C	1 k Hz
TIM5	1M Hz
System Clock	4 M Hz

Code Planning and Software Architecture

This project implements a modular real-time flight control system on a bare-metal STM32 microcontroller. The software is organized around a continuous cyclic loop in main.c, which handles system initialization, sensor data acquisition, filtering, control computation, and actuator output. The MPU6050 IMU provides 3-axis gyroscope and 3-axis accelerometer data over I2C, which is processed through a Kalman filter to estimate pitch, roll, and yaw. These estimates are smoothed for display and logged via UART. Independent PID controllers are used for each axis to compute control signals that are converted to PWM outputs driving servo motors. A UART-based interface allows real-time tuning of PID gains through interrupt-driven digit parsing. The system is structured across well-defined modules: I2C.c for sensor reads, PID.c for control logic, PWM.c for actuation, uart.c for user input and logging, and delay.c for precise timing.

Several technical challenges shaped the project’s development. Configuring I2C without the STM32 HAL required manual timing calculations to achieve a stable 100 kHz clock from a 4 MHz core, balancing accuracy and simplicity. Kalman filter tuning and gyroscope bias correction demanded iterative testing to produce stable orientation estimates. Parsing UART input into usable PID gains without dynamic memory or standard string functions led to a fixed-digit input approach. Mapping PID output to servo-friendly PWM ranges involved careful

scaling and constraints to maintain control stability. These design choices prioritized reliability, modularity, and low overhead—traits essential for embedded flight systems—and were guided by a consistent, hands-on testing process throughout development.

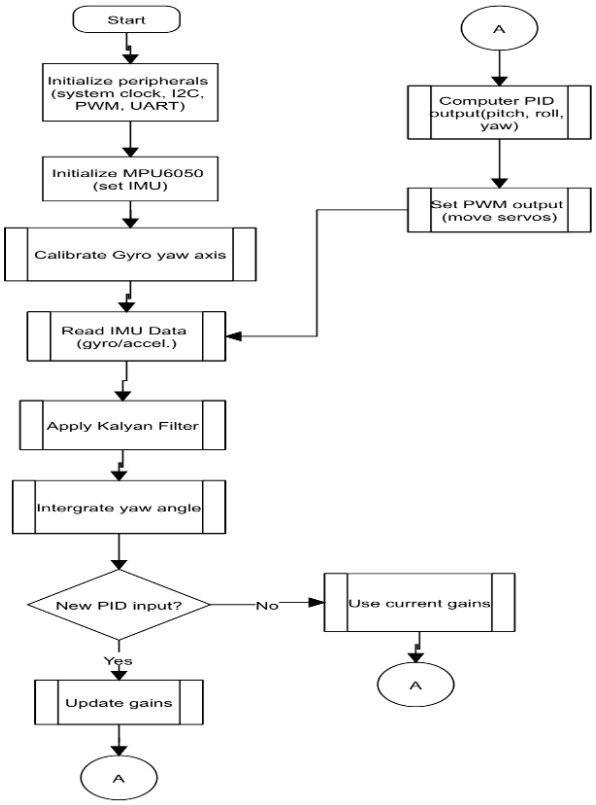


Figure 1 Flowchart of SteadiFly Code

System Schematic

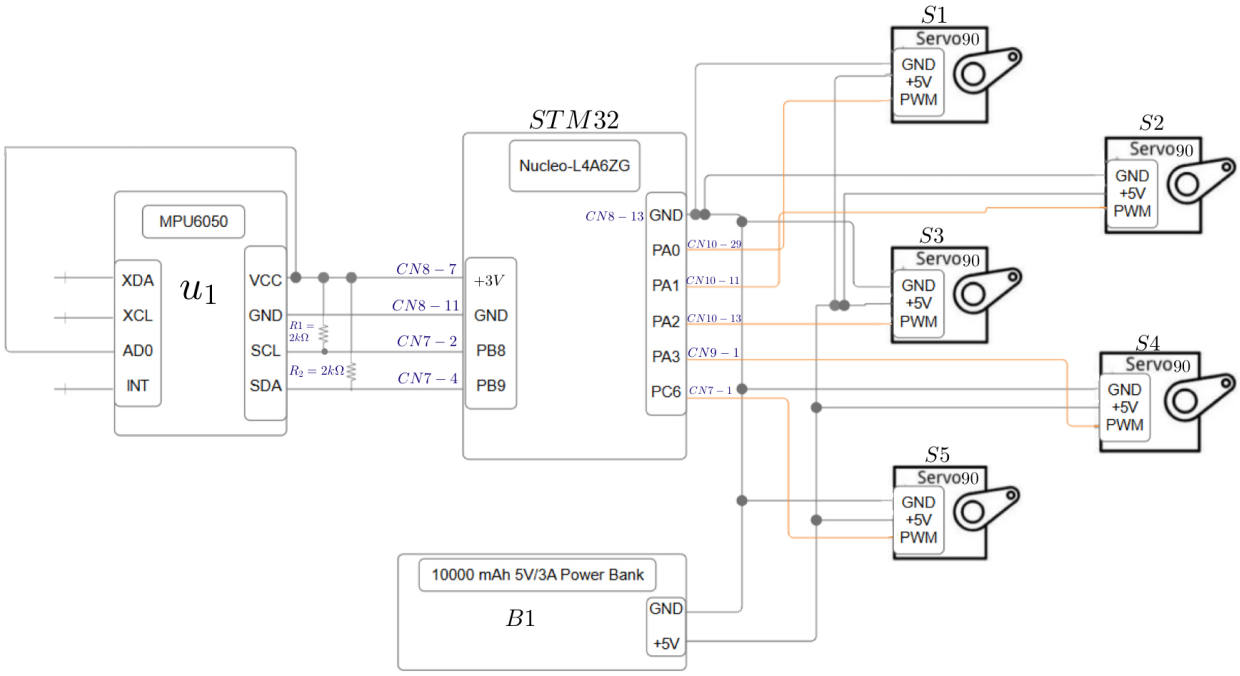


Figure 2 SteadiFly Wiring Diagram

Calculation & Characterization

Project Assignment Questions:

Question (Professor Penvenne):
When you wrote "6-axis gyroscope," did you mean a 3-axis gyroscope and a 3-axis accelerometer?

Response:
Yes, that is correct. The term "6-axis gyroscope" was used informally in our original description. More precisely, the MPU6050 is a 6-degree-of-freedom (6-DOF) inertial measurement unit (IMU) that contains a 3-axis gyroscope and a 3-axis accelerometer in a single package. Thank you for pointing out the ambiguity—we have revised the terminology in the final documentation to reflect the correct interpretation.

Calculations Used In Project [3]:
To find Roll (Accelerometer, radian)

$$\varphi = \tan^{-1}\left(\frac{A_y}{A_x}\right) \quad A_i = linear\ acceleration$$

To find Pitch (Accelerometer, radian)

$$\theta = \tan^{-1}\left(-\frac{A_x}{\sqrt{(A_y^2+A_z^2)}}\right)$$

To find Yaw (Gyroscope, degree)

$$\sigma = \sigma_o + \Sigma_{i=1}^{k}(\omega_i \cdot \Delta t)$$

- $\sigma_o = initial\ position$
- $\omega_i = angular\ velocity$
- $\Delta t = sample\ period$

Attribution

Dante Benedetti: Contributed to system initialization, UART communication, and assisted in integrating the live PID tuning interface. Provided support in testing the UART interrupt logic and verifying real-time communication robustness.

Nathan Heil: Led overall software architecture and implementation. Developed the complete main control loop, including Kalman filtering, sensor fusion, PID logic integration, and real-time logging. Wrote the core logic in main.c and designed the modular structure across all source files.

Benjamin Tavares: Implemented and tested the low-level I2C communication routines for MPU6050 data acquisition. Wrote the I2C.c module and assisted in verifying sensor readings and data reliability under varying conditions. Designed and modeled enclosure/body in SolidWorks.

Gulianna Corteguera: Configured and validated the PWM system for actuator control. Authored the PWM.c module, mapped pulse widths to servo positions, and supported field testing for output stability and servo calibration.

Appendices

Bill of Materials

LAST UPDATE: 2025-JUN-5

FINAL PROJECT: STEADIFLY ASSEMBLY

qty	part type	designator/ id	package/ footprint	description	manufacturer	vendor: p/n	unit cost	total cost
1	NUCLEO BOARD		3.94 X 2.36 IN.	STM32 NUCLEO-L4A6ZG	STMICROELECTRONICS	DK:497-NUCLEO-L4A6ZG-ND	$ 19.99	$ 19.99
1	BATTERY	B1	5.83 X 4.18 0.7 IN.	10000mAh 5V/3A POWER BANK	VANYUST	AZ:B0CQM5Q6HQ	$ 9.99	$ 9.99
2	RESISTOR	R1, R2	THROUGH-HOLE	RES 2K OHM 5% 1/8W AXIAL	STACKPOLE ELECTRONICS INC	DK:CF18JT2K00	$ 0.10	$ 0.20
1	GYROSCOPE/ACCELEROMETER	U1	QFN	MPU-6050 6-AXIS ACCELEROMETER GYROSCOPE SENSOR	HILETGO	AZ:B01DK83ZYQ	$ 6.99	$ 6.99
5	SERVO	S1-S5	MIRCO SERVO	SERVOMOTOR RC 4.8V	DFROBOT	DK:SER0006	$ 1.66	$ 9.99
26	WIRES		PRE-CUT	JUMPER WIRE M/M 6" 20COND	SPARKFUN ELECTRONICS	DK:12795	$ -	$ 2.95
1	OUTSIDE BOX		3D PRINT	MAIN HOUSING FOR ELECTRONICS	CP DIGITAL FABRICATION LAB		$ 20.00	$ 20.00
1	TAIL BOX		3D PRINT	HOUSING FOR TAIL END	CP DIGITAL FABRICATION LAB		$ 10.00	$ 10.00
1	LID		3D PRINT	DETACHABLE TOP COVER	CP DIGITAL FABRICATION LAB		$ 10.00	$ 10.00
5	CARDBOARD WING		CARDBOARD CUTOUT	WINGS & WING FLAPS			$ -	$ -
1	BLACK ELECTRICAL TAPE		0.75 X 2.94 IN.	3/4" BLACK ELECTRICAL INSULATION	3M	AZ:B001AXD0EY	$ 4.67	$ 4.67
1	GLUE		1.16 X 3.63 X 7.13 IN.	E6000 ADHESIVE	E6000	AZ:B00JP3X9WE	$ 3.99	$ 3.99
2	WOODEN DOWEL		1/4" DIAMETER	SUPPORT ROD FOR WINGS			$ 3.99	$ 8.00
4	FLAT HEAD PHILLIPS SCREWS		M2 x 12mm	LID ASSEMBLY SCREWS	HILLMAN		$ 0.45	$ 1.80
4	FAT HEAD PHILLIPS SCREWS		M2 x 6mm	NUCLEO BOARD FASTENERS	HILLMAN		$ 0.43	$ 1.72
							TOTAL	$110.29

VENDOR KEY:
DK = Digi-Key
AZ = Amazon

References

[1]*DC power supplies datasheets*. [Online]. Available: https://www.mouser.com/c/ds/?product=DC+Power+Supplies. [Accessed: 05-Jun-2025]

[2]Admin, “Complete St Datasheet Guide: Specs, features, and applications,” *Datasheet Info*, 18-May-2025. [Online]. Available: https://infodatasheet.com/st-datasheet.html. [Accessed: 04-Jun-2025]

[3]*MPU-6000 and MPU-6050 product specification revision ...*, 19-Aug-2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. [Accessed: 05-Jun-2025]

[4]*SG90 9 G Micro Servo*. [Online]. Available: https://www.friendlywire.com/projects/ne555-servo-safe/SG90-datasheet.pdf. [Accessed: 05-Jun-2025]

[5]*Chatgpt*. [Online]. Available: https://chatgpt.com/. [Accessed: 05-Jun-2025]

C source Code

```
-------------------------------------------------------------------------------
main.h
-------------------------------------------------------------------------------
/* ---------------------------------------------------------------------------
 * EE 329 - HEADER FILE FOR MAIN CONTROL MODULE
 *
 * @file     : main.h
 * @brief    : Header for main.c file. Contains defines and function prototypes.
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * @toolchain: STM32CubeIDE v1.12.0
 * ------------------------------------------------------------------------- */

#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32l4xx_hal.h"

/* ---------------------------------------------------------------------------
 * Function Prototypes
 * ------------------------------------------------------------------------- */
void Error_Handler( void );           // Trap function for unrecoverable faults
void SystemClock_Config( void );      // Configure system clock to use MSI

/* ---------------------------------------------------------------------------
 * GPIO Pin Definitions for Peripherals
```

```
 * ------------------------------------------------------------------------- */
#define B1_Pin                  GPIO_PIN_13
#define B1_GPIO_Port            GPIOC
#define LD3_Pin                 GPIO_PIN_14
#define LD3_GPIO_Port           GPIOB
#define USB_OverCurrent_Pin     GPIO_PIN_5
#define USB_OverCurrent_GPIO_Port GPIOG
#define USB_PowerSwitchOn_Pin   GPIO_PIN_6
#define USB_PowerSwitchOn_GPIO_Port GPIOG
#define STLK_RX_Pin             GPIO_PIN_7
#define STLK_RX_GPIO_Port       GPIOG
#define STLK_TX_Pin             GPIO_PIN_8
#define STLK_TX_GPIO_Port       GPIOG
#define USB_SOF_Pin             GPIO_PIN_8
#define USB_SOF_GPIO_Port       GPIOA
#define USB_VBUS_Pin            GPIO_PIN_9
#define USB_VBUS_GPIO_Port      GPIOA
#define USB_ID_Pin              GPIO_PIN_10
#define USB_ID_GPIO_Port        GPIOA
#define USB_DM_Pin              GPIO_PIN_11
#define USB_DM_GPIO_Port        GPIOA
#define USB_DP_Pin              GPIO_PIN_12
#define USB_DP_GPIO_Port        GPIOA
#define TMS_Pin                 GPIO_PIN_13
#define TMS_GPIO_Port           GPIOA
#define TCK_Pin                 GPIO_PIN_14
#define TCK_GPIO_Port           GPIOA
#define SWO_Pin                 GPIO_PIN_3
#define SWO_GPIO_Port           GPIOB
#define LD2_Pin                 GPIO_PIN_7
#define LD2_GPIO_Port           GPIOB

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

---

main.c

```
/* ------------------------------------------------------------------------
 * EE 329 - FULL PID CONTROL SYSTEM WITH UART TUNING
 *
 * @file     : main.c
 * @brief    : Implements a Kalman-filtered PID control loop with live UART tuning
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG @ 80 MHz (TIM5 @ 1 MHz)
 * @toolchain: STM32CubeIDE v1.12.0
 * @note     : Ensure MPU6050 is connected via I2C (ADDR = 0x69)
 * ------------------------------------------------------------------------- */

#include "main.h"
#include "I2C.h"
```

```c
#include "UART.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "PWM.h"
#include "PID.h"

#define MPU_ADDR ( 0x69 )

extern int digit_buf[9];       // UART live tuning input buffer
extern int digit_ready;        // Flag indicating when to update PID gains

/* ---------------------------------------------------------------------------
 * Timer5_Init()
 * Initializes TIM5 for microsecond-precision timekeeping (1 MHz frequency).
 * Used to timestamp data for logging or precise control timing.
 * ------------------------------------------------------------------------- */
void Timer5_Init( void ) {
   RCC->APB1ENR1 |= RCC_APB1ENR1_TIM5EN;
   TIM5->PSC      = 79;                  // 80 MHz / 80 = 1 MHz -> 1 us/tick
   TIM5->ARR      = 0xFFFFFFFF;          // Max 32-bit count
   TIM5->CR1     |= TIM_CR1_CEN;        // Enable timer
}

/* ---------------------------------------------------------------------------
 * Kalman filter structure and helper functions
 * Used to fuse and smooth noisy accelerometer and gyroscope data.
 * ------------------------------------------------------------------------- */
typedef struct {
   float angle;     // Estimated angle
   float bias;      // Gyro bias
   float rate;      // Unbiased angular rate
   float P[2][2];   // Error covariance matrix
} Kalman_t;

void Kalman_Init( Kalman_t* k ) {
   k->angle   = 0.0f;
   k->bias    = 0.0f;
   k->P[0][0] = 1.0f;
   k->P[0][1] = 0.0f;
   k->P[1][0] = 0.0f;
   k->P[1][1] = 1.0f;
}

/*
 * Kalman_Update()
 * Applies a Kalman filter step to estimate the new angle.
 * newAngle - from accelerometer, newRate - from gyro, dt - time delta
 */
float Kalman_Update( Kalman_t* k, float newAngle, float newRate, float dt ) {
   const float Q_angle   = 0.005f;
   const float Q_bias    = 0.01f;
   const float R_measure = 0.03f;

   k->rate  = newRate - k->bias;
```

```
    k->angle += dt * k->rate;

    k->P[0][0] += dt * (dt * k->P[1][1] - k->P[0][1] - k->P[1][0] + Q_angle);
    k->P[0][1] -= dt * k->P[1][1];
    k->P[1][0] -= dt * k->P[1][1];
    k->P[1][1] += Q_bias * dt;

    float S   = k->P[0][0] + R_measure;
    float K0  = k->P[0][0] / S;
    float K1  = k->P[1][0] / S;
    float y   = newAngle - k->angle;

    k->angle += K0 * y;
    k->bias  += K1 * y;

    float P00_temp = k->P[0][0];
    float P01_temp = k->P[0][1];

    k->P[0][0] -= K0 * P00_temp;
    k->P[0][1] -= K0 * P01_temp;
    k->P[1][0] -= K1 * P00_temp;
    k->P[1][1] -= K1 * P01_temp;

    return k->angle;
}

/* ------------------------------------------------------------------------
 * getPitch() and getRoll()
 * Converts raw accelerometer readings to angles in degrees.
 * Uses arctangent relationships between axis readings.
 * ------------------------------------------------------------------------ */
float getPitch( int16_t ax, int16_t ay, int16_t az ) {
    return atan2f(ax, sqrtf(ay * ay + az * az)) * (180.0f / 3.14159f);
}

float getRoll( int16_t ax, int16_t ay, int16_t az ) {
    return atan2f(ay, sqrtf(ax * ax + az * az)) * (180.0f / 3.14159f);
}

/* ------------------------------------------------------------------------
 * CalibrateGyro()
 * Calculates offset for the gyroscope's Z-axis by averaging 1000 samples.
 * This removes constant drift from Z-axis yaw rate.
 * ------------------------------------------------------------------------ */
void CalibrateGyro( int16_t* offset_z ) {
    int32_t sum_z = 0;
    int16_t gz;
    for (int i = 0; i < 1000; i++) {
        MPU6050_ReadGyro(NULL, NULL, &gz); // Read only gz
        sum_z += gz;
        delay_us(1000);
    }
    *offset_z = sum_z / 1000;
}
```

```c
/* ---------------------------------------------------------------------------
 * main()
 * Main control loop: initializes hardware, filters IMU data, updates PID control,
 * prints diagnostics, and drives servo motors based on angle corrections.
 * ------------------------------------------------------------------------- */
int main( void ) {
   Timer5_Init();

   // Sensor variables and system state
   int16_t gx, gy, gz, ax, ay, az;
   float yaw = 0.0f;
   int16_t offset_z = 0;
   const float dt = 0.009f;            // Sampling interval
   float sensitivity = 16.40f;         // Gyro sensitivity scale factor

   // Kalman filter state for pitch and roll
   Kalman_t kal_pitch, kal_roll;
   Kalman_Init(&kal_pitch);
   Kalman_Init(&kal_roll);

   // Filtered output for display
   static float smooth_pitch = 0.0f;
   static float smooth_roll  = 0.0f;
   static float smooth_yaw   = 0.0f;

   // Peripheral initialization
   HAL_Init();
   SystemClock_Config();
   SysTick_Init();
   I2C_Init();
   LPUART_init();
   NVIC->ISER[2] |= (1 << (LPUART1_IRQn & 0x1F));
   __enable_irq();
   PWM_Init();

   LPUART_Print("\x1B[2J\x1B[H"); // Clear terminal
   MPU6050_Init();
   I2C_WriteByte(MPU_ADDR, 0x1B, 0x18); // Configure gyro full scale
   CalibrateGyro(&offset_z);

   // Initial PID gains and scratch buffers for UART tuning
   float kp = 1.5f, ki = 0.006f, kd = 0.005f;
   float kps = 1.2f, kis = 0.01f, kds = 0.1f;

   while (1) {
      // ----------------------- Sensor Input -----------------------
      MPU6050_ReadGyro(&gx, &gy, &gz);
      MPU6050_ReadAccel(&ax, &ay, &az);

      float raw_pitch = getPitch(ax, ay, az);
      float raw_roll  = getRoll(ax, ay, az);
      float gx_dps    = gx / sensitivity;
      float gy_dps    = gy / sensitivity;
      float gz_dps    = (gz - offset_z) / sensitivity;
```

```
      // ----------------------- Sensor Fusion -----------------------
      float pitch = Kalman_Update(&kal_pitch, raw_pitch, gx_dps, dt);
      float roll  = Kalman_Update(&kal_roll,  raw_roll,  gy_dps, dt);
      yaw += gz_dps * dt;

      // ------------------------- Smoothing -------------------------
      smooth_pitch = 0.9f * smooth_pitch + 0.1f * pitch;
      smooth_roll  = 0.9f * smooth_roll  + 0.1f * roll;
      smooth_yaw   = 0.9f * smooth_yaw   + 0.1f * yaw;

      // -------------------- Live UART PID Tuning --------------------
      kps = digit_buf[0] + (digit_buf[1] / 10.0f) + (digit_buf[2] / 100.0f);
      kis = (digit_buf[3] / 10.0f) + (digit_buf[4] / 100.0f) + (digit_buf[5] /
1000.0f);
      kds = (digit_buf[6] / 10.0f) + (digit_buf[7] / 100.0f) + (digit_buf[8] /
1000.0f);

      if (digit_ready == 1) {
         kp = kps;
         ki = kis;
         kd = kds;
         digit_ready = 0;
      }

      // -------------------- PID Computation --------------------
      PID_t pid_pitch, pid_roll, pid_yaw;
      PID_Init(&pid_pitch, kp, ki, kd, -90.0f, 90.0f);
      PID_Init(&pid_roll,  kp, ki, kd, -90.0f, 90.0f);
      PID_Init(&pid_yaw,   kp, ki, kd, -90.0f, 90.0f);

      float pitch_out = PID_Compute(&pid_pitch, 0.0f, pitch, dt);
      float roll_out  = PID_Compute(&pid_roll,  0.0f, roll,  dt);
      float yaw_out   = PID_Compute(&pid_yaw,   0.0f, yaw,   dt);

      // -------------------- Data Logging --------------------
      uint32_t time_us = TIM5->CNT;
      float t_ms = time_us / 1000.0f;
      int t_ms100 = (int)(t_ms * 100);

      char log_buf[128];
      sprintf(log_buf, "%d,%d,%d,%d,%d,%d,%d\r\n",
              t_ms100, (int)(pitch * 100), (int)(pitch_out * 100),
              (int)(roll * 100), (int)(roll_out * 100),
              (int)(yaw * 100), (int)(yaw_out * 100));
      LPUART_Print(log_buf);

      // -------------------- Output to Servos --------------------
      PWM_SetPulse(1, angle_to_pulse(+pitch_out));
      PWM_SetPulse(2, angle_to_pulse(-pitch_out));
      PWM_SetPulse(3, angle_to_pulse(+roll_out));
      PWM_SetPulse(4, angle_to_pulse(+roll_out));
      PWM_SetPulse(5, angle_to_pulse(-yaw_out));

      // -------------------- Serial Display Output --------------------
      char msg[160];
```

```
        sprintf(msg,
                "\x1B[H\x1B[?25lPitch=%d\xB0\r\nRoll=%d\xB0\r\nYaw=%d\xB0\r\n"
                "Kp = %d.%02d\r\nKi = 0.%03d\r\nKd = 0.%03d\r\n",
                (int)(-smooth_pitch - 0.5f),
                (int)(smooth_roll + 0.5f),
                (int)(smooth_yaw + 0.5f),
                (int)(kps * 100) / 100, abs((int)(kps * 100) % 100),
                (int)(kis * 1000), (int)(kds * 1000));
        LPUART_Print(msg);

        delay_us(1000);  // 1 ms delay for next iteration
    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK) {
        Error_Handler();
    }

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.MSIState = RCC_MSI_ON;
    RCC_OscInitStruct.MSICalibrationValue = 0;
    RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
        Error_Handler();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
                                  RCC_CLOCKTYPE_SYSCLK |
                                  RCC_CLOCKTYPE_PCLK1 |
                                  RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK) {
        Error_Handler();
    }
}

void Error_Handler(void)
{
    __disable_irq();
    while (1) {
        // trap forever
    }
}

#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
```

```
{
   // user can print error location here
}
#endif
```

I2C.h

```
/* ---------------------------------------------------------------------------
 * EE 329 - I2C HEADER FOR MPU6050 INTERFACE
 *
 * @file     : I2C.h
 * @brief    : Header declarations for I2C communication and MPU6050 access
 * @project  : EE329 MPU-6050 Interface
 * @author   : Nathan Heil
 * @version  : 1.0
 * @date     : 2025-05-18
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------- */

#ifndef INC_I2C_H_
#define INC_I2C_H_

#include "stm32l4xx.h"
#include "delay.h"

/* ---------------------------------------------------------------------------
 * I2C Hardware Initialization and Communication
 * ------------------------------------------------------------------------- */
void    I2C_Init( void );                                       // Initialize I2C1
with GPIOB
void    I2C_WriteByte( uint8_t dev_addr,
                       uint8_t reg_addr,
                       uint8_t data );                          // Write single
byte to register
uint8_t I2C_ReadByte( uint8_t dev_addr,
                      uint8_t reg_addr );                       // Read single byte
from register
void    I2C_ReadBytes( uint8_t dev_addr,
                       uint8_t reg_addr,
                       uint8_t* buffer,
                       uint8_t len );                           // Read multiple
bytes

/* ---------------------------------------------------------------------------
 * MPU-6050 Sensor Control Functions
 * ------------------------------------------------------------------------- */
void    MPU6050_Init( void );                                   // Initialize and
configure sensor
void    MPU6050_ReadGyro( int16_t* gx,
                          int16_t* gy,
                          int16_t* gz );                        // Read gyroscope
values
void    MPU6050_ReadAccel( int16_t* ax,
```

```
                        int16_t* ay,
                        int16_t* az );                        // Read
accelerometer values

#endif /* INC_I2C_H_ */
```

I2C.c

```
/* ---------------------------------------------------------------------------
 * EE 329 - I2C DRIVER FOR MPU6050 IMU SENSOR
 *
 * @file     : I2C.c
 * @brief    : Implements low-level I2C functions for MPU6050 access.
 *             Supports byte and burst reads, and gyro/accel interface.
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * @toolchain: STM32CubeIDE v1.12.0
 * ------------------------------------------------------------------------- */

#include "I2C.h"
#include "delay.h"

/* ---------------------------------------------------------------------------
 * I2C_Init()
 * Configures GPIOB and I2C1 peripheral for I2C communication.
 * Uses PB8 (SCL) and PB9 (SDA) with AF4 function.
 * ------------------------------------------------------------------------- */
void I2C_Init(void) {
   RCC->AHB2ENR  |= RCC_AHB2ENR_GPIOBEN;

   GPIOB->MODER  &= ~(GPIO_MODER_MODE8 | GPIO_MODER_MODE9);
   GPIOB->MODER  |=  (GPIO_MODER_MODE8_1 | GPIO_MODER_MODE9_1);
   GPIOB->OTYPER |=  (GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9);
   GPIOB->PUPDR  &= ~(GPIO_PUPDR_PUPD8 | GPIO_PUPDR_PUPD9);
   GPIOB->OSPEEDR |= ((3 << GPIO_OSPEEDR_OSPEED8_Pos) |
                      (3 << GPIO_OSPEEDR_OSPEED9_Pos));
   GPIOB->AFR[1] &= ~((0xF << GPIO_AFRH_AFSEL8_Pos) |
                      (0xF << GPIO_AFRH_AFSEL9_Pos));
   GPIOB->AFR[1] |=  ((0x4 << GPIO_AFRH_AFSEL8_Pos) |
                      (0x4 << GPIO_AFRH_AFSEL9_Pos));

   RCC->APB1ENR1 |= RCC_APB1ENR1_I2C1EN;
   I2C1->CR1     &= ~I2C_CR1_PE;
   I2C1->CR1     &= ~I2C_CR1_ANFOFF;
   I2C1->CR1     &= ~I2C_CR1_DNF;
   I2C1->TIMINGR  = 0x00000508;              // Set bus speed
   I2C1->CR2     &= ~I2C_CR2_ADD10;
   I2C1->CR1     |= I2C_CR1_PE;              // Enable I2C
}

/* ---------------------------------------------------------------------------
 * I2C_WriteByte()
```

```
 * Writes a single byte to a register of a slave device.
 * dev_addr  - 7-bit device address
 * reg_addr  - register address to write to
 * data      - byte to write
 * ------------------------------------------------------------------------- */
void I2C_WriteByte(uint8_t dev_addr, uint8_t reg_addr, uint8_t data) {
   I2C1->CR2 = 0;
   I2C1->CR2 |= (dev_addr << 1);
   I2C1->CR2 |= (2 << I2C_CR2_NBYTES_Pos);
   I2C1->CR2 &= ~I2C_CR2_RD_WRN;
   I2C1->CR2 |= I2C_CR2_START | I2C_CR2_AUTOEND;

   while (!(I2C1->ISR & I2C_ISR_TXIS));
   I2C1->TXDR = reg_addr;
   while (!(I2C1->ISR & I2C_ISR_TXIS));
   I2C1->TXDR = data;
   while (!(I2C1->ISR & I2C_ISR_STOPF));
   I2C1->ICR |= I2C_ICR_STOPCF;
}

/* ---------------------------------------------------------------------------
 * I2C_ReadByte()
 * Reads a single byte from a register of a slave device.
 * dev_addr  - 7-bit device address
 * reg_addr  - register address to read from
 * returns   - the byte read from the register
 * ------------------------------------------------------------------------- */
uint8_t I2C_ReadByte(uint8_t dev_addr, uint8_t reg_addr) {
   uint8_t data;
   I2C_ReadBytes(dev_addr, reg_addr, &data, 1);
   return data;
}

/* ---------------------------------------------------------------------------
 * I2C_ReadBytes()
 * Performs burst read from I2C slave starting at reg_addr.
 * dev_addr - 7-bit device address
 * reg_addr - start register
 * buffer   - pointer to destination buffer
 * len      - number of bytes to read
 * ------------------------------------------------------------------------- */
void I2C_ReadBytes(uint8_t dev_addr, uint8_t reg_addr, uint8_t* buffer, uint8_t len)
{
   I2C1->CR2 = 0;
   I2C1->CR2 |= (dev_addr << 1);
   I2C1->CR2 |= (1 << I2C_CR2_NBYTES_Pos);
   I2C1->CR2 &= ~I2C_CR2_RD_WRN;
   I2C1->CR2 |= I2C_CR2_START;

   while (!(I2C1->ISR & I2C_ISR_TXIS));
   I2C1->TXDR = reg_addr;
   while (!(I2C1->ISR & I2C_ISR_TC));

   I2C1->CR2 = 0;
   I2C1->CR2 |= (dev_addr << 1) | I2C_CR2_RD_WRN;
```

```
   I2C1->CR2 |= (len << I2C_CR2_NBYTES_Pos);
   I2C1->CR2 |= I2C_CR2_START | I2C_CR2_AUTOEND;

   for (uint8_t i = 0; i < len; i++) {
      while (!(I2C1->ISR & I2C_ISR_RXNE));
      buffer[i] = I2C1->RXDR;
   }

   while (!(I2C1->ISR & I2C_ISR_STOPF));
   I2C1->ICR |= I2C_ICR_STOPCF;
}

/* ---------------------------------------------------------------------------
 * MPU6050_Init()
 * Initializes MPU6050 IMU sensor by waking it up and setting gyro config.
 * Uses I2C_WriteByte to write to power and config registers.
 * ------------------------------------------------------------------------- */
void MPU6050_Init(void) {
   I2C_WriteByte(0x69, 0x6B, 0x00);  // Wake up device
   delay_us(100000);                 // Wait 100ms
   I2C_WriteByte(0x69, 0x1B, 0x00);  // Set gyro range to ±250 dps
}

/* ---------------------------------------------------------------------------
 * MPU6050_ReadGyro()
 * Reads 6 bytes from the gyro output registers and converts to signed values.
 * gx, gy, gz - pointers to destination integers for gyro data
 * ------------------------------------------------------------------------- */
void MPU6050_ReadGyro(int16_t* gx, int16_t* gy, int16_t* gz) {
   uint8_t data[6];
   I2C_ReadBytes(0x69, 0x43, data, 6);

   *gx = (data[0] << 8) | data[1];
   *gy = (data[2] << 8) | data[3];
   *gz = (data[4] << 8) | data[5];
}

/* ---------------------------------------------------------------------------
 * MPU6050_ReadAccel()
 * Reads 6 bytes from the accelerometer output registers and stores values.
 * ax, ay, az - pointers to destination integers for accel data
 * ------------------------------------------------------------------------- */
void MPU6050_ReadAccel(int16_t* ax, int16_t* ay, int16_t* az) {
   uint8_t data[6];
   I2C_ReadBytes(0x69, 0x3B, data, 6);

   *ax = (data[0] << 8) | data[1];
   *ay = (data[2] << 8) | data[3];
   *az = (data[4] << 8) | data[5];
}
```

PID.h

```
/* ---------------------------------------------------------------------------
 * EE 329 - PID CONTROL HEADER
```

```
 *
 * @file     : pid.h
 * @brief    : PID controller data structure and function declarations
 * @project  : EE329 Closed-Loop Control Systems
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------- */

#ifndef PID_H
#define PID_H

#include <stdint.h>

/* -------------------------------------------------------------------------
 * PID_t Structure
 * Stores gain parameters, state variables, and output constraints.
 * ------------------------------------------------------------------------- */
typedef struct {
   float kp;            // Proportional gain
   float ki;            // Integral gain
   float kd;            // Derivative gain
   float prev_error;    // Previous error (for derivative)
   float integral;      // Accumulated integral
   float output_min;    // Minimum output limit
   float output_max;    // Maximum output limit
} PID_t;

/* -------------------------------------------------------------------------
 * Function Prototypes
 * ------------------------------------------------------------------------- */
void  PID_Init(PID_t* pid,
               float kp,
               float ki,
               float kd,
               float min_out,
               float max_out);   // Initializes PID gains and constraints

float PID_Compute(PID_t* pid,
                  float setpoint,
                  float measurement,
                  float dt);     // Computes PID output based on input and dt

#endif /* PID_H */
```

PID.c

```
/* -------------------------------------------------------------------------
 * EE 329 - PID CONTROL MODULE
 *
 * @file     : pid.c
 * @brief    : Implements a configurable PID controller with saturation limits
 * @project  : EE329 Closed-Loop Control Systems
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
```

```
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------- */

#include "pid.h"

/* ---------------------------------------------------------------------------
 * PID_Init()
 * Initializes PID gain parameters and output constraints.
 *
 * pid       - pointer to PID_t struct instance
 * kp        - proportional gain
 * ki        - integral gain
 * kd        - derivative gain
 * min_out   - minimum output saturation limit
 * max_out   - maximum output saturation limit
 * ------------------------------------------------------------------------- */
void PID_Init(PID_t* pid, float kp, float ki, float kd, float min_out, float max_out)
{
   pid->kp         = kp;
   pid->ki         = ki;
   pid->kd         = kd;
   pid->prev_error = 0.0f;
   pid->integral   = 0.0f;
   pid->output_min = min_out;
   pid->output_max = max_out;
}

/* ---------------------------------------------------------------------------
 * PID_Compute()
 * Calculates the PID control output based on current error.
 * Applies anti-windup clamping to output limits.
 *
 * pid        - pointer to PID_t struct instance
 * setpoint   - target value
 * measurement- current system value
 * dt         - time step in seconds
 * returns    - PID output (clamped within bounds)
 * ------------------------------------------------------------------------- */
float PID_Compute(PID_t* pid, float setpoint, float measurement, float dt) {
   float error      = setpoint - measurement;
   pid->integral   += error * dt;
   float derivative = (error - pid->prev_error) / dt;

   float output = (pid->kp * error) +
                  (pid->ki * pid->integral) +
                  (pid->kd * derivative);

   if (output > pid->output_max) {
      output = pid->output_max;
   } else if (output < pid->output_min) {
      output = pid->output_min;
   }
```

```
   pid->prev_error = error;
   return output;
}
```

PWM.h

```
/* ---------------------------------------------------------------------------
 * EE 329 - PWM DRIVER HEADER
 *
 * @file     : pwm.h
 * @brief    : Header for PWM control functions using TIM2 and TIM3
 * @project  : EE329 Embedded Actuation / Servo Control
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * --------------------------------------------------------------------------- */

#ifndef PWM_H
#define PWM_H

#include <stdint.h>

/* ---------------------------------------------------------------------------
 * Function Prototypes
 * --------------------------------------------------------------------------- */

/**
 * @brief  Initializes TIM2 and TIM3 for 50 Hz PWM on GPIOA pins
 */
void PWM_Init( void );

/**
 * @brief  Sets pulse width for a given PWM channel
 * @param  channel: Servo index (1–5)
 * @param  pulse_us: Desired pulse in microseconds (1000–2000 µs)
 */
void PWM_SetPulse( uint8_t channel,
                   uint16_t pulse_us );

/**
 * @brief  Converts a servo angle (in degrees) to a PWM pulse width
 * @param  angle: Input angle [-60°, +60°]
 * @return PWM pulse in microseconds [1000, 2000]
 */
uint16_t angle_to_pulse( float angle );

#endif /* PWM_H */
```

PWM.c

```
/* ---------------------------------------------------------------------------
```

```
 * EE 329 - PWM DRIVER MODULE
 *
 * @file     : pwm.c
 * @brief    : Generates PWM signals for up to 5 servos using TIM2 and TIM3
 * @project  : EE329 Embedded Actuation / Servo Control
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------- */

#include "pwm.h"
#include "stm32l4xx.h"

/* ---------------------------------------------------------------------------
 * PWM_Init()
 * Configures GPIOA pins and TIM2/TIM3 to generate 50 Hz PWM signals
 * TIM2 channels: PA0–PA3 for Servos 1–4
 * TIM3 channel1: PA6 for Servo 5
 * ------------------------------------------------------------------------- */
void PWM_Init(void) {
   // Enable GPIOA and required timer clocks
   RCC->AHB2ENR   |= RCC_AHB2ENR_GPIOAEN;
   RCC->APB1ENR1  |= RCC_APB1ENR1_TIM2EN;
   RCC->APB1ENR1  |= RCC_APB1ENR1_TIM3EN;

   // Configure PA0–PA3 for TIM2 CH1–CH4 alternate function (AF1)
   for (int i = 0; i < 4; i++) {
      GPIOA->MODER &= ~(0x3 << (i * 2));
      GPIOA->MODER |=  (0x2 << (i * 2));
      GPIOA->AFR[0] &= ~(0xF << (i * 4));
      GPIOA->AFR[0] |=  (0x1 << (i * 4));
   }

   // TIM2 setup: 50 Hz PWM (20 ms period)
   TIM2->PSC     = 79;      // 4 MHz / 80 = 50 kHz
   TIM2->ARR     = 999;     // 50 kHz / 1000 = 50 Hz

   TIM2->CCMR1   = (6 << TIM_CCMR1_OC1M_Pos) | TIM_CCMR1_OC1PE |
                   (6 << TIM_CCMR1_OC2M_Pos) | TIM_CCMR1_OC2PE;
   TIM2->CCMR2   = (6 << TIM_CCMR2_OC3M_Pos) | TIM_CCMR2_OC3PE |
                   (6 << TIM_CCMR2_OC4M_Pos) | TIM_CCMR2_OC4PE;

   TIM2->CCER   |= TIM_CCER_CC1E | TIM_CCER_CC2E |
                   TIM_CCER_CC3E | TIM_CCER_CC4E;

   TIM2->CR1    |= TIM_CR1_ARPE;
   TIM2->EGR    |= TIM_EGR_UG;
   TIM2->CR1    |= TIM_CR1_CEN;

   // Configure PA6 for TIM3 CH1 alternate function (AF2)
   GPIOA->MODER  &= ~(0x3 << (6 * 2));
   GPIOA->MODER  |=  (0x2 << (6 * 2));
   GPIOA->AFR[0] &= ~(0xF << GPIO_AFRL_AFSEL6_Pos);
   GPIOA->AFR[0] |=  (0x2 << GPIO_AFRL_AFSEL6_Pos);
```

```
   // TIM3 setup: 50 Hz PWM on CH1 (Servo 5)
   TIM3->PSC     = 79;
   TIM3->ARR     = 999;
   TIM3->CCMR1  |= (6 << TIM_CCMR1_OC1M_Pos) | TIM_CCMR1_OC1PE;
   TIM3->CCER   |= TIM_CCER_CC1E;
   TIM3->CR1    |= TIM_CR1_ARPE;
   TIM3->EGR    |= TIM_EGR_UG;
   TIM3->CR1    |= TIM_CR1_CEN;
}

/* ---------------------------------------------------------------------------
 * PWM_SetPulse()
 * Sets pulse width in microseconds for a specified servo channel.
 * Acceptable range is 1000–2000 us. Converts to 50 Hz ticks.
 *
 * channel   - Servo output (1–5)
 * pulse_us  - Pulse width in microseconds
 * ------------------------------------------------------------------------- */
void PWM_SetPulse(uint8_t channel, uint16_t pulse_us) {
   if (pulse_us > 2000) pulse_us = 2000;
   if (pulse_us < 1000) pulse_us = 1000;

   uint16_t ticks = (pulse_us * 50) / 1000; // convert us to 50 kHz ticks

   switch (channel) {
      case 1: TIM2->CCR1 = ticks; break;
      case 2: TIM2->CCR2 = ticks; break;
      case 3: TIM2->CCR3 = ticks; break;
      case 4: TIM2->CCR4 = ticks; break;
      case 5: TIM3->CCR1 = ticks; break;
      default: break;
   }
}

/* ---------------------------------------------------------------------------
 * angle_to_pulse()
 * Converts an angle in degrees (±60°) to a corresponding pulse width in µs.
 * Maps angle to range [1000, 2000] µs (i.e. 1–2 ms typical servo range).
 *
 * angle - desired angle from -60 to +60 degrees
 * returns - PWM pulse width in microseconds
 * ------------------------------------------------------------------------- */
uint16_t angle_to_pulse(float angle) {
   if (angle > 60.0f) angle = 60.0f;
   if (angle < -60.0f) angle = -60.0f;

   float norm = (angle + 60.0f) / 120.0f;
   return (uint16_t)(1000 + norm * 1000);
}
```

---

delay.h

```
   /**
 ******************************************************************************
```

```
 * @file     : delay.h
 * @project  : EE329 Lab A3
 * @author   : Nathan Heil
 * @date     : 2025-05-18
 * @brief    : Header for microsecond delay functions using SysTick
 ******************************************************************************
 */

#ifndef INC_DELAY_H_
#define INC_DELAY_H_

#include <stdint.h>     // defines uint32_t

/* ---------------- Function Prototypes ---------------- */
void SysTick_Init(void);                    // sets up SysTick timer
void delay_us(const uint32_t time_us);      // delays for time_us µs

#endif /* INC_DELAY_H_ */
```

---

delay.c

```
   /**
 ******************************************************************************
 * @file     : delay.c
 * @project  : EE329 Lab A3
 * @author   : Nathan Heil
 * @date     : 2025-05-18
 * @brief    : Software microsecond delay using SysTick hardware timer
 * target    : STM32L4A6ZG @ 4 MHz
 ******************************************************************************
 */

#include "delay.h"
#include "stm32l4xx.h"
/* ------------------------------ SysTick_Init -------------------------------
 * Configures the ARM SysTick timer to run off the core clock.
 * Interrupts are disabled for precise microsecond timing.
 * Warning: this disables HAL_Delay functionality.
 * ---------------------------------------------------------------------------- */
void SysTick_Init(void) {
   SysTick->CTRL |= (SysTick_CTRL_ENABLE_Msk |
                     SysTick_CTRL_CLKSOURCE_Msk);  // enable SysTick
   SysTick->CTRL &= ~SysTick_CTRL_TICKINT_Msk;     // disable SysTick IRQ
}

/* ------------------------------- delay_us ----------------------------------
 * Generates a blocking delay of time_us microseconds using the SysTick timer.
 *
 * Inputs :
 *   time_us : number of microseconds to delay (must be > 0)
 *
 * Returns :
 *   none (blocks until time_us has passed)
 *
 * Notes :
```

```
 *   @4 MHz, delay_us(1) results in ~10-15 us due to setup overhead.
 * ------------------------------------------------------------------------- */
void delay_us(const uint32_t time_us) {
   if (time_us == 0) return;

   // Calculate timer reload value from system clock
   SysTick->LOAD = (uint32_t)((time_us * (SystemCoreClock / 1000000)) - 1);
   SysTick->VAL  = 0;                                  // clear current count
   SysTick->CTRL &= ~SysTick_CTRL_COUNTFLAG_Msk;  // clear overflow flag

   // Wait for COUNTFLAG to set when timer expires
   while (!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk));
}
```

---

uart.h

```
/* ------------------------------------------------------------------------------
 * EE 329 - UART DRIVER MODULE
 *
 * @file     : uart.h
 * @brief    : Header file for LPUART initialization and data display utilities
 * @project  : EE329 UART-based Gyro Data Monitor
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------- */

#ifndef INC_UART_H_
#define INC_UART_H_

#include <stdint.h>

/* ------------------------------------------------------------------------------
 * Function Prototypes
 * ------------------------------------------------------------------------- */

/**
 * @brief  Initializes LPUART1 and configures GPIOG pins PG7 (TX), PG8 (RX)
 */
void LPUART_init(void);

/**
 * @brief  Sends a null-terminated string over UART
 * @param  msg: Pointer to C-string to transmit
 */
void LPUART_Print(const char *msg);

/**
 * @brief  Prints formatted gyro values over UART
 * @param  gx: Gyro X-axis value
 * @param  gy: Gyro Y-axis value
 * @param  gz: Gyro Z-axis value
 */
void UART_PrintGyroData(int16_t gx, int16_t gy, int16_t gz);
```

```c
#endif /* INC_UART_H_ */
```

---

uart.c

```c
/* ------------------------------------------------------------------------
 * EE 329 - UART DRIVER MODULE
 *
 * @file     : uart.c
 * @brief    : Provides LPUART1 initialization and serial print support
 *             Includes UART interrupt handler and live tuning input buffer
 * @project  : EE329 UART-based Gyro Data Monitor
 * @author   : Nathan H., Dante B., Benjamin T., Gulianna C.
 * @version  : 1.0
 * @date     : 6/5/25
 * @target   : STM32 NUCLEO-L4A6ZG
 * ------------------------------------------------------------------------ */

#include "stm32l4xx.h"
#include "uart.h"
#include <stdio.h>

/* ------------------------------------------------------------------------
 * LPUART_init()
 * Configures LPUART1 and GPIOG pins for asynchronous serial communication
 * PG7 = TX, PG8 = RX, alternate function AF8
 * ------------------------------------------------------------------------ */
void LPUART_init(void) {
   PWR->CR2        |= PWR_CR2_IOSV;
   RCC->AHB2ENR    |= RCC_AHB2ENR_GPIOGEN;
   RCC->APB1ENR2   |= RCC_APB1ENR2_LPUART1EN;

   GPIOG->MODER   &= ~((3 << 14) | (3 << 16));
   GPIOG->MODER   |=  ((2 << 14) | (2 << 16));
   GPIOG->AFR[0]   = (GPIOG->AFR[0] & ~(0xF << 28)) | (8 << 28);
   GPIOG->AFR[1]   = (GPIOG->AFR[1] & ~(0xF << 0))  | (8 << 0);
   GPIOG->OSPEEDR |= (3 << 14) | (3 << 16);
   GPIOG->OTYPER  &= ~(GPIO_OTYPER_OT7 | GPIO_OTYPER_OT8);
   GPIOG->PUPDR   &= ~(GPIO_PUPDR_PUPD7 | GPIO_PUPDR_PUPD8);

   LPUART1->CR1   &= ~USART_CR1_UE;              // Disable UART
   LPUART1->CR1   &= ~(USART_CR1_M1 | USART_CR1_M0);
   LPUART1->CR1   |= (USART_CR1_TE | USART_CR1_RE);  // TX and RX enable
   RCC->CCIPR     &= ~RCC_CCIPR_LPUART1SEL;
   RCC->CCIPR     |= RCC_CCIPR_LPUART1SEL_0;     // PCLK1 clock source
   LPUART1->BRR    = 0x22b9;                     // 9600 baud @ 4 MHz
   LPUART1->CR1   |= USART_CR1_UE;               // Enable UART
   LPUART1->CR1   |= USART_CR1_RXNEIE;           // Enable RX interrupt
}

/* ------------------------------------------------------------------------
 * LPUART_Print()
 * Sends a null-terminated string over UART
 * @param message: Pointer to string
 * ------------------------------------------------------------------------ */
```

```c
void LPUART_Print(const char* message) {
   while (*message) {
      while (!(LPUART1->ISR & USART_ISR_TXE));
      LPUART1->TDR = *message++;
   }
}

/* ---------------------------------------------------------------------------
 * UART_PrintGyroData()
 * Formats and prints gyro X, Y, Z values to UART terminal
 * @param gx, gy, gz: Raw gyroscope values
 * ------------------------------------------------------------------------- */
void UART_PrintGyroData(int16_t gx, int16_t gy, int16_t gz) {
   char buf[64];
   sprintf(buf, "\x1B[HGyX=%d\tGyY=%d\tGyZ=%d\r\n", gx, gy, gz);
   LPUART_Print(buf);
}

/* ---------------------------------------------------------------------------
 * UART live tuning input buffer
 * Stores up to 9 digits for Kp, Ki, Kd via UART input
 * ------------------------------------------------------------------------- */
int digit_buf[9]  = {1, 6, 0, 0, 1, 0, 1, 0, 0};  // Default PID values
int digit_idx     = 0;                            // Current index
int digit_ready   = 0;                            // Flag to commit buffer

/* ---------------------------------------------------------------------------
 * LPUART1_IRQHandler()
 * UART interrupt handler for character reception
 * Accepts digit characters and populates buffer
 * Sets digit_ready on newline
 * ------------------------------------------------------------------------- */
void LPUART1_IRQHandler(void) {
   if (LPUART1->ISR & USART_ISR_RXNE) {
      char ch = LPUART1->RDR;

      // Finish input on Enter key
      if (ch == '\r' || ch == '\n') {
         digit_ready = 1;
         digit_idx = 0;   // Reset index for next entry
      }

      // Accept digits only
      else if (ch >= '0' && ch <= '9' && digit_idx < 9) {
         digit_buf[digit_idx++] = ch - '0';
      }
   }
}
```