- **Cubli Source Code**
  - **CmakeList.txt** — Cross-platform build configuration
  - **platformio.ini** — Pio configuration
  - **include** — Library files
    - bsp
      - ESP-BSP library
        - bspdriver.h
        - adc.h
        - can.h
        - key.h
        - led.h
        - my_eeprom.h
    - config — config.h
    - common — time.h
    - control
      - cubli_mini.h
      - param.h
      - serial_commander.h
      - wifi_commander.h
    - imu
      - ahrs.h
      - mpu6050.h
  - **lib** — Fetching from Platformio Library Manager
    - ESP32CAN
    - Simple FOC
  - **src**
    - bsp
      - can.cpp
      - key.cpp
      - led.cpp
      - my_eeprom.cpp
    - control
      - cubli_mini.cpp
      - serial_commander.cpp
      - wifi_commander.cpp
      - param.cpp
    - imu
      - ahrs.cpp
      - mpu6050_driver.cpp
    - main.cpp
    - Source File — NameSpace
      - ImuDriver — MPU6050Driver
      - Imu — AHRS
      - Control
        - SerialCommander
        - WifiCommander
        - CubliminiControl
      - Comm — Time
      - BSP — BSPDriver
        - AdcDriver
        - LedDriver
        - KeyDriver
        - CanDriver

IMU (MPU6050)

1. Namespace Declaration — `using namespace CubliMini::Imu` / `using namespace CubliMini::ImuDriver;`
2. Instantiate AHRS Class — `AHRS ahrs;`
3. IMU Initialization
   - IIC initialization — `I2C. -> begin(_sda_pin, _scl_pin, (uint32_t)400000);`
   - MPU6050 initialization — `if(MPU6050Init() == true)`
   - Set Accelerometer and Gyroscope Range
     - `SetAccRange(AccConfig_e:=ACC16_4G);`
     - `SetGyroRange(GyroConfig_e:=GYRO_1000DEG);`
   - Interrupt config initialization — `MPU6050SetInterruptInit();`
     - `pinMode(Interrupt_pin_, INPUT_PULLUP);`
     - `attachInterrupt(Interrupt_Pin_, InterruptFuntion, FALLING);`
   - Offset calculation (Mean Filter)
     - `CalGyroOffset();`
       - `MPU6050ReadData(offset_raw_data, offset_data);`
       - `I2cRead(MPU6050_I2CADDR_DEFAULT, MPU6050_ACCEL_OUT, mpu_buf, 14);`
       - `MeanFilter(_imu_raw_data);`
       - Read & Mean filter
     - `printf("IMU: calibrate offset success, gx: %f gy: %f gz: %f\r\n",`
4. Sensor Update Task Creation — `cubli_mini.SensorUpdate();`
   - ① If (ahrs_->imuUpdate())
     - 1. Interruption & Fetching data
       - `mpu6050_driver.ResetMPU6050Interruptflag();`
       - `mpu6050_driver.ImuGetRawValue(imu_data_, imu_raw_data_);`
     - 2. IMU Static State Detection — `ImuStatic();`
     - 3. Mahony Filter to Euler Angle Convertion
       - `ImuAHRSUpdate(q_, imu_data_);`
       - `ConvertToEulerAngleBy0(imu_data_.angle, q_);`
   - ② Model Attitude Angle Convertion
     - `p_balance_.sensor.x.angle = ahrs_->imu_data_.angle.pitch;`
     - `p_balance_.sensor.x.gyro = ahrs_->imu_data_.gyro.gx;`
     - `p_balance_.sensor.y.angle = ahrs_->imu_data_.angle.roll;`
     - `p_balance_.sensor.y.gyro = ahrs_->imu_data_.gyro.gy;`
     - `p_balance_.sensor.z.angle = ahrs_->imu_data_.angle.yaw;`
     - `p_balance_.sensor.z.gyro = ahrs_->imu_data_.gyro.gz;`
   - ③ CAN Communication — `bsp_driver_can_CanSetMsg(motor_get_speed_ch2, motor_get_speed_ch3);`
     - 1. CAN Frame — `CAN_frame_t rx_frame;`
     - 2. Queue Storage
       - `if (xQueueReceive(CAN_cfg.rx_queue, &rx_frame, 0) == pdTRUE)`
       - `if (rx_frame.FIR.B.FF == CAN_Frame_std && rx_frame.FIR.B.RTR == CAN_no_RTR)`
       - `if (rx_frame.MsgID == mCAN_GET_MOTOR_SPEED_FRAME)`
       - `_get_ch2_speed = -(float)(rx_frame.data.u8[0] |`
         `rx_frame.data.u8[1] << 8 |`
         `rx_frame.data.u8[2] << 16 |`
         `rx_frame.data.u8[3] << 24) / 1000.0F;`
       - `_get_ch3_speed = -(float)(rx_frame.data.u8[4] |`
         `rx_frame.data.u8[5] << 8 |`
         `rx_frame.data.u8[6] << 16 |`
         `rx_frame.data.u8[7] << 24) / 1000.0F;`
       - `rx_frame_count_++;`
       - `if (rx_frame_count_ > CAN_OFFLINE_COUNT){`
         `can_is_online_ = eOFF_LINE;`
         `}`
       - `return can_is_online_;`
     - 3. Online(Full) Judgment — `CanIsOnline();`

Serial & Wifi communication

1. Serial Class Declaration — `SerialCommander serial_commander;`
2. RTOS Tasks — `xTaskCreatePinnedToCore(ModeTask, "ModeTask", 1000, NULL, 2, NULL, 1);`
   - ① Serial Fetching
     - `serial_commander.Run(Serial, cubli_mini);`
     - `if (isSentinel(ch))` — `GetDrSet(received_chars_, control);`
   - ② PID & Median Param Calibration
     - `cubli_mini.PBalanceAutoCalibration();`
     - `cubli_mini.VBalanceAutoCalibration();`
3. Wifi Class Declaration — `WifiCommander wifi_commander;`
   - Wifi Account & Password, TCP, Port number Settings
4. Wifi Connection — `WifiConnectStatus_e WifiCommander::ConnectWiFi()`
   - ① Wifi Information Input — `wifi.begin(ssid_c_str(), password_c_str());`
   - ② If Connection Succeed — `wifi_connect_status_ = WifiConnectStatus_e::eWIFI_SUCCESS;`
     - I. Wifi Task Creation — `void WifiCommanderTask(void *parameter)`
     - II. Customer Server Creation — `wifi_commander.TpClientInpacks(cubli_mini);`

Main Functions

- IMU: MPU6050 Gyroscope & Accelerometer — Mahony Filter (AHRS algorithm)
- Internal and External Communications
  - Serial & Wifi communication
  - CAN communication between MCU1 and MCU2
  - IIC communication of IMU
- Basic Peripheral Collection (Key & LED)
- Power Supply ADC Acquisition
- FreeRTOS Multitasking
- EEPROM Writing (PID)
- Simple FOC Motor Control

Cubli Control

1. Namespace Declaration — `using namespace CubliMini::Control;`
2. Instantiate Control Class — `CubliMiniControl cubli_mini;`
3. Initialization — `void CubliMiniControl::Init()`
   - `ParamInit(p_balance_.param, u_ch2_balance_.param, wifi_param_);`
     - ① `EepromInit();` — `EEPROM.begin(4096);`
     - ② `ReadPBalanceParam(p_balance_.param);`
     - ③ `ReadVBalanceParam(u_balance_.param);`
     - ④ `ReadWifiParam(wifi_param);`
     - `ReadP_PARAM_ADDR, (uint8_t *)&_param, sizeof(_param));`
   - `bsp_driver_.Init();`
     - `blue_led_.Init(BLUE_LED_PIN);`
     - `green_led_.Init(GREEN_LED_PIN);`
     - `motor_green_led_.Init(MOTOR_GREEN_LED_PIN);`
     - ① LEDs and Keys
       - `key1_.Init(KEY_1_PIN);`
       - `key2_.Init(KEY_2_PIN);`
     - ② Voltage ADC — `bat_voltage_.Init(BAT_VOLTAGE_PIN);`
     - ③ CAN Bus — `can_.Init(CAN_TXD_PIN, CAN_RXD_PIN, CAN_SPEED_1000KBPS);`
4. RTOS Mode Control — `xTaskCreatePinnedToCore(ModeTask, "ModeTask", 5000, NULL, 2, NULL, 1);` — `cubli_mini.ModeControl();`
   - ① Fetching Key Status & Actions
     - `KeyAction_e key1_action = bsp_driver_.key1_.GetKeyAction();`
     - `KeyAction_e key1_action = bsp_driver_.key2_.GetKeyAction();`
     - Status: `eKEY_STATUS_NORMAL = 0,` `eKEY_STATUS_UP = 1,` `eKEY_STATUS_DOWN = 2,` `eKEY_STATUS_DLTPSS_DOWN = 3,`
     - Action: `eKEY_ACTION_NO_PRESS = 0x00,` `eKEY_ACTION_SHORT_PRESS = 0x01,` `eKEY_ACTION_LONG_PRESS = 0x02,`
   - Accordingly
   - ② Control Mode Switch — `switch (control_mode_)`
     - `eNORMAL = 0x00,`
     - `ePCONT_BALANCE = 0x01,`
     - `ePCONT_CALIBRATION = 0x02,`
     - `eVCONT_BALANCE = 0x03,`
     - `eVACILATER_BALANCE = 0x04,`
     - `eVACILATER_CALIBRATION = 0x05,`
     - `eVACILATER_CALIBRATIONING = 0x07,`
     - `eMOTOR_TEST = 0x08`
   - ③ Voltage Threshold Definition
     - `#define LOW_POWER_VOLTAGE_THRESHOLD 8`
     - `if (bsp_driver_.GetBatVoltage() < LOW_POWER_VOLTAGE_THRESHOLD)`
       - `bsp_driver_.bule_led_.AlwaysOn();`
       - `SystemLow_low_power_ = true;`
       - `control_mode_ = eNORMAL;`
   - ④ Motor Drive & CAN Status Indication — `uint8_t temp_status, = (uint8_t)bsp_driver_.can_.can_is_online_ << 1 | motor_bit_status_ch1;`
5. RTOS Control Task
   - `if (system_low_power_ == false) switch (control_mode_)`
     - Sufficient Voltage (> 8 V)
       - `case eNORMAL: MotorStop();`
       - `case ePCONT_BALANCE: PBalance();`
         - ① Control Threshold Sensoring
           - `if (fabs(p_balance_.sensor.x.angle - p_balance_.param.x.angle_offset) < protection_threshold_ && fabs(p_balance_.sensor.y.angle - p_balance_.param.y.angle_offset) < protection_threshold_ && can_is_online_ == false)`
           - `p_balance_.sensor.x.speed`
           - `p_balance_.sensor.y.speed`
           - `p_balance_.sensor.z.speed`
         - ② Speed Control
           - I. Fetching Curent Speed — `void CubliMiniControl::GetAxisMoveSpeed();`
           - II. Calibrating Speed Param
             - `float x_axis_speed = (arip_balance_.sensor.x, p_balance_.param.x);`
             - `float y_axis_speed = (arip_balance_.sensor.y, p_balance_.param.y);`
             - `float z_axis_speed = (arip_balance_.sensor.z, p_balance_.param.z);`
         - ③ Speed Limit — `float CubliMiniControl::Limit(float _data, float _limit)`
           - `if (_data >= _limit){ _data = _limit;}`
           - `if (_data <= -_limit){ _data = -_limit;}`
       - `case eVCONT_BALANCE: Ch2AccisBalance();`
         - ① Control Threshold Sensoring
         - ② Speed Control (Only Ch2)
           - `u_ch2_balance_.sensor.speed = motor_get_speed_ch2;`
           - `motor_set_speed_ch2 = (arip_ch2_balance_.sensor, u_ch2_balance_.param);`
       - `case ePCONT_CALIBRATION:`
       - `case eVACILATER_CALIBRATION:`
         - Execute Above cases — `MotorStop();`
       - `case eMOTOR_TEST:`
         - `motor_set_speed_ch1 = 0.7F;`
         - `motor_set_speed_ch2 = 0.7F;`
         - `motor_set_speed_ch3 = 0.7F;`
     - Insufficient Voltage (< 8 V)
       - `if (now_power_reminder_time_ MinMillis(1000) && system_low_power_ == true) MotorStop();`
6. CAN Sending — `bsp_driver_can_CanSendMotorSpeed(motor_set_speed_ch2, motor_set_speed_ch3);`
   - ① CAN Frame Structure (Library)
     - `tx_frame.FIR.B.FF = (CAN_frame_std);`
     - `tx_frame.MsgID = mCAN_SET_MOTOR_SPEED_FRAME;`
     - `tx_frame.FIR.B.DLC = 8;`
     - `tx_frame.data.u8[0] = (int32_t)(_set_ch2_speed * 1000.0f) & 0xff;`
     - `...`
     - `tx_frame.data.u8[4] = (int32_t)(_set_ch3_speed * 1000.0f) & 0xff;`
   - ② Protocol Frame
     - `/** \brief CAN Frame structure */`
     - `typedef struct {`
       - `CAN_FR_t FR; /**< \brief Frame information record*/`
       - `uint32_t MsgID; /**< \brief Message ID */`
       - `union {`
         - `uint8_t u8[8]; /**< \brief Payload byte access*/`
         - `uint32_t u32[2]; /**< \brief Payload u32 access*/`
       - `} data;`
       - `} CAN_frame_t;`
   - ③ Frame Sending — `ESP32CAN.CANWriteFrame(&tx_frame);`