QEADynamicPlanning

QEA小车最终测试（动态路径规划）

项目简介

本项目旨在利用激光雷达实现在固定矩形区域内扫描和检测方形垃圾桶作为障碍物，以及圆形垃圾桶作为目标物。通过使用梯度上升法自动规划最优路径，从机器的当前位置到达目标圆形垃圾桶，并且能够自动避开方形垃圾桶。在测试过程中，还会进行人为随机改动障碍物的位置。

PID算法编写

由于我们项目中会用到很多次PID算法，所以我们将PID算法写成库的形式供其他模块调用。

定义PID句柄结构体:

```
enum PIDType
{
    PIDType_Pos = 1, //位置式
    PIDType_Inc = 2, //增量式
};

volatile struct PIDHanldeDef
{
    double kp; //p比例系数
    double ki; //i比例系数
    double kd; //d比例系数
    double value; //实际测量值
    double target; //目标值
    double error; //误差
    double totalError; //总误差
    double lastError; //上一次误差
    double lastError2; //上上一次误差
    double minPWM; //最小输出PWM
    double maxPWM; //最大输出PWM
    double pwm; //最终输出PWM
    enum PIDType pidType; //PID算法类型
};
```

创建PID句柄

```
void CreatePID(volatile struct PIDHanldeDef* def, enum PIDType pidType,
double kp, double ki, double kd, double minPWM, double maxPWM)
{
    def->kp = kp;
    def->ki = ki;
    def->kd = kd;
    def->value = 0;
```

```
    def->target = 0;
    def->error = 0;
    def->totalError = 0;
    def->lastError = 0;
    def->lastError2 = 0;
    def->minPWM = minPWM;
    def->maxPWM = maxPWM;
    def->pwm = 0;
    def->pidType = pidType;
}
```

外部程序通过调用PIDTick完成一次PID计算

```
double PIDTick(volatile struct PIDHanldeDef* def, double newValue)
{
    def->value = newValue;
    switch (def->pidType) {
        case PIDType_Pos: //位置式PID
            PosTick(def);
            break;
        case PIDType_Inc: //增量式PID
            IncTick(def);
            break;
    }
    return def->pwm;
}
```

位置式PID实现原理

```
void PosTick(volatile struct PIDHanldeDef* def)
{
    def->lastError = def->error;
    def->error = def->target - def->value;
    def->totalError += def->error;

    double pwm = def->kp * def->error +
                 def->ki * def->totalError +
                 def->kd * (def->error - def->lastError);
    def->pwm += pwm;
    if (def->pwm > def->maxPWM) def->pwm = def->maxPWM;
    if (def->pwm < def->minPWM) def->pwm = def->minPWM;
}
```

增量式PID实现原理

```
void IncTick(volatile struct PIDHanldeDef* def)
{
```

```
    def->lastError2 = def->lastError;
    def->lastError = def->error;
    def->error = def->target - def->value;

    double pwm = def->kp * (def->error - def->lastError) +
                 def->ki * def->error +
                 def->kd * (def->error - 2 * def->lastError + def-
>lastError2);
    def->pwm += pwm;
    if (def->pwm > def->maxPWM) def->pwm = def->maxPWM;
    if (def->pwm < def->minPWM) def->pwm = def->minPWM;
}
```

电机速度PID控制

我们通过增量式PID算法实现小车电机速度的闭环控制。

电机PID控制结构体：

```
volatile struct WheelPWM
{
    volatile struct PIDHanldeDef pidHanldeDef; //pid算法句柄
    double speed; //电机速度
    double dis; //路程
    double pwm; //最终输出pwm
};
```

初始化电机PID控制

```
void Wheel_PID_Init()
{
    CreatePID(&leftPWM.pidHanldeDef, PIDType_Inc, 960, 240, 0, -1200,
1200);
    CreatePID(&rightPWM.pidHanldeDef, PIDType_Inc, 960, 240, 0, -1200,
1200);
}
```

完成一次电机PID计算并更新PWM

```
void Wheel_PID_Tick()
{
    //获取左右轮速度
    leftPWM.speed = (short) __HAL_TIM_GET_COUNTER(&htim8);
    __HAL_TIM_SET_COUNTER(&htim8, 0);
    rightPWM.speed = (short) __HAL_TIM_GET_COUNTER(&htim4);
    __HAL_TIM_SET_COUNTER(&htim4, 0);
```

```
    //通过速度积分计算行驶过的路程
    leftPWM.dis += leftPWM.speed;
    rightPWM.dis += rightPWM.speed;

    //计算一次pid
    leftPWM.pwm = PIDTick(&leftPWM.pidHanldeDef, leftPWM.speed);
    rightPWM.pwm = PIDTick(&rightPWM.pidHanldeDef, rightPWM.speed);

    //更新左轮pwm
    if (leftPWM.pwm > 0) {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, leftPWM.pwm);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    } else if (leftPWM.pwm < 0) {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 0);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, -leftPWM.pwm);
    } else {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 0);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    }

    //更新右轮pwm
    if (rightPWM.pwm > 0) {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, rightPWM.pwm);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
    } else if (rightPWM.pwm < 0) {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, -rightPWM.pwm);
    } else {
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
    }
}
```

启动电机PID速度控制

```
    //启动编码器
    HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_1 | TIM_CHANNEL_2);
    HAL_TIM_Encoder_Start(&htim8, TIM_CHANNEL_1 | TIM_CHANNEL_2);

    //开启pwm输出 驱动频率30kHz 0-1200
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);

    //开启编码器速度采集 1kHz
    HAL_TIM_Base_Start_IT(&htim7);
```

在定时器回调函数中调用PID控制算法：

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  //进行一次pid计算
  if (htim->Instance == TIM7) {
      Wheel_PID_Tick();
  }

  /* USER CODE END Callback 1 */
}
```

读取解析IMU数据

我们通过STM32CubeMX配置IMU串口和FreeRTOS，在DMA中断回调函数中使用双缓冲接收串口数据，并通过任务通知将接收到的数据传递给IMUDecoder任务来进行解析。 IMUDecoder任务函数：

```
void IMUDecoder(void *argument)
{
  /* USER CODE BEGIN IMUDecoder */
    imuTaskHandle = xTaskGetCurrentTaskHandle(); //将当前任务句柄送给imu
    /* Infinite loop */
    for(uint32_t cnt = 0; ; ++cnt)
    {
        if (HAL_UART_STATE_READY == HAL_UART_GetState(&huart2)) {
            HAL_UART_Receive_DMA(&huart2, imuRecvBuff, 128); //开启imu串口通
信
        }

        uint32_t value = 0;
        if (pdTRUE == xTaskNotifyWait(0, 0xffffffff, &value,
pdMS_TO_TICKS(100)) && value == 0x12345678) {
            while (!DecodeIMUPackage()) {} //解析imu数据包
        }
        osDelay(1);
    }

  /* USER CODE END IMUDecoder */
}
```

通过串口传输完一半和传输完回调函数实现双缓冲。
DMA中断回调函数：

```
void IMU_RxHalfCpltCallback()
{
    imuPackage = imuRecvBuff + 0;
    //通知解析imu任务
    BaseType_t flag = 0;
    xTaskNotifyFromISR(imuTaskHandle, 0x12345678, eSetValueWithOverwrite,
&flag);
    portYIELD_FROM_ISR(flag);
}

void IMU_RxCpltCallback()
{
    imuPackage = imuRecvBuff + 64;
    //通知解析imu任务
    BaseType_t flag = 0;
    xTaskNotifyFromISR(imuTaskHandle, 0x12345678, eSetValueWithOverwrite,
&flag);
    portYIELD_FROM_ISR(flag);
}
```

IMU解析函数：

```
int DecodeIMUPackage()
{
    if (imuPackage) {
        if (imuBuffOffset + 64 >= 1024) imuBuffOffset = 0;
        memcpy(imuBuff + imuBuffOffset, imuPackage, 64);
        imuBuffOffset += 64;
        imuPackage = 0;
    }
    //定位帧头
    int packageStart = 0;
    int flag = 0;
    for (; imuBuffOffset > 5 && packageStart < imuBuffOffset - 4;
++packageStart) {
        uint8_t crc8 = CRC8_Table(imuBuff + packageStart, 4);

        if (imuBuff[packageStart] == 0xFC && crc8 == imuBuff[packageStart
+ 4]) {
            flag = 1;
            break;
        }
    }
    //将帧头移动到起始位置
    if (flag) {
        memcpy(imuBuff, imuBuff + packageStart, imuBuffOffset -
packageStart);
        imuBuffOffset -= packageStart;
    }

    int flag2 = 0;
```

```
    //判断帧头
    uint8_t crc8 = CRC8_Table(imuBuff, 4);
    if (imuBuff[0] == 0xFC && crc8 == imuBuff[4]) { //是帧头
        int cmd = imuBuff[1]; //指令类别
        int dataLen = imuBuff[2]; //数据长度

        //判断是否接收完整个数据
        if (imuBuffOffset >= 5 + 2 + dataLen + 1) { //5->帧头 2->crc16 1->
帧尾标记
            uint16_t crc16FromBuff = (imuBuff[5] << 8) | imuBuff[6];
            uint16_t crc16 = CRC16_Table(imuBuff + 7, dataLen);

            if (crc16 == crc16FromBuff) {
                HandleIMUPackage(cmd, imuBuff + 7, dataLen);
                flag2 = 1;
            }

            int frameLen = 5 + 2 + dataLen + 1;
            memcpy(imuBuff, imuBuff + frameLen, imuBuffOffset - frameLen);
            imuBuffOffset -= frameLen;
        }
    }
    return flag2;
}
```

将IMU传感器数据发送给上位机

```
char message[128] = {0};
snprintf(message, 128, "[imu]heading:%.4f pitch:%.4f roll:%.4f\r\n",
robotIMU.heading, robotIMU.pitch, robotIMU.roll);
HAL_UART_Transmit(&huart1, message, strlen(message), 100);
```

上位机接收到的数据示例：

```
[imu]heading:5.6123 pitch:0.0045 roll:0.0247
[imu]heading:5.6419 pitch:0.0047 roll:0.0246
[imu]heading:5.6761 pitch:0.0048 roll:0.0245
[imu]heading:5.7051 pitch:0.0049 roll:0.0242
[imu]heading:5.7284 pitch:0.0051 roll:0.0239
```