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。

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。

pwm结构体：

```
volatile struct WheelPWM
{
    double kp; //p比例系统
    double ki; //i比例系统
    double kd; //d比例系数
    double speed; //电机速度
    double target; //目标速度
    double error; //误差
    double lastError; //上一次误差
    double lastError2; //上上一次误差
    double minPWM; //最小输出pwm值
    double maxPWM; //最大输出pwm值
    double pwm; //最终输出pwm值
    double dis; //路程
};
```

增量式PID算法：

```
void Tick(volatile struct WheelPWM* wheelPwm)
{
    wheelPwm->lastError2 = wheelPwm->lastError;
    wheelPwm->lastError = wheelPwm->error;
    wheelPwm->error = wheelPwm->target - wheelPwm->speed;

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

```
>minPWM;
}
```

```
void 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
    Tick(&leftPWM);
    Tick(&rightPWM);

    //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);
    }
}
```

启动增量式速度pid控制算法：

```
    //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);
```

```
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) {
      PID_Tick();
  }

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