(定位系统)UWB 测距(多对多)基本原理及测距代码实现

一、(定位系统)双边测距原理介绍

本工程依旧采用了文档《Decawave 官方双边测距原理介绍.docx》中介绍的双边双向测距(Double-sided Two-WAY Ranging)方法，[此处我们不在赘述其原理，如不理解可参考上述文档]。考虑到(定位系统)标签不再仅与单个基站进行测距，所以我们需要对双边双向测距的流程进行了优化，让单个标签与 n 个基站测距所用的时间尽可能的缩短，以保证可以容纳更多的标签数量。

1.1 单个标签测距流程介绍

(定位系统)单个标签测距具体流程如下图所示：仅需使用 1+n+1 条空中数据包即可完成单个标签与 n 个基站的测距。

第一个 1 代表：1 条 Poll 数据包(标签发送)

第一个 n 代表：n 条 Resp 数据包(基站发送)

第二个 1 代表：1 条 Final 数据包(标签发送)

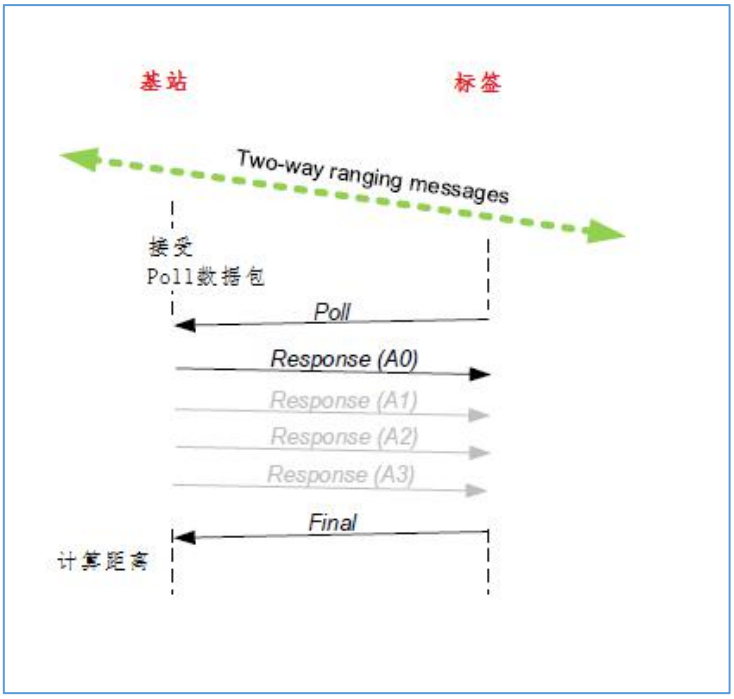

图 1 单个标签测距流程图

1.2 多个标签测距流程介绍

(定位系统)多个标签测距依旧延续了 1.1 节提到的标签测距流程，但为了让每个标签相互不进行干扰，采用了时分多址技术(TDMA)，即在一段固定的时间内给每个标签安排唯一的时间空隙(此处我们是按照标签的 ID 对标签进行时间分配，分配者为基站 A0)。

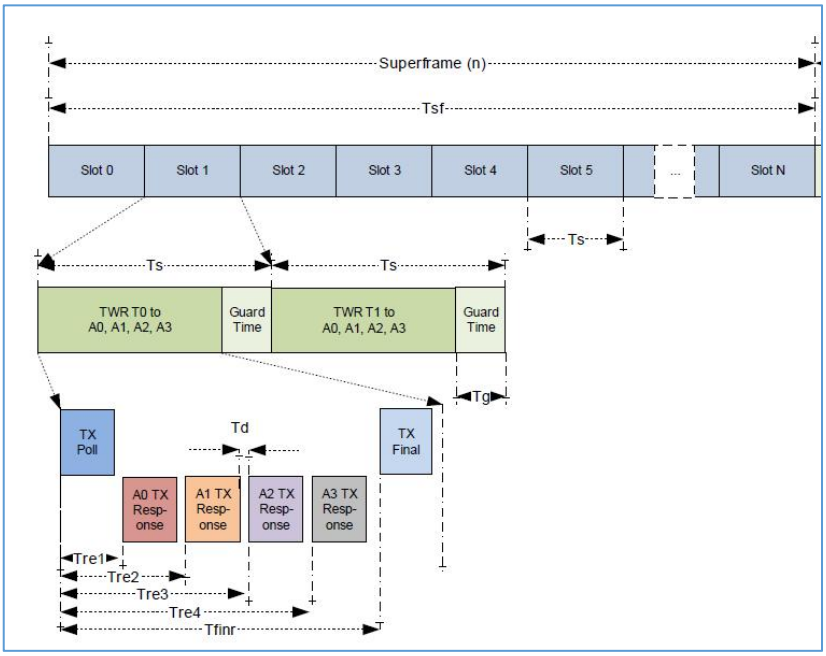

图 2 多个标签时分多址分配图

1.3 标签距离滤波介绍

(定位系统)工程中还对 UWB 的距离数据进行了滤波处理，滤波方式采用了卡尔曼滤波，网上关于卡尔曼滤波的资料很多，这里笔者将不展开讲。

1.4 标签坐标解算介绍

(定位系统)工程中还对 UWB 标签进行了坐标解算，其采用三边定理算法，因为算法较为简单所以放在 STM32 中也无碍，如读者想实现更高精度的定位，必定涉及到更多更复杂的算法，笔者强烈建议大家使用电脑或者额外增加处理器进行坐标解算。下面简单介绍工程中用到三边定位算法:

三边定位算法原理：已知 3 个基站的坐标 $A_0(x_1,y_1)$,$A_1(x_2,y_2)$,$A_2(x_3,y_3)$,且已知待测标签 $Tn(x_0,y_0)$到 3 个基站的距离值分别为 d1,d2,d3。

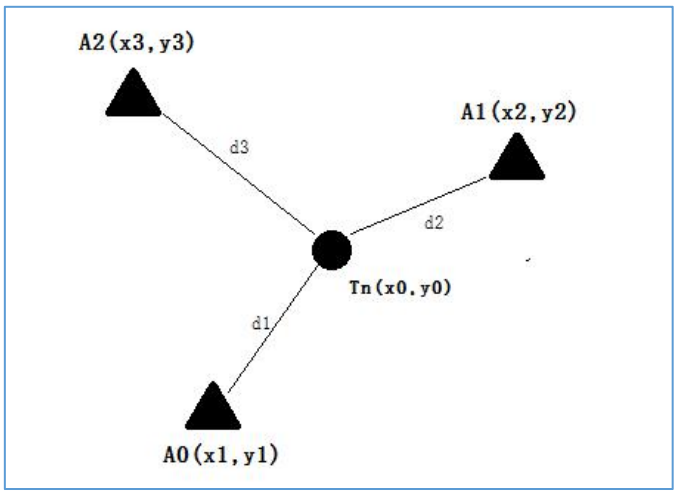

图 3 三边定位算法示意图

二维定位情况下，根据毕达哥拉斯定理(勾股定理)得到待测标签 Tn 及焦点的位置计算公式如下：

$$(x_1-x_0)^2 + (y_1-y_0)^2 = d_1^2$$
$$(x_2-x_0)^2 + (y_2-y_0)^2 = d_2^2$$
$$(x_3-x_0)^2 + (y_3-y_0)^2 = d_3^2$$

理想场景下：标签 Tn 到达基站 A_0,A_1,A_2 的距离值都是精准，及这时标签 Tn 有且仅有一个正确的解，如图红点所示。

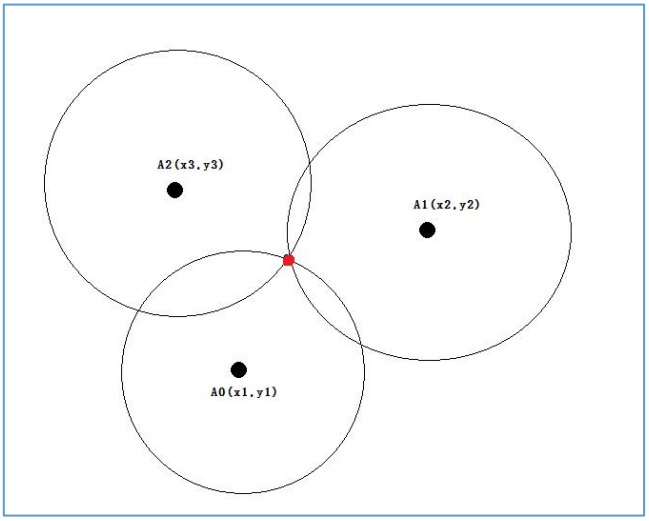

图 4 理想场景示意图

真实场景下：真实的测距是存在误差的，产生误差的原因包括测距本身的误差和实际场景中遮挡或者金属等原因影响到电磁波的穿透性和传输效果从而产生的误差，但该误差为正向误差，及测量值比实际偏大，则形成了如下图所示的效果，解算的 Tn 坐标不再是一点，而是一个区域，因此我们需要在如下红色区域查找一个最优解。

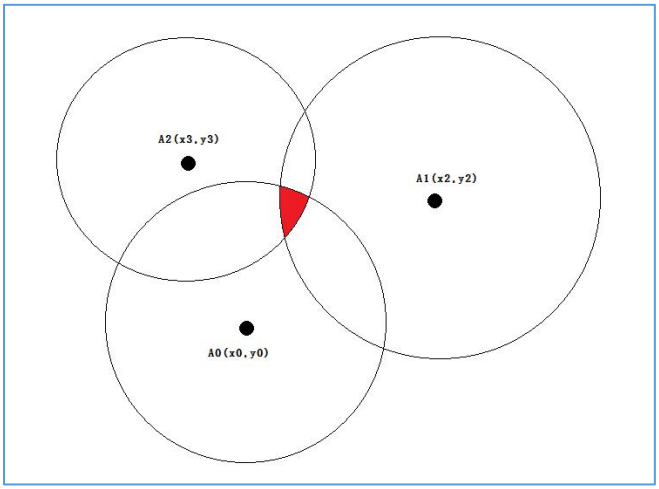

图 5 真实场景示意图

针对上图真实场景存在的问题，学术上有众多算法来优化该现象，例如最小二乘法、三角形质心法、加权三边算法等。

1.5 单个标签测距通信协议(对 1.1 的补充说明)

Poll 数据包:21 字节(数据段)
Resp 数据包:27 字节(数据段)
Final 数据包:53 字节(数据段)

Poll数据包

1字节	1字节	1字节	1字节	1字节	16字节
功能码	测距Err Num	透传地址 LSB	透传地址 MSB	透传数据Len	透传数据
0x01	-	-	-	-	-

Resp数据包

1字节	2字节	1字节	1字节	1字节	1字节	1字节	16字节
功能码	时间戳 修正	测距值 （上一次距离值）	测距Err Num	透传地址 LSB	透传地址 MSB	透传数据Len	透传数据
0x70	-	-	-	-	-	-	-

Final数据包

1字节	1字节	5字节	5字节	5字节	5字节	5字节	5字节	5字节	5字节	5字节	5字节	1字节
功能码	测距Err Num	Poll 发送时间戳	A0 Resp 接受时间戳	A1 Resp 接受时间戳	A2 Resp 接受时间戳	A3 Resp 接受时间戳	A4 Resp 接受时间戳	A5 Resp 接受时间戳	A6 Resp 接受时间戳	A7 Resp 接受时间戳	Final 发送时间戳	Resp 有效位
0x02	-	-	-	-	-	-	-	-	-	-	-	-

图 6 空中数据包格式

二、(定位系统)测距代码实现

[UWB-S1 定位系统源码]与[UWB-S1 测距源码(官改)]差别对比

	UWB-S1 定位系统源码(STM32)	UWB-S1 测距源码(官改)
距离滤波	卡尔曼滤波	无
坐标解算	三边定理	无
信号诊断	有	无
透传指令	有	无
中断方式	有	无
标签功耗	低	高

2.1 主函数

```
int main(void)
{
    init();        //外设初始化

    twr_task();//标签/基站 twr 任务

    for(;;)
    {
    }
}

void twr_task(void)
{
    twr_init();     //twr 初始化(重点)

    twr_run();     //twr 工作(重点)
}
```

2.2 基站/标签初始化 twr_init()

初始化 DW1000 设备

```
int twr_init(void)
{
    uint16_t s1switch = 0;

    //禁用 DW1000 IRQ 中断(配置阶段,禁用)
    port_DisableEXT_IRQ();

    s1switch = osal_Transfer_Byte((uint16)(sys_para.device_switch & 0xFFFF));

     //检查是否配置 DW1000
      while(s1switch == 0x0000)
      {
            Sleep(500);
            App_Module_Sys_Work_Mode_Event(Sys_Operate_Mode_CFG_ING);
            App_Module_Sys_IO_Led_Mode_Set(Sys_Operate_Mode_CFG_ING);
      }

     //初始化 DW1000
      while(inittestapplication(s1switch) == (uint32)-1)//重点函数
      {
            _dbg_printf("初始化失败\n");
            reset_DW1000();//硬件复位 dw1000
            Sleep(500);
      }

    //使能 DW1000 IRQ 中断
    port_EnableEXT_IRQ();

    //显示配置的信息
    App_Module_UWB_Mode_Display((uint16_t)sys_para.device_switch);
}
```

2.2.1 Inittestapplication 函数

```
uint32 inittestapplication(uint16_t s1switch)
{
    uint32 devID;
    int result;
```

```
//降低 SPI 速率值低于 3MHz
port_set_dw1000_slowrate();

//读取设备 Dev ID 值(SPI  低速率)
devID = instance_readdeviceid();
if(DWT_DEVICE_ID != devID)
{
    //唤醒 dw1000
    port_wakeup_dw1000();

    //再次读取设备 Dev ID 值(SPI  低速率)
    devID = instance_readdeviceid() ;
    if(DWT_DEVICE_ID != devID)
        return(-1) ;

    //软复位 dw1000
    dwt_softreset();
}

//硬件复位 dw1000
reset_DW1000();

//判断配置为基站/标签
if((s1switch & SWS1_ANC_MODE) == 0)
    instance_mode = TAG;
else
    instance_mode = ANCHOR;

//根据设备角色,初始化 dw1000
result = instance_init(instance_mode) ;
if (0 > result) return(-1) ;

//提高 SPI 速率值 20MHz
port_set_dw1000_fastrate();
//读取设备 Dev ID 值(SPI  高速率)
devID = instance_readdeviceid();

if (DWT_DEVICE_ID != devID)
{
    return(-1) ;
}

//设置 16bit 地址
addressconfigure(s1switch, instance_mode) ;
```

```
#ifdef HW_8Anc
    if((instance_mode == ANCHOR) && (instance_anchaddr > (MAX_ANCHOR_LIST_SIZE - 1)))
#else
    if((instance_mode == ANCHOR) && (instance_anchaddr > 0x3))
#endif
    {
        _dbg_printf("非法地址");
        return (-1);
        //非法配置
    }
    else
    {

        dr_mode = decarangingmode(s1switch);

        chan = chConfig[dr_mode].channelNumber ;

        //根据配置，选择 sfconfig(时间槽配置)和 chConfig(信道配置)
        instance_config(&chConfig[dr_mode], &sfConfig[dr_mode]) ;
    }
    return devID;
}
```

在 Inittestapplication 函数里，主要干了 3 件事。

1、验证 SPI 通讯是否正常

2、初始化 dw1000 设备 instance_init()

3、配置 dw1000 信道 instance_config()

在 instance_init 中采用中断方式，相比[UWB-S1 测距源码(官改)]的轮训方式效率更高

```
dwt_setinterrupt(xxxxxxxxxxxx);

if(inst_mode == ANCHOR)
{
     //基站回调函数设置
    dwt_setcallbacks(tx_conf_cb, rx_ok_cb_anch, rx_to_cb_anch, rx_err_cb_anch);
}
else
{
     //标签回调函数设置
    dwt_setcallbacks(tx_conf_cb, rx_ok_cb_tag, rx_to_cb_tag, rx_err_cb_tag);
}
```

2.3 基站/标签运行 twr_run()

如读者对其通信流程不敢兴趣，只对输出的距离值数据，定位数据及输出格式感兴趣，可以关注下述**红色字体加粗** 4 个函数。(这里将展开下面 4 个函数)

1、tag_distance_cpy(); //标签距离赋值

2、tag_distance_fil(); //标签距离滤波(卡尔曼)

3、tag_position_cal(); //标签坐标计算

4、tag_output(); //数据输出打印

如读者对通讯流程感兴趣研究通信原理(上述第一章节介绍)且想修改底层逻辑，关注**蓝色字体加粗** 2 个函数

1、tag_run(); //标签运行程序(在 instance_tag.c 里头)

2、anch_run(); //基站运行程序(在 instance_anch.c 里头)

(这里不展开描述，工程中都已经中文注释，请读者自行查看)。

```
void twr_run(void)
{
    Int    rx = 0;
    while(1)
    {
      instance_data_t* inst = instance_get_local_structure_ptr(0);

      int monitor_local = inst->monitor ;
      int txdiff = (portGetTickCnt() - inst->timeofTx);

       instance_mode = instance_get_role();

       if(instance_mode == TAG)
       {
            tag_run();//标签运行程序
       }
       else
       {
            anch_run();//基站运行程序
       }

         //if delayed TX scheduled but did not happen after expected time then it has failed... (has to be < slot period)
         //if anchor just go into RX and wait for next message from tags/anchors
         //if tag handle as a timeout
```

```
            if((monitor_local == 1) && ( txdiff > inst->slotDuration_ms))
            {
                int an = 0;
                uint32 tdly ;
                uint32 reg1, reg2;

                reg1 = dwt_read32bitoffsetreg(0x0f, 0x1);
                reg2 = dwt_read32bitoffsetreg(0x019, 0x1);
                tdly = dwt_read32bitoffsetreg(0x0a, 0x1);
//              an = sprintf((char*)&usbVCOMout[0], "T%08x %08x time %08x %08x", (unsigned int) reg2, (unsigned int) reg1,
//                              (unsigned int) dwt_read32bitoffsetreg(0x06, 0x1), (unsigned int) tdly);

                 inst->wait4ack = 0;

                if(instance_mode == TAG)
                {
                        tag_process_rx_timeout(inst);
                }
                else //if(instance_mode == ANCHOR)
                {
                        dwt_forcetrxoff();    //this will clear all events
                        //dwt_rxreset();
                        //enable the RX
                        inst->testAppState = TA_RXE_WAIT ;
                }
                inst->monitor = 0;
            }

            rx = instance_newrange();

            //检测到有新的距离数据产生,则打印
            if(rx == TOF_REPORT_T2A)
            {
                 tag_distance_cpy(); //标签距离赋值
                 tag_distance_fil();//标签距离滤波
                 tag_position_cal(); //标签坐标计算
                 tag_output();//数据输出打印
            }

             App_Module_Sys_Work_Mode_Event(Sys_Operate_Mode_WORK_DONE);
             App_Module_Sys_IO_Led_Mode_Set(Sys_Operate_Mode_WORK_DONE);
        }
}
```

2.3.1 tag_distance_cpy()函数

```
void tag_distance_cpy(void)
{
	instance_otpt.anc_addr = instance_newrangeancadd() & 0xf;
	instance_otpt.tag_addr = instance_newrangetagadd() & 0xf;
	instance_otpt.newRangeTime = instance_newrangetim() & 0xffffffff;
	instance_otpt.lnum = instance_get_lcount() & 0xFFFF;
	if(instance_mode == TAG)
		instance_otpt.rnum = instance_get_rnum();
	else
		instance_otpt.rnum = instance_get_rnuma(instance_otpt.tag_addr);
	instance_otpt.trxa =    instance_get_txantdly();
	instance_otpt.mask = instance_validranges();

	for(int i = 0; i < MAX_ANCHOR_LIST_SIZE; i++)
	{
		instance_otpt.dist[i] = instance_get_idist_mm(i);
	}

	instance_otpt.rx_diag = instance_newrangerx_diag();

	instance_newtrasport_cmd(instance_otpt.resv_trasport);

	instance_cleardisttableall();
}
```

2.3.2 tag_distance_fil()函数

```
  void tag_distance_fil(void)
{
	if(sys_para.dist.is_kalman_filter)
	{
		//instance_otpt.tag_addr:标签地址
		//instance_otpt.dist:原始距离数据
		//instance_otpt.filter_dist:滤波距离数据
		//instance_otpt.mask:标签距离有效位
		//sys_para.dist.kalman_Q:卡尔曼系数 Q
		//sys_para.dist.kalman_R:卡尔曼系数 R
		processTagRangeReports_KalmanFilter(instance_otpt.tag_addr,
			instance_otpt.dist,
```

```
                instance_otpt.filter_dist,
                instance_otpt.mask,
                sys_para.dist.kalman_Q,
                sys_para.dist.kalman_R);
    }
}
```

2.3.3 tag_position_cal()函数

```
void tag_position_cal(void)
{
    if(sys_para.loc.is_loc)
    {
        int ranges[MAX_NUM_ANCS];
        int loc_num = 0;
        int mask = instance_otpt.mask;
        vec3d_t tmp_report, tmp_report_sum;
        memset(&tmp_report, 0, sizeof(vec3d_t));
        memset(&tmp_report_sum, 0, sizeof(vec3d_t));

        //如果开启滤波，则使用滤波距离进行三边测距
        if(sys_para.dist.is_kalman_filter)
        {
            memcpy(ranges, instance_otpt.filter_dist, MAX_NUM_ANCS);
        }
        else
        {
            memcpy(ranges, instance_otpt.dist, MAX_NUM_ANCS);
        }

        switch(sys_para.loc.loc_dimen)
        {
            //二维定位
            case Sys_Dimension_Two:
                {
                  //进行定位,此处省略
                }
                break;

            //三维定位
            case Sys_Dimension_Three:
                {
                  //进行定位,此处省略
                }
                break;
```

```
                //零维/一维定位
                case Sys_Dimension_Zero:
                case Sys_Dimension_One:
                case Sys_Dimension_Invalid:        //无效定位
                default:
                    break;
            }
        }
}
```

2.3.4 tag_output()函数

```
void tag_output(void)
{
    uint8_t buf[512];
    memset(buf, 0, sizeof(buf));

    switch(sys_para.cmd_mode)
    {
        //字符串形式输出原始距离
        case Sys_Commond_Mode_Char_RawDist:
            {
                //输出原始数据 json 格式,此处省略
            }
            break;

        //字符串形式输出卡尔曼滤波距离
        case Sys_Commond_Mode_Char_KalmanDist:
            {
                //输出滤波数据 json 格式,此处省略
            }
            break;

        //字符串形式输出定位数据
        case Sys_Commond_Mode_Char_Pos:
            {
                //输出定位数据 json 格式,此处省略
            }
            break;

        //字符串形式输出信号强度
        case Sys_Commond_Mode_Char_Sign:
            {
                //输出信号强度 json 格式,此处省略
```

```
            }
            break;

        //十六进制输出
        case Sys_Commond_Mode_Hex:
            {
                //输出十六进制格式,此处省略
            }
            break;

        default:
            break;

    }
    memset(&instance_otpt, 0, sizeof(instance_otpt));//清除结构体

}
```