题目要求：

找出ITM软件中的整像素运动估计模块代码，并对搜索过程做简要分析

一、确定整像素文件搜索代码位置

1、打开项目文件进行查看，由于像素运动估计的英文为pixel motion estimation，故运动估计的简写可能为me，打开项目工程文件，浏览所有的头文件和源文件，可以猜测整像素运动估计的实现过程在fast_me.h和fast_me.c文件中(至于fast还需要进行下一步的观察)。

2、打开头文件fast_me.h进行查看，可以清晰的看到如下注释：模块功能描述：对于快速整像素运动估计和分数像素运动估计的宏定义和全局变量。故实现整像素运动估计可能在此模块实现。

```
/*
*************************************************************************************
*****
* File name:

* Function: Macro definitions and global variables for fast integer pel motion
            estimation and fractional pel motio estimation
*
*************************************************************************************
*****
*/
```

3、继续查看头文件fast_me.h,有FastIntegerPelBlockMotionSearch 和 MHMC_FastIntegerPelBlockMotionSearch两个搜索函数，故运动搜索在此实现

```
int                                     //  ==> minimum motion cost after search
FastIntegerPelBlockMotionSearch  (pel_t**   orig_pic,     // <--  not used
                                  int       ref,          // <--  reference frame (0... or -1 (backward))
                                  int       pic_pix_x,    // <--  absolute x-coordinate of regarded AxB block
                                  int       pic_pix_y,    // <--  absolute y-coordinate of regarded AxB block
                                  int       blocktype,    // <--  block type (1-16x16 ... 7-4x4)
                                  int       pred_mv_x,    // <--  motion vector predictor (x) in sub-pel units
                                  int       pred_mv_y,    // <--  motion vector predictor (y) in sub-pel units
                                  int*      mv_x,         //  --> motion vector (x) - in pel units
                                  int*      mv_y,         //  --> motion vector (y) - in pel units
                                  int       search_range, // <--  1-d search range in pel units
                                  int       min_mcost,    // <--  minimum motion cost (cost for center or huge value)
                                  double    lambda) ;     // <--  lagrangian parameter for determining motion cost

int                                     //  ==> minimum motion cost after search
    MHMC_FastIntegerPelBlockMotionSearch  (pel_t**   orig_pic,     // <--  not used
    int       ref,          // <--  reference frame (0... or -1 (backward))
    int       pic_pix_x,    // <--  absolute x-coordinate of regarded AxB block
    int       pic_pix_y,    // <--  absolute y-coordinate of regarded AxB block
    int       blocktype,    // <--  block type (1-16x16 ... 7-4x4)
    int       pred_mv_x,    // <--  motion vector predictor (x) in sub-pel units
    int       pred_mv_y,    // <--  motion vector predictor (y) in sub-pel units
    int*      mv_x,         //  --> motion vector (x) - in pel units
    int*      mv_y,         //  --> motion vector (y) - in pel units
    int       fw_pred_mv_x,    // <--  forward motion vector predictor (x) in sub-pel units
    int       fw_pred_mv_y,    // <--  forward motion vector predictor (y) in sub-pel units
    int*      fw_mv_x,         //  --> forward motion vector (x) - in pel units
    int*      fw_mv_y,         //  --> forward motion vector (y) - in pel units
    int       search_range, // <--  1-d search range in pel units
    int       min_mcost,    // <--  minimum motion cost (cost for center or huge value)
    double    lambda) ;     // <--  lagrangian parameter for determining motion cost
```

4、确定正像素运动估计实现在fast_me中，打卡fast_me.c文件，通过fast_me.c文件的注释部分:可以看出在该文件中实现了整像素运动估计和分数像素运动估计

```
/*
************************************************************************************
*****
* File name:
* Function:
Fast integer pel motion estimation and fractional pel motion estimation
algorithms are described in this file.
1. get_mem_FME() and free_mem_FME() are functions for allocation and release
of memories about motion estimation
2. FME_BlockMotionSearch() is the function for fast integer pel motion
estimation and fractional pel motion estimation
3. DefineThreshold() defined thresholds for early termination
*
************************************************************************************
*****
*/
```

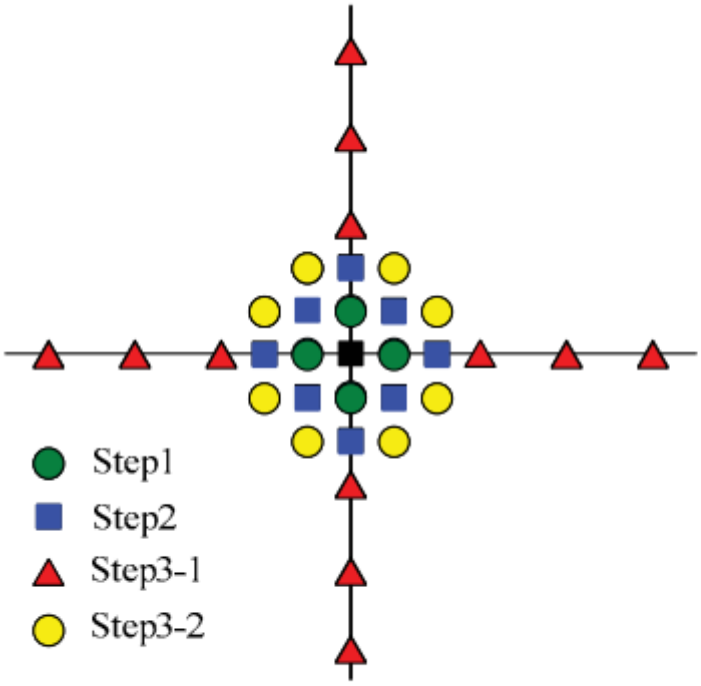
二、搜索过程的简要分析

通过对函数名FastIntegerPelBlockMotionSearch的注释可知，搜索的实现过程采用了UMHexagonS 搜索算法。

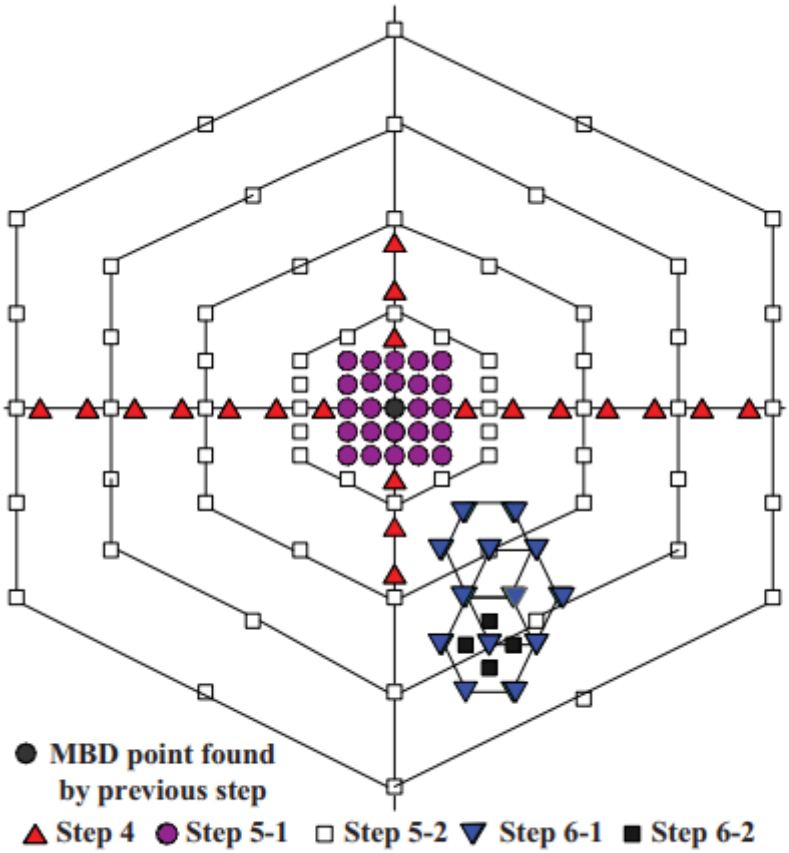
(一)、UMHexagonS 算法

UMHexagonS 算法是一种开始进行运动估计的算法，有较大的搜索范围中，并且能够避免陷入局部最优解的问题，能够精简搜索点的数量，提高搜索的效率，被用于各种不同纹理特征和运动剧烈程度的图像帧序列。

搜索的过程图形：



UMHexagonS search pattern(Step1 to 3)



UMHexagonS search pattern(Step4 to 6)

算法步骤概览：

步骤一：起始点搜索预测(共有5种模式求取预测的MV)

1、中值预测：利用空间相关性，通过已求出的、当前帧的左、上、右上邻块的运动矢量求取中间值得到该块的运动矢量初始估计
2、原点预测：预测初始运动矢量为0，即当前块的位置
3、上层预测：将已求出当前块位置的上一级模式的运动矢量作为当前模式的运动矢量预测值（算法采用从模式1<16*16>到模式7<4*4>的分级搜索策略）
4、对应块预测：选取前一帧对应位置的运动矢量作为当前块运动矢量的预测值
5、相邻参考帧预测：利用时间相关性，将之前的参考帧运动矢量按时间进行缩放

步骤二：不甚满意区块搜索（对应代码里面first_step)

1、以上一步预测的起始搜索点为中心，用非对称十字型搜索(Unsymmetrical-cross search)方法进行寻找，获取当前最佳点，判断该点属于满意或者很满意区，以决定跳转到步骤三、步骤四或继续搜索
非对称十字型搜索：先后对x轴和y轴进行搜索，y轴的搜索范围是x轴的一半
2、以当前最佳点为中心，在[-2,2]的方形区域5*5中进行全搜索，获得当前最佳点，判断是否属于满意或很满意区，跳到相应的步骤三或步骤四，或继续搜索；
3、用大六边形模板(16点)进行搜索，直至搜索到能符合相应阈值而进入到步骤三或者步骤四的点为止，否则也结束步骤二的搜索二进入到步骤三。

步骤三：满意区的块搜索（对应代码里面sec_step)

以当前最佳点为中心进行六边形反复搜索，直到预测出的最佳点为六边形的中心

步骤四：很满意区块搜索（对应代码里面third_step）

> 以当前最佳点为中心进行小菱形搜索，直到预测出的最佳点为菱形中心

(二)、整像素运动估计代码FastIntegerPelBlockMotionSearch 函数分析

部分变量定义和声明：

```
static int Diamond_x[4] = { -1, 0, 1, 0 };//菱形插值
static int Diamond_y[4] = { 0, 1, 0, -1 };
static int Hexagon_x[6] = { 2, 1, -1, -2, -1, 1 };//六边形插值
static int Hexagon_y[6] = { 0, -2, -2, 0, 2, 2 };
//大六角形插值
static int Big_Hexagon_x[16] = { 0, -2, -4, -4, -4, -4, -4, -2, 0, 2, 4, 4, 4,
4, 4, 2 };
static int Big_Hexagon_y[16] = { 4, 3, 2, 1, 0, -1, -2, -3, -4, -3, -2, -1, 0,
1, 2, 3 };
```

步骤一：起始点搜索预测：

在初始化所需变量后首先进行起始搜索点的预测，在代码中共采用中值预测、原点预测、上层块预测、对应块预测、相邻参考帧预测五种预测方法。

1、中值预测

```
    ////////////search around the predictor and (0,0)
    //check the center median predictor
    cand_x = center_x;
    cand_y = center_y;
    mcost = MV_COST(lambda_factor, mvshift, cand_x, cand_y, pred_x, pred_y);
    if (ref != -1) {
        mcost += REF_COST(lambda_factor, ref);
    }

    mcost = PartCalMad(ref_pic, orig_pic, get_ref_line, blocksize_y,
blocksize_x, blocksize_x4, mcost, min_mcost, cand_x, cand_y);
    McostState[search_range][search_range] = mcost;
    if (mcost < min_mcost)
    {
        min_mcost = mcost;
        best_x = cand_x;
        best_y = cand_y;
    }

    iXMinNow = best_x;
    iYMinNow = best_y;
    for (m = 0; m < 4; m++)
    {
        cand_x = iXMinNow + Diamond_x[m];
        cand_y = iYMinNow + Diamond_y[m];
        SEARCH_ONE_PIXEL
    }
```

2、原点预测

```
if (center_x != pic_pix_x || center_y != pic_pix_y)
{
    cand_x = pic_pix_x;
    cand_y = pic_pix_y;
    SEARCH_ONE_PIXEL

        iXMinNow = best_x;
    iYMinNow = best_y;
    for (m = 0; m < 4; m++)
    {
        cand_x = iXMinNow + Diamond_x[m];
        cand_y = iYMinNow + Diamond_y[m];
        SEARCH_ONE_PIXEL
    }
}
```

3、上层块预测：

```
if (blocktype>1)
{
    cand_x = pic_pix_x + (pred_MV_uplayer[0] / 4);
    cand_y = pic_pix_y + (pred_MV_uplayer[1] / 4);
    SEARCH_ONE_PIXEL
    if ((min_mcost - pred_SAD_uplayer)<pred_SAD_uplayer*betaThird)
        goto third_step;
    else if ((min_mcost - pred_SAD_uplayer)<pred_SAD_uplayer*betaSec)
        goto sec_step;
}
```

4、对应块预测

```
//coordinate position prediction
if ((img->number > 1 + ref && ref != -1) || (ref == -1 && Bframe_ctr > 1))
{
    cand_x = pic_pix_x + pred_MV_time[0] / 4;
    cand_y = pic_pix_y + pred_MV_time[1] / 4;
    SEARCH_ONE_PIXEL
}
```

5、相邻参考帧预测

```
//prediciton using mV of last ref moiton vector
if ((ref > 0) || (img->type == B_IMG && ref == 0))
{
cand_x = pic_pix_x + pred_MV_ref[0] / 4;
cand_y = pic_pix_y + pred_MV_ref[1] / 4;
SEARCH_ONE_PIXEL
}
//strengthen local search
iXMinNow = best_x;
iYMinNow = best_y;
for (m = 0; m < 4; m++)
```

```
    {
    cand_x = iXMinNow + Diamond_x[m];
    cand_y = iYMinNow + Diamond_y[m];
    SEARCH_ONE_PIXEL
    }
```

步骤二：不甚满意区块搜索

代码中对应`first_step`(*Unsymmetrical-cross search*)，首先进行非对称十字型搜索，再进行5*5区域全搜索。

1、非对称十字型搜索

```
// 非对称十字型搜索
st_step: //Unsymmetrical-cross search
    iXMinNow = best_x;
    iYMinNow = best_y;

// 水平方向上的搜索
    for (i = 1; i <= search_range / 2; i++)
    {
        search_step = 2 * i - 1;
        cand_x = iXMinNow + search_step;
        cand_y = iYMinNow;
        SEARCH_ONE_PIXEL
            cand_x = iXMinNow - search_step;
        cand_y = iYMinNow;
        SEARCH_ONE_PIXEL
    }
// 垂直方向上的搜索，垂直方向上是水平方向上的一般
    for (i = 1; i <= search_range / 4; i++)
    {
        search_step = 2 * i - 1;
        cand_x = iXMinNow;
        cand_y = iYMinNow + search_step;
        SEARCH_ONE_PIXEL
            cand_x = iXMinNow;
        cand_y = iYMinNow - search_step;
        SEARCH_ONE_PIXEL
    }
    EARLY_TERMINATION

    iXMinNow = best_x;
    iYMinNow = best_y;
```

2、5*5区域搜索

```
// Uneven Multi-Hexagon-grid Search
    // 5*5区域搜索
    for (pos = 1; pos < 25; pos++)
    {
        cand_x = iXMinNow + spiral_search_x[pos];
        cand_y = iYMinNow + spiral_search_y[pos];
        SEARCH_ONE_PIXEL
    }
    EARLY_TERMINATION
```

3、大六边形进行搜索

```
    for (i = 1; i <= search_range / 4; i++)
    {
        iAbort = 0;
        for (m = 0; m < 16; m++)
        {
            cand_x = iXMinNow + Big_Hexagon_x[m] * i;
            cand_y = iYMinNow + Big_Hexagon_y[m] * i;
            SEARCH_ONE_PIXEL1(1)
        }
        if (iAbort)
        {

            EARLY_TERMINATION
        }
    }
```

同样在此之后进行满意度的判断，决定是否直接跳到最后阶段。

步骤三：满意区块搜索

代码中对应`second_step`(*Extended Hexagon-based Search*)

```
sec_step:  //Extended Hexagon-based Search
    iXMinNow = best_x;
    iYMinNow = best_y;
    for (i = 0; i < search_range; i++)
    {
        iAbort = 1;
        for (m = 0; m < 6; m++)
        {
            cand_x = iXMinNow + Hexagon_x[m];
            cand_y = iYMinNow + Hexagon_y[m];
            SEARCH_ONE_PIXEL1(0)
        }
        if (iAbort)
            break;
        iXMinNow = best_x;
        iYMinNow = best_y;
    }
```

步骤四：很满意区块搜索

代码中对应`third_step`(*the third step with a small search pattern*)

```
third_step: // the third step with a small search pattern
    iXMinNow = best_x;
    iYMinNow = best_y;
    for (i = 0; i < search_range; i++)
    {
        iSADLayer = 65536;
        iAbort = 1;
        for (m = 0; m < 4; m++)
        {
            cand_x = iXMinNow + Diamond_x[m];
            cand_y = iYMinNow + Diamond_y[m];
```

```
            SEARCH_ONE_PIXEL1(0)
        }
        if (iAbort)
            break;
        iXMinNow = best_x;
        iYMinNow = best_y;
    }

    *mv_x = best_x - pic_pix_x;
    *mv_y = best_y - pic_pix_y;
    return min_mcost;
```

MHMC_FastSubPelBlockMotionSearch

MHMC(Multi hypothesis Motion compensated) 多假设运动补偿，MHMC_FastSubPelBlockMotionSearch和FastSubPelBlockMotionSearch 相比，搜索的方法一致，仅仅在设置获取参考图像线的时候多了一个函数设定，从而在后序的操作中提升运动搜索的效率。

```
    //===== set function for getting reference picture lines =====
    if ((center_x > search_range) && (center_x < img->width -1-search_range-
blocksize_x) &&
        (center_y > search_range) && (center_y < height-1-search_range-
blocksize_y)   )
    {
        get_ref_line = FastLineX;
    }
    else
    {
        get_ref_line = UMVLineX;
    }
    // 增加的设定
    if ((pic4_pix_x + *fw_mv_x > 1) && (pic4_pix_x + *fw_mv_x < max_pos_x4 - 2)
&&
        (pic4_pix_y + *fw_mv_y > 1) && (pic4_pix_y + *fw_mv_y < max_pos_y4 - 2)
  )
    {
        Fw_PelY_14 = FastPelY_14;
    }
    else
    {
        Fw_PelY_14 = UMVPelY_14;
    }
```