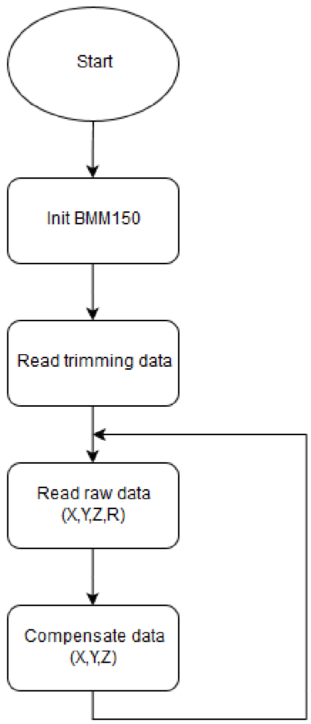

Calculation of magnetic field with temperature compensation

Trimming data required for sensor data compensation is stored in the non-volatile memory of BMM150, thus it is read out by the BMM150 API during initialization.

This is how you would manually read trimming data:

```
s8 dig_x1 = i2cRead(0x5D);
s8 dig_y1 = i2cRead(0x5E);

s8 dig_x2 = i2cRead(0x64);
s8 dig_y2 = i2cRead(0x65);

u8 dig_xy1 = i2cRead(0x71);
s8 dig_xy2 = i2cRead(0x70);

u16 dig_z1 = (i2cRead(0x6B) << 8) | i2cRead(0x6A);
s16 dig_z2 = (i2cRead(0x69) << 8) | i2cRead(0x68);
s16 dig_z3 = (i2cRead(0x6F) << 8) | i2cRead(0x6E);
s16 dig_z4 = (i2cRead(0x63) << 8) | i2cRead(0x62);

u16 dig_xyz1 = (i2cRead(0x6D) << 8) | i2cRead(0x6C);
```

You can then compute the compensated output by using the following functions:

```
s16 bmm150_compensate_X(s16 raw_mag_data_x, u16 raw_data_r)//you could also use s32 resolution
output by following the comments below
{
        s16 compensated_X = 0; //for 32 bit resolution, use s32 compensated_X = 0;

        if(raw_mag_data_x != (-4096))
        {
                if((raw_data_r != 0) && (dig_xyz1 != 0))
                {
                        compensated_X = ((s16)(((u16)((((s32)dig_xyz1)<< 14)/(raw_data_r !=
0 ? raw_data_r : dig_xyz1))) - ((u16)0x4000)));
                }
                else
                {
                        compensated_X = (-32768);//for 32 bit resolution, return ((s32)(-
2147483647-1)) instead
                        return compensated_X;
                }
                compensated_X = ((s16)((((s32)raw_mag_data_x) *((((((((s32)dig_xy2)
*((((s32)compensated_X) *((s32)compensated_X)) >>7)) +(((s32)compensated_X)
*((s32)(((s16)dig_xy1)<< 7))))>> 9) +((s32)0x100000)) *((s32)(((s16)dig_x2) +((s16)0xA0)))) >>
12))>> 13)) +(((s16)dig_x1)<< 3);
        }
        else
        {
                compensated_X = (-32768);//for 32 bit resolution, return ((s32)(-2147483647-
1)) instead
        }
        return compensated_X;
}
```

```
s16 bmm150_compensate_Y(s16 raw_mag_data_y, u16 raw_data_r)//you could also use s32 resolution
output by following the comments below
{
	s16 compensated_Y = 0; //for 32 bit resolution, use s32 compensated_Y = 0;

	if(raw_mag_data_x != (-4096))
	{
		if((raw_data_r != 0) && (dig_xyz1 != 0))
		{
			compensated_Y = ((s16)(((u16)((((s32)dig_xyz1)<< 14)/(raw_data_r !=
0 ? raw_data_r : dig_xyz1))) -((u16)0x4000)));
		}
		else
		{
			compensated_Y = (-32768);//for 32 bit resolution, return ((s32)(-
2147483647-1)) instead
			return compensated_Y;
		}
		compensated_Y = ((s16)((((s32)raw_mag_data_y) *((((((((s32)dig_xy2)
*((((s32)compensated_Y) *((s32)compensated_Y)) >>7)) +(((s32)compensated_Y)
*((s32)(((s16)dig_xy1)<< 7))))>> 9) +((s32)0x100000)) *((s32)(((s16)dig_y2) +((s16)0xA0)))) >>
12))>> 13)) +(((s16)dig_y1)<< 3);
	}
	else
	{
		compensated_Y = (-32768);//for 32 bit resolution, return ((s32)(-2147483647-
1)) instead
	}
	return compensated_Y;
}


s16 bmm150_compensate_Z(s16 raw_mag_data_z, u16 raw_data_r)//you could also use s32 resolution
output by following the comments below
{
	s32 compensated_Y = 0;

	if ((raw_mag_data_z != (-16384))
	   ) {
		if ((dig_z2 != 0)
		&& (dig_z1 != 0)
		&& (raw_data_r != 0)
		&& (dig_xyz1 != 0)) {
			compensated_Y = (((((s32)(raw_mag_data_z - dig_z4))<< 15) -
((((s32)dig_z3) * ((s32)(((s16)raw_data_r) -((s16) dig_xyz1))))>> 2))/(dig_z2 +
((s16)(((((s32)dig_z1) * ((((s16)raw_data_r)<< 1)))+(1 << 15))>> 16))));
		} else {
			compensated_Y = (-32768);//for 32 bit resolution, return ((s32)(-
2147483647-1)) instead
			return compensated_Y;
		}
		/* FOR 16 BIT ONLY, IGNORE IF 32 bit resolution: saturate result to +/- 2
microTesla */
		if (compensated_Y > (32767)) {
			compensated_Y =  (32767);
		} else {
			if (compensated_Y < (-32767))
				compensated_Y = (-32767);
		}
	} else {
		compensated_Y = (-32768);//for 32 bit resolution, return ((s32)(-2147483647-
1)) instead
	}
	return (s16)compensated_Y;//for 32 bit resolution, return compensated_Y instead
}
```

Or if your MCU allows floating point computation:

```
float bmm150_compensate_X_float(s16 raw_mag_data_x, u16 raw_data_r)
{
	float compensated_X = 0;

	if (raw_mag_data_x != (-4096)
	   ) {
		if ((raw_data_r != 0)
		&& (dig_xyz1 != 0)) {
			compensated_X = ((((float)dig_xyz1)
			* 16384.0 / raw_data_r) - 16384.0);
		} else {
			compensated_X = 0.0f;
			return compensated_X;
		}
		compensated_X = (((raw_mag_data_x * ((((((float)dig_xy2) *
			(compensated_X*compensated_X /
			268435456.0) +
			compensated_X * ((float)dig_xy1)
			/ 16384.0)) + 256.0) *
			(((float)dig_x2) + 160.0)))
			/ 8192.0)
			+ (((float)dig_x1) *
			8.0)) / 16.0;
	} else {
		compensated_X = 0.0f;
	}
	return compensated_X;
}

float bmm150_compensate_Y_float(s16 raw_mag_data_y, u16 raw_data_r)
{
	float compensated_Y = 0;

	if (raw_mag_data_y != (-4096)
	   ) {
		if ((raw_data_r != 0)
		&& (dig_xyz1 != 0)) {
			compensated_Y = ((((float)dig_xyz1)
			* 16384.0
			/raw_data_r) - 16384.0);
		} else {
			compensated_Y = 0.0f;
			return compensated_Y;
		}
		compensated_Y = (((raw_mag_data_y * ((((((float)dig_xy2) *
			(compensated_Y*compensated_Y
			/ 268435456.0) +
			compensated_Y * ((float)dig_xy1)
			/ 16384.0)) +
			256.0) *
			(((float)dig_y2) + 160.0)))
			/ 8192.0) +
			(((float)dig_y1) * 8.0))
			/ 16.0;
	} else {
		compensated_Y = 0.0f;
	}
	return compensated_Y;
}
```

```c
float bmm150_compensate_Z_float (s16 raw_mag_data_z, u16 raw_data_r)
{
	float compensated_Z = 0;

	if (raw_mag_data_z != (-16384)) {
		if ((dig_z2 != 0)
		&& (dig_z1 != 0)
		&& (dig_xyz1 != 0)
		&& (raw_data_r != 0)) {
			compensated_Z = ((((((float)raw_mag_data_z)-
			((float)dig_z4)) * 131072.0)-
			(((float)dig_z3)*(((float)raw_data_r)
			-((float)dig_xyz1))))
			/((((float)dig_z2)+
			((float)dig_z1)*((float)raw_data_r) /
			32768.0) * 4.0)) / 16.0;
		}
	} else {
		compensated_Z = 0.0f;
	}
	return compensated_Z;
}
```