2012 4th International Conference on Signal Processing Systems (ICSPS 2012)
IPCSIT vol. 58 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V58.21

An Optimization for Distance Estimation in Real-Time with DM6437

He Tian-xiang[+], FU Xiao-ning and Hou Guo-qiang

School of Electromechanical Engineering, Xidian University
Xi'an, China, 710071

Abstract. The passive ranging system on monocular has earned its bright application prospect in modern opto-electronic countermeasures and traffic monitoring. This technology is characterized by using the imaging direction and the distance-related features in the target image to estimate distance by solving a proper equation [1]. The distance-related features could be obtained by making use of complex image processing algorithms [2]. Limited by the complexity and time-consuming of the algorithm, to realize such an algorithm in real-time is much difficult. To solve this problem, in addition to using high-performance image processing chip, optimization on program body is also a quite effective method. Taking into account the hardware characteristics of DM6437, the DSP chip, a simplified [3] Scale Invariant Feature Transform (SIFT [4]) image matching algorithm was developed. On the other hand, a lot of effort has been done for optimization subroutine itself based on the characteristics of C64x+ DSP core, such as using intermediate variables, intrinsic operations, word access to short type data, the compiler settings, software pipelining and linear assembly. It is showed by our final experiments that the operation speed has been significantly improved. For an image sequence with size of 256×256 pixels, the operation speed was improved from 4 seconds per frame to 25 frames per second, which meets the requirements of real-time location algorithm perfectly while the ranging error is less than 7%.

Keywords: video image, distance estimation, DM6437, optimization, SIFT

1. Introduction

Supported by a grant from the National Natural Science Foundation of China (No. 60872136), we are occupied with key technology research on monocular passive ranging. This ranging algorithm has constituted a special distance estimation equation by making use of target's azimuth and pitching, α_n and β_n, rotation-invariant linearity feature and observer's coordinates (x_n, y_n, z_n). And the linearity feature L_n can be obtained from adjacent frames through an image matching algorithm. Generally, the difficulty in image procession is large volumes of data and complexity of the algorithm. In the ranging algorithm, image matching based on SIFT was used for extracting the linearity feature as a more robustness and automation consideration. Therefore, the ranging algorithm, or exactly the linearity feature extracting algorithm becomes more complicated, and dominates the time-consuming part in total system. For realizing such an algorithm in real-time, it is preferable to use high-performance hardware to meet formidable operational overhead. Considering that, we choose TMS320DM6437 [5] as DSP processor.

The DM6437 is a fixed-point DSP based on the third generation high performance, advanced VelociTI™ very-long-instruction-word (VLIW) structure and faces digital multimedia applications. Not only it has the characteristics of high master frequency, quick operating speed, adopts and supports C64X+ instruction set as well as rich exterior connections, but also integrated with the video processing subsystems. So, it is very suitable for applications in digital media and video image gathering field. Its cost dropped 50% compared to DM642, making it becomes a special template which is suitable for all kinds of video image processing applications. Moreover, C64X+ includes a richer set of instructions compared to C64X. So it can effectively enhance the ranging algorithm timeliness and precision, and is very suitable for the sequence image tracking

[+] Corresponding author.
E-mail address: michaelhe@163.com.

and ranging process. Considering all these merits, we make DM6437 as DSP core processor in our sequence image tracking and ranging system.

The authors transplant the distance estimation algorithm based image matching into TMS320DM6437. After transplanted, it takes DSP 4 seconds to give a target distance output. That is far from the real-time requirements to the distance estimation algorithm.

2. Algorithm Introduction

Our system under developed can capture image in frame one by one, then sends the image data into processing unit, which carries on the image pretreatment, image matching, target distance ratio between adjacent frames in sequence and linearity feature information, then it combined with imaging direction information through optoelectronic theodolite and the observer's coordinates in 3D space by GPS, then accomplish the target tracking and passive ranging. All in all, this work must be done at 25 frames per second. Finally, the processed image displayed on display module; or the processed result can also be sent to tracking control unit. The imaging system block diagram is shown in Figure 1.

In Figure 1, the optoelectronic theodolite can follow the rotation of optical axis in imaging system, and provides the DSP target ranging system with information of azimuth α_n and pitching β_n for camera as the observer to imaged moving target. The camera's coordinates in three-dimensional space was supplies by the GPS unit. The main task of TMS320DM6437 is to execute target image matching and to acquire the target's rotation-invariant linearity feature L_n, L_{n+1} between adjacent frames. With these parameters, a distance estimation equation was settled for passive ranging based on monocular imaging system. Bellow is the ranging equation

$$D_2 r_{n+1}^2 + D_1 r_{n+1} + D_0 = 0 \tag{1}$$

Where

$$\begin{aligned} D_2 &= H'[1-(l_{n+1}l_n + m_{n+1}m_n + n_{n+1}n_n)^2] + \rho^4[(l_{n+1}l_n + m_{n+1}m_n + n_{n+1}n_n)^2 - 1] \\ &= (\rho^4 - H')[(l_{n+1}l_n + m_{n+1}m_n + n_{n+1}n_n)^2 - 1] \end{aligned} \tag{2}$$

$$\begin{aligned} D_1 = {} & 2H'\{l_{n+1}(x_{n+1}-x_n) + m_{n+1}(y_{n+1}-y_n) + n_{n+1}(z_{n+1}-z_n) - \\ & (l_{n+1}l_n + m_{n+1}m_n + n_{n+1}n_n)[l_n(x_{n+1}-x_n) + m_n(y_{n+1}-y_n) + n_n(z_{n+1}-z_n)]\} + \\ & 2\rho^3\{l_n(x_{n+1}-x_n) + m_n(y_{n+1}-y_n) + n_n(z_{n+1}-z_n) - \\ & (l_{n+1}l_n + m_{n+1}m_n + n_{n+1}n_n)[l_{n+1}(x_{n+1}-x_n) + m_{n+1}(y_{n+1}-y_n) + n_{n+1}(z_{n+1}-z_n)]\} \end{aligned} \tag{3}$$

$$\begin{aligned} D_0 = {} & H'\{[l_n(x_{n+1}-x_n) + m_n(y_{n+1}-y_n) + n_n(z_{n+1}-z_n)]^2 + \\ & (x_{n+1}-x_n)^2 + (y_{n+1}-y_n)^2 + (z_{n+1}-z_n)^2\} + \\ & \rho^2\{[l_{n+1}(x_{n+1}-x_n) + m_{n+1}(y_{n+1}-y_n) + n_{n+1}(z_{n+1}-z_n)]^2 - \\ & (x_{n+1}-x_n)^2 - (y_{n+1}-y_n)^2 - (z_{n+1}-z_n)^2\} \end{aligned} \tag{4}$$

$$H' = (\frac{f_n}{f_{n+1}})^2 (\frac{L_{n+1}}{L_n})^2 \tag{5}$$

f_n and f_{n+1} are denoted as focus of optical system in the camera at the n-th and (n+1)-th sampling time respectively [6];

$$\begin{cases} l_n = \cos\alpha_n \cos\beta_n \\ m_n = \sin\alpha_n \cos\beta_n \\ n_n = \sin\beta_n \end{cases} \tag{6}$$

$$\rho = L_{n+1} / L_n \tag{7}$$

The distance estimation equation shows as formula (1) requires observer in motion when estimate image target distance. This is suitable for the varifocal imaging system to the target distance estimate.

As for the linearity feature selection, the authors adopt the method introduced in the literature [6]. This method based on the SIFT image matching algorithm. After withdraws stable matching points from neighboring image frames, then selects three matching points conforms to the given conditions, then obtains the image target's linearity feature from a circumcircle of triangle determined by these three matching points.

3. Algorithm and Program Code Optimization

The purpose of optimization to a DSP system is to make a series of adjustments on hardware resources and simplification in software subroutine, which is of utmost important for a system requires real-time implementation. The software subroutine simplification can be handled at algorithm level, at program level, or either. The algorithm's efficiency is a very important factor in a DSP processing system. To meet the requirement of real-time realization, the algorithm's concise degree is the key consideration in designing. The code level optimization mainly begins with the program flow arrangement, DSP memories access and memory occupancy. This includes adjustment to branching instructions, circulation instructions and image data removal operations.

2.1 . Algorithm optimization

In the distance estimation algorithm of this paper, the most complex and the most time-consuming part is the SIFT image matching algorithm, among which to extract the partial maximum and minimum values so as to get characteristic points by using Gauss differential operator in the spatial scale, as well as to seek the SIFT descriptor process are by far the most complex parts, which is the primary factor restricts the distance estimation algorithm's real-time implementation. The authors have made vast improvements from the following several aspects to the SIFT algorithm:

1) As we known, key points in SIFT are gained from scale-space extrema in Gauss Pyramid images. Classics algorithm need establish Gauss Pyramid images in 6 scales. This computation is much time-consuming. For this reason, the authors adjust the Gauss Pyramid images to 4 scales, thus reduced computational process of 2 Gauss Pyramid images, simultaneously 65536×2 words memory space was saved. Then subtract two neighboring Gauss Pyramid images and 3 Gauss differential criterion space layers formed. Similarly, this also reduced certain computation load and 65536×2 words memory space saved.

2) In the process of calculating the gradient direction and the magnitude of middle Gauss differential images' each pixel, The authors only calculate the pixel's gradient direction and the module value of extreme points, but not to calculate every pixel's gradient direction and the magnitude. And then, by saving it into the extreme point chain instead of opening up array space, approximately 65536×6 words memory space was saved.

3) In the SIFT algorithm, the key point's descriptor is a 128 dimensions vector. The authors adopt dimension-reduced strategy [3]. Around the characteristic point's circular region, we only withdraw 20 dimensions vector according to certain rules, thus also reduced the computation load for the following matching process.

After the above algorithm optimization, the efficiency of the ranging algorithm has been enhanced greatly. Table 1 shows the cycles CPU consumed before or after algorithm optimization.

Table 1: The cycles CPU consumed with or without optimization

with or without optimization	The cycles CPU consumed
Algorithm before optimized	2,861,001,125
Algorithm after optimized	1,231,643,802

2.2 Program code optimization

2.2.1. Data type choice and transformation

The C6000 compiler defines a size for each data type [7] (signed and unsigned) listed in table 2:

Table 2: The C6000 data types

Data types	Bits
char	8
short	16
int	32
long	40

In the ranging algorithm, the authors give careful considerations to the data type size when writing code.

a) The C6000 registers are all 32-bit, so values larger than 32 bits, such as 40-bit and 64-bit quantities, are stored in register pairs. On the C6000 architecture, some of the ports for long and double-word operands are shared between functional units. This places a constraint on which long or double-word operations can be scheduled on a data path in the same execute packet. The authors replaced some unnecessary long data type by int data type, and the operations can be implemented using short type variables are all changed to short type variables. When establishing the Gauss Pyramid images using the SIFT algorithm, the authors applied the IQmath [8] library functions to convert the Gauss image int type data into short type, which not only decreases the memory space needed in large degree, but also brings about enormous conveniences for the subsequent computation and optimization.

b) As for the fixed-point multiplication, uses short type as inputs whenever possible, thus can use the C64x+ interior 16-bit multipliers effectively, and speeds up the instruction execution speed. It takes one clock cycle for "short * short" versus five cycles for "int * int". Using short type data to execute computation also brings about great conveniences as well to the application of intrinsics.

c) Using int or unsigned int types for loop counters, rather than short or unsigned short data types to avoid unnecessary sign-extension. The authors defined all loop counters as int type in the ranging algorithm.

2.2.2. Introduce intermediate variables

In the program segment with massive repeated computations, introduces some intermediate variables to save the intermediate result and avoid repeated computations can save some CPU cycles. In the target ranging algorithm of paper, there are much expressions that exist massive repeated computations as follows:

```
xyh=_IQdiv(((yt2-yt1)*(yt3*yt3-yt1*yt1+xt3*xt3-xt1*xt1)-(yt3-yt1)*(yt2*yt2-yt1*yt1+xt2*xt2-
xt1*xt1)),(2*(xt3-xt1)*(yt2-yt1)-2*((xt2-xt1)*(yt3-yt1))));
yxh=_IQdiv(((xt2-xt1)*(xt3*xt3-xt1*xt1+yt3*yt3-yt1*yt1)-(xt3-xt1)*(xt2*xt2-xt1*xt1+yt2*yt2-
yt1*yt1)),(2*(yt3-yt1)*(xt2-xt1)-2*((yt2-yt1)*(xt3-xt1))));
```

The cycles CPU consumed not using and using intermediate variables shown in Table 3.

Table 3: Comparison between using intermediate variables and not

optimization method	the cycles CPU consumed
Not using intermediate variables	1,231,643,802
using intermediate variables	1,231,312,346

Table 3 shows that after introducing intermediate variables, the clock cycles the ranging algorithm consumed reduced 331456. This method plays a certain role to reduce CPU cycles.

2.2.3. Using intrinsic operators

The C6000 compiler recognizes a number of intrinsic operators [9], which can optimize the computations quickly. Intrinsics are used like functions that map directly to inlined C64x+ instructions. The intrinsics are specified with a leading underscore, and are accessed by calling them as you do a function. The frequently used intrinsics listed in Table 4.

Table 4: The frequently-used intrinsics

Intrinsics	assembly instruction
int _abs(int src)	ABS
int _abs2(int src)	ABS2
int _add2(int src1,int src2)	ADD2
int _sub2(int src1,int src2)	SUB2
int _mpy(int src1,int src2)	MPY
uint &_amem4(void * ptr)	LDW,STW

The comparison of cycles CPU consumed in the distance estimation algorithm between using and not using intrinsics shown in Table 5.

Table 5: Comparison between using intrinsics and not

optimization method	the cycles CPU consumed
not using intrinsics	869,602,126
using intrinsics	868,403,082

2.2.4. Using word access to short type data

In every clock cycle, the C6000 DSP can only complete 2 times memory access. In order to maximize data throughput on the C6000, it is often desirable to use a single load or store instruction to access multiple data values consecutively located in memory. For example, all C6000 devices have instructions with corresponding intrinsics, such as _add2( ), _mpy2( ), and _sub2( ), that operate on 16-bit data stored in the high and low parts of a 32-bit register. Thus, when operating on a stream of 16-bit data, you can use word (32-bit) accesses to read two 16-bit values at a time, and then use intrinsics to operate on the data. This will reduce CPU much access times to internal memory and save CPU overhead. The following circular program segment used to establish the differential Gauss images. The CPU visits to each array element directly whose data type is short. The number of circulation times is 256×256.

```
for(i=0;i<256;i++)
{
   for(j=0;j<256;j++)
   {
        unsigned short tem;
        tem=_mpy(i,256)+j;
                                  dog[0][tem] = dog[1][tem]-dog[0][tem];
                                dog[1][tem] = dog[2][tem]-dog[1][tem];
                                  dog[2][tem] = dog[3][tem]-dog[2][tem];
   }
}
```

Upon using word access to short type data in the above program segment, the number of circulation times changes to 256×128, and the program after using intrinsics shown as follows:

```
for(i=0;i<256;i++)
{
  for(j=0;j<128;j++)
  {
    unsigned short tem;
    tem=_mpy(i,256)+2*j;
   _amem4(&dog[0][tem])=_sub2(_amem4(&dog[1][tem]),_amem4(&dog[0][tem]));
   _amem4(&dog[1][tem])=_sub2(_amem4(&dog[2][tem]),_amem4(&dog[1][tem]));
   _amem4(&dog[2][tem])=_sub2(_amem4(&dog[3][tem]),_amem4(&dog[2][tem]));
  }
}
```

The comparison of cycles CPU consumed between using and not using word access to short type data method shown in Table 6.

Table 6: Comparison between using word access to short type and not

not using or using optimization	the cycles CPU consumed
Before optimization	23,192,956
After optimization	11,274,437

2.2.5. Changing the compiler's behavior with options

The C6000 compiler provides massive options [9] can control the operation of the compile. The most frequently used compiler optimization options shown as follows:

-pm Combines source files to perform program-level optimization.

-mt The option indicates that your code does not use the aliasing technique. The option also indicates that a pointer to a character type does not alias (point to) an object of another type. If your code does not contain any of the aliasing techniques, you should use the option to improve the optimization of your code.

-on The n denotes the level of optimization (0,1,2,and 3), which controls the type and degree of optimization. The larger digit is following -o, the higher level is code optimized. The -o3 is the maximum amount of optimization, but in some cases, the error may possibly occur when using -o3 to perform loop optimizations and software pipelining (-o2 possesses similar phenomenon, whereas -o0 and -o1 will not).

-msn Controls code size on four levels (0, 1, 2, and 3). The larger the value of n, the more effort the compiler invests in optimizing your code.

-k Keeps the assembly language (.asm) file.

-g Enables symbolic debugging.

The target ranging estimation algorithm is not willing to produce software pipeline before making some adjustments, therefore uses the following options:

C16x -g -k -pm -ms0 -o1

After making some adjustments, the most iterative programs can form the software pipeline, therefore uses the following options:

C16x -k -pm -ms0 -o3

2.2.6. Optimizing software pipeline

The C64x/C64x+ DSP pipeline [10] provides flexibility to simplify programming and improve performance. Software pipelining schedules instructions from a loop so that multiple iterations of the loop execute in parallel. When you use the -o2 and -o3 compiler options, the compiler attempts to software pipeline your code with information that it gathers from your program. In multiple iterations, the inner nested loop is the only one can form software pipeline. There are some limits to form software pipeline as follows:

(1)Although the software pipeline loop can contain the intrinsic, but the ordinary function call should be avoided;

(2)The conditional branches cannot be contained in the circulation to terminate the loop, and the break instruction cannot be included either;

(3)The loop counter supposed to be in declined manner. If the loop counter changed within the body of a loop, the loop cannot form software pipeline;

(4)If the size of a loop is too big and uses more than 64 registers (regarding C64x/C64x+), the loop cannot form software pipeline. When this is case, it is required to simplify the loop code or to change it into some small ones;

(5)If the loop control is too complex, try to simplify the loop. In a loop body contains complex conditional branches (such as complex if-then-else structure), the number of register used for condition judgment not supposed to surpass 6.

a) Using MUST_ITERATE

A trip count is the number of loop iterations executed. The trip counter is the variable used to count each iteration. When the trip counter reaches a limit equal to the trip count, the loop terminates.

The minimum safe trip count is the number of iterations of the loop that are necessary to safely execute the software pipelined version of the loop. All software pipelined loops have a minimum safe trip count requirement. If the known minimum trip count is not above the minimum safe trip count, redundant loops will be generated. Sometimes the compiler cannot determine if the loop always executes more than the minimum safe trip count. Therefore, the compiler will generate two versions of the loop. One is the unpipelined version that executes if the trip count is less than the minimum safe trip count. The other is the software-pipelined version that executes if the trip count is equal to or greater than the minimum safe trip count. In order to help the compiler generate only the software pipelined version of the loop, use the MUST_ITERATE pragma can guarantee that a loop executes a certain number of times. The syntax of the pragma is:

#pragma MUST_ITERATE([min, max, multiple]);

The arguments min and max are programmer-guaranteed minimum and maximum trip counts. The trip count of the loop must be evenly divisible by multiple.

The number of times the nested loop iterates in 2.2.4 is 128. The following pseudo-instruction tells the compiler that the loop is guaranteed to run exactly 128 times:

#pragma MUST_ITERATE (128, 128);

The comparison of cycles CPU consumed between using MUST_ITERATE and not using shown in Table 7.

Table 7: Comparison between using MUST_ITERATE and not

optimization method	the cycles CPU consumed
not using MUST_ITERATE	11,274,437
using MUST_ITERATE	11,135,548

As to the number of times a loop iterates is uncertain, using pseudo-instruction can inform the compiler that a loop executes at least a certain number. For example, the inner nested loop in the ranging algorithm that calculates the Gauss filter template executes at least 9 times, use pseudo-instruction as follows:

#pragma MUST_ITERATE (9);

b) simplifying loops

Within a nested loop, the innermost loop is the only one can form software pipeline. When a loop contains cause-effect relationship, judgment and frequently jump out of it will seriously prevent the code parallelism and pipelining. It is necessary to reduce the condition branches as far as possible. In order to exert superiorities of code parallelism, to adjust the nested loop flow and reduce condition branches or move it out of the loop body is necessary. In the ranging algorithm, to seek extreme point among DOG differential images, a great deal of condition branches used in loop body. Therefore, the complier can't form software pipeline. On the one hand, the authors have made some improvements to the SIFT algorithm. Considering the pixel values either negative or positive on the DOG differential images, when establishing the DOG differential images only need to save its absolute values. Consequently, it only needs to seek maximum value on the DOG differential image, which caused the condition branches reduced one half. On the other hand, the authors recombined the judgment instructions and merge it into one, thus improve the execution speed and reduce the code size also.

2.2.7. Using linear assembly

When the compiler does not fully exploit the potential of the C6000 architecture, you may be able to get better performance by writing your code (mainly loop modules) in linear assembly [10]. Linear assembly is the input for the assembly optimizer.

Linear assembly is similar to regular C6000 assembly code in that you use C6000 instructions to write your code. With linear assembly, however, you do not need to specify all of the information that you need to specify in regular C6000 assembly code. With linear assembly code, you have the option of specifying the

information or letting the assembly optimizer specify it for you. Here is the information that you do not need to specify in linear assembly code: parallel instructions, pipeline latency, register usage and which functional unit is being used. But there are still some restricts need to be considered. For example, it is supposed that the branches skip out of the loop body should be avoided to optimize the loop body. Moreover, loops that can be most efficiently software pipelined have loop trip counters count down. Otherwise, the assembly optimizer cannot efficiently complete the optimization work possibly.

As for 2.2.4 code segment, the comparison of cycles CPU consumed between using linear assembly and not shown in Table 8.

Table 8: Comparison between using linear assembly and not

optimization method	the cycles CPU consumed
C code	11,135,548
using linear assembly	864,536

Table 8 shows that after using the linear assembly the cycles CPU consumed decreased greatly.

4. Comprehensive Optimization Result

By comparison, after using above algorithm optimization along with the combination of all above optimization strategies to the target ranging estimation algorithm, the cycles CPU consumed decreased vastly shown as table 9, and the final compiler optimization options set as: C16x -k -pm -ms0 -o3

Table 9: Comparison between using comprehensive optimization and not

optimization method	the cycles CPU consumed
before optimization	2,861,001,125
using comprehensive optimization	23,101,254

Table 9 shows that after the target distance estimation algorithm optimized, its efficiency has been enhanced enormously. The master frequency of DM6437 is 600MHz. The time it takes to process one 256×256 pixels image frame is: 23101254/600000000=0.0385ms. So one second it can process 25 image frames, and this processing speed can satisfy real-time implement requirement.

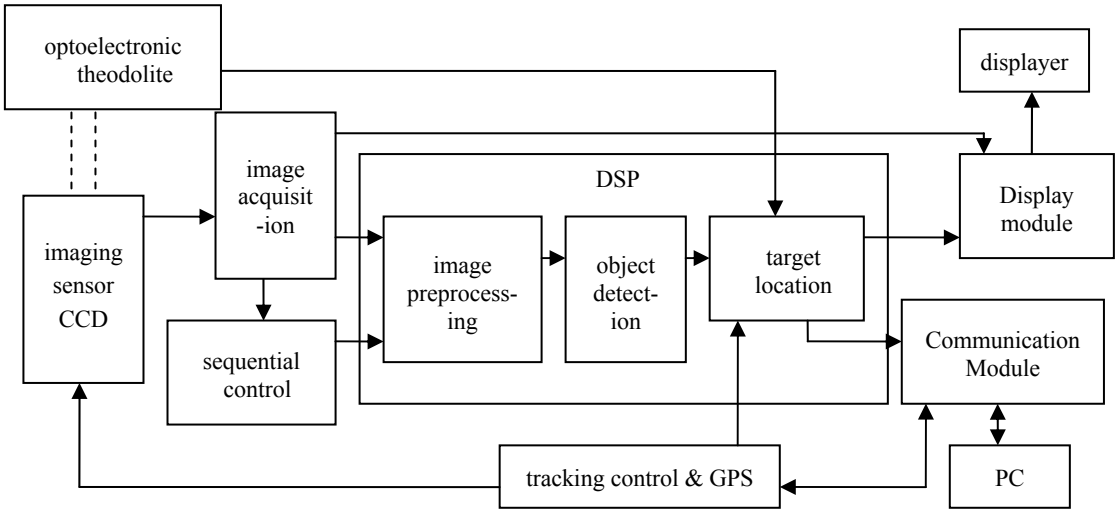

Fig. 1: The imaging system block diagram

5. Conclusion

Upon fully understanding to C64x+ core characteristics, the authors made a variety of optimization to the target distance estimation algorithm both at algorithm level and program level. At program level, a series of strategies such as using intermediate variables, intrinsics, word access to short type data, compiler settings,

and software pipelining and linear assembly are applied to it. As a result, the program parallelism was enhanced extremely, and the hardware resources are fully utilized. The final experiment shows that the target distance estimation error is smaller than 7%. The ranging algorithm can be accomplished at the speed of 25 frames per second, thus well satisfies the real-time application requirements.

6. Acknowledgment

This Project was supported by a grant from the National Natural Science Foundation of China (No. 60872136) and Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2011JM8002).

7. References

[1] FU Xiao-ning, GAO Wen-jing, and WANG Da-bao. “A method for distance estimation based on the imaging system,” Chinese Patent, 201010107208.2.

[2] WANG Di, FU Xiao-ning. “Method for Imaged Target Ranging Based On Line Segment Features”. CCCA 2011, February 20-21, 2011, Hongkong.

[3] ZHU Fu-shuai, “Research on the SIFT for image matching,” Xi’an, Xidian University, Jun. 2009.

[4] Lowe D G. Distinctive Image Features from Scale-Invariant Key-points[J]. International Journal of Computer Vison.2004，60(2):91-110.

[5] Texas Instruments Incorporated. TMS320DM6437 Digital Media Processor. SPRS345D, 2008.6

[6] FU Xiao-Ning, LIU Shang-Qian, and LI En-Ke, “A real time image sequence processing algorithm for target ranging,” Proceedings of SPIE - The International Society for Optical Engineering, 27th International Congress on High-Speed Photography and Photonics, vol. 6279 PART 2, 2007, pp. 62793A.

[7] TMS320C6000 Programmer’s Guide, SPRU198J, 2010.4

[8] TMS320C64x+ IQmath Library User's Guide, Sprugg9, 2008

[9] TMS320C6000 Optimizing Compiler v7.0 User's Guide, Spru187Q, 2010.2

[10] Texas Instruments Incorporated. TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide. SPRU732J, 2010.7