CMPT 414 Model-Based Computer Vision

Group Project

Stereo Matching by Image Cut

Presented by:

Bowen Wang 301267523

Enhong Han 301324883

Project Demo Link: https://youtu.be/Zp8P9vZexDc

Abstract

The disparity of an image is very useful for identifying the distance from each object to the camera. There are several ways of doing stereo matching, including Block matching, Hough Method, and Dynamic Programing. In 2001, Boykov introduced a new method, Graph Cuts to find approximation to the minimum energy of the image via *min-cut(max-flow)* in polynomial time. Our group are interested in his article and following improvement. This report is about our implementation of part of this method and test with classical stereo matching image *tsukubai*.

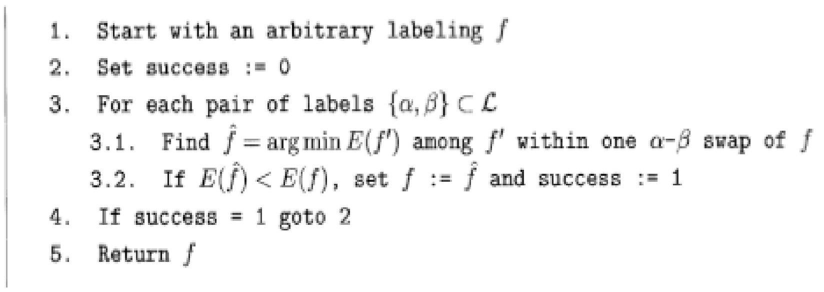
Methodology

1. **Energy Minimization**

   As many other computer vision tasks, Stereo Matching can be turned into an Energy Minimization problem where the matching cost is minimized, and some constraints can be enforced. Initially, a graph is constructed by connecting pixels to all of their possible disparities, but in our implementation, we added some heuristic to the pixel energy initialization as we use the minimum intensity difference among each disparity to eliminate further calculation. The total energy of each disparity is measured as $E(f) = \sum_{p \in P} D_p(f_p) + \sum_{(p,q) \in N} V_{p,q}(f_p, f_q)$ The stereo matching image is the image with minimum total energy among all disparities. There are several algorithms to achieve this, we choose to use $\alpha - \beta$ swap to help us do the process.

2. **$\alpha - \beta$ Swap**

   If the image has more than one disparity to separate, the difficulty of the problem increases from P to NP-hard. To divide the problem and conquer, basic idea is reducing the problem to a series of 2-way-cut sub-problems. $\alpha - \beta$ swap is a method to help with this situation. To implement $\alpha - \beta$ swap, first we need calculate the combination of designed disparity interval, for each combination we look for all possible swaps between element in α set and β set, each iteration we pick a number of pixels to do swap and calculate the minimum total energy from possible swaps. If the minimum total energy less than current total energy, which indicates the swap have positive effect on energy minimization, the swap is confirmed and now we look for next possible swap until no good swap can be made. The pseudocode is shown in Figure I.



```
1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each pair of labels $\{\alpha, \beta\} \subset \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min E(f')$ among $f'$ within one $\alpha$-$\beta$ swap of $f$
   3.2. If $E(\hat{f}) < E(f)$, set $f$ := $\hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$
```

Figure I. Pseudocode for $\alpha - \beta$ Swap

3. Min-Cut/ Max-Flow

By using Min-Cut/ Max-Flow method we can easily get paths of minimum energy from source disparity to sink disparity, which support us do multiple $\alpha - \beta$ swap at the same time. Min-Cut/ Max-Flow method requires 3 steps ("Grow", "Augment ", "Adopt") and 3 types of nodes ("Active", "Passive", "Orphan"). In Grow step, grow search trees from S and T at the same time until 2 search trees meet at any node. Then do Augment step, split the search tree into sub-trees or forest according to the result from Grow step. Last, in Adopt step, restructure the search tree of S and T. The pseudocode writes like this:

```
Initialize: S ={s}, T={t}, A={s,t}, O=∅
while true
    Grow S or T to find an augmenting path P from s to t
    If P =  ∅
        Terminate
    Augment on P
    Adopt orphans
end while
```

a) Grow Step:

In Grow step, expand the search tree. Find neighbor nodes of active nodes with available capacity to be new child of the tree and label the node "active" as well. Do the search repeatedly until every neighbor of active node has been scanned, change the state of the node to "passive". This step ends when any "active" node starts to scan nodes in the other tree. To finish this step, find a grow path. The pseudocode writes like this:

```
while A != 0:
    pick an active node p∈A
    for every neighbor q such that
        tree_cap(p→q)>0
    if TREE(q) = Ø then add q to search tree as an active node
        TREE(q):= TREE(p),PARENT(q):= p,
        A:= A ∪{q}
    if TREE(q)!= Ø and TREE(q) != TREE(p)
        return P = PATH s→t
    end for
    remove p from A
end while
return P=Ø
```

b) Augment Step:

In Augment Step, we are looking for the max flow in the path found in Grow Step that make an edge on the path fully filled.   The child node linked with this edge is called "orphans". During the Augment Step, S and T can be split into multiple trees. The pseudocode writes like this:

```
find the bottleneck capacity Δ on P
update the residual graph by pushing flow Δ through P
for each edge (p,1) in P that becomes saturated
    if TREE(p) = TREE(q) = S then set
        PARENT(q) = Ø and O = O U{q}
    if TREE(p) = TREE(q) = T then set
        PARENT(p) = Ø and O = O U{p}
end for
```

c) Adopt Step:

In Adopt Step, we find each orphans a new parent that ensure capacity not overflow. The pseudocode writes like this:

```
while O != Ø
    pick an orphan node p∈O and remove it from O
    process p
end while
```

Approaches

We choose to do our implement in Python, which has more package support. In our code, we used following packages help us finish our task:

cv2 package for basic image processing includes image reading, image show
maxflow package for Min-Cut/Max-Flow path finding.
numpy package for data stream processing.
itertools package for getting combinations of $\alpha - \beta\ Pairs.$
copy package for deep copy lists.

Firstly, we process the image to get intensity of each pixel and find the edge of each pixel record into a 2D-array. Then, we build our initial state by comparing intensity difference between each pixel from right image and pixel with disparity on left image, choose the disparity with minimum intensity difference as the initial disparity for each pixel.

Secondly, we calculate energy of each disparity. As

$$E(f) = \sum_{p\in P} D_p(f_p) + \sum_{(p,q)\in N} V_{p,q}(f_p, f_q).$$

We choose to calculate D term simply by intensity difference between left and right image |Il-Ir|, if the pixel is around the boundary of the image, we use the data energy of the pixel on its left to represent its data energy. For V term, we first use the function introduced in our lecture notes which is the disparity difference of the pixel and its neighbours to represent. For neighbour choosing, we choose to use 4 pixels (up, down, left, right) as the neighbour of the pixel. However, in later testing we find it's not accurate for energy estimation. Considering the pixels which have large intensity difference with neighbors should more likely to be in different disparity group, we find another conditional function to estimate smooth energy that based on both current label and intensity difference write like:

$$(\alpha\ != \ \beta) * (C_1 * (|I_r - I_l| \geq t) + C_2 * (|I_r - I_l| < t))$$

α, β are the label of current pixel and its neighbor, t is threshold of intensity difference, C_1 is coefficient of energy that intensity difference greater than threshold. C_2 is coefficient of energy that intensity difference smaller than threshold. With the help of this function, our energy estimation becomes better for later use.

Thirdly, we start $\alpha - \beta$ swap, which is the most important part for stereo matching. Our team tried several methods to implement this algorithm, each approach of this step is listed below.

1. Brute-Force $\boldsymbol{\alpha - \beta}$ swap

We first implemented a most naïve version of $\alpha - \beta$ swap algorithm, which basically does one swap each time and calculate total energy after each swap? No doubt the performance is bad. By testing with "*tsukuba*", which is a 384*288-pixel image, this method took more than

9 hours to solve the problem (still no result returned).

2. Improved Brute-Force $\alpha - \beta$ swap

After checking the logic of the algorithm, we notice there are many energy terms are calculated multiple times. To be more specific, each time we calculate total energy of a set, we calculate energy of every element in the set. However, it is not necessary to do this as majority of elements keep unchanged, only one element swap contribute to the total change. Thus, we decided to improve this approach by measuring energy change instead of calculating total energy repeatedly

3. Min-Cut/ Max-Flow

With the help of *PyMaxflow* package, we can take use of Min-Cut/Max-Flow to significantly improve $\alpha - \beta$ swap performance. By doing this, We create a new graph with all pixels in alpha set or beta set through *maxflow.Graph[float]*. Then we add edge between each pixel and its left and up neighbors if any. For the energy between each node, we use the sum of data term and smooth energy term calculated by our improved function. Remember, less energy assign means easier to cut, the conditional function has great support on energy estimation and cut accuracy.

Then we do *maxflow()* built in *PyMaxflow* library help us find the min-cut and we do the label change in our sets until global min energy reached.

Results

The results of our disparity image are saved in folders with variable applied. Folder name "XX-XX-XX" indicates "*energy assign for intensity difference greater than threshold - energy assign for intensity difference less than threshold - threshold*". In each results folder, several image files are named on the iterations of alpha-beta swap. Note, the last iteration shouldn't have any difference comparing the second last iteration as the last iteration is just checking whether there is any swap can make, if not, terminate swapping.

We first used $min(|I_r - I_l|, 20)^2$for data term, but we find the result is not good. From Figure II, we can find there are many linear noises in the disparity image, we conclude the reason on the square significantly enhanced intensity difference and the *min()* makes a cutoff of energy assignment to large intensity difference area which increased the error of gathering them together.

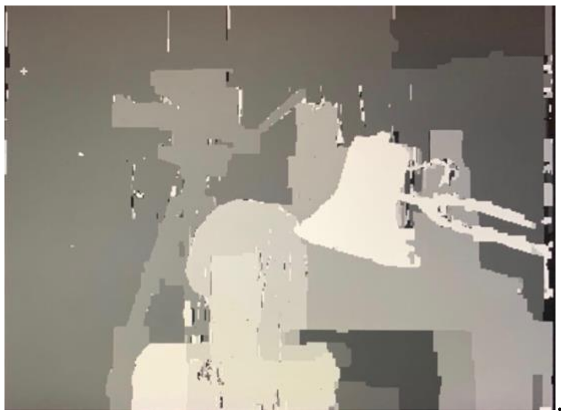
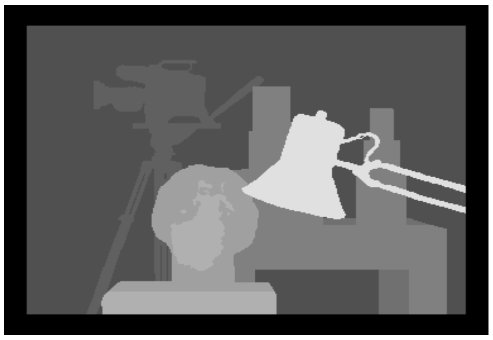
.

Figure II: Previous Result

After we change the data term energy function to $|I_r - I_l|$, noises are eliminated. Here is our best result choose from our testing with threshold = 5, grater intensity energy = 10, less intensity energy = 15.

(a) Input Right Image


(b) Ground Truth

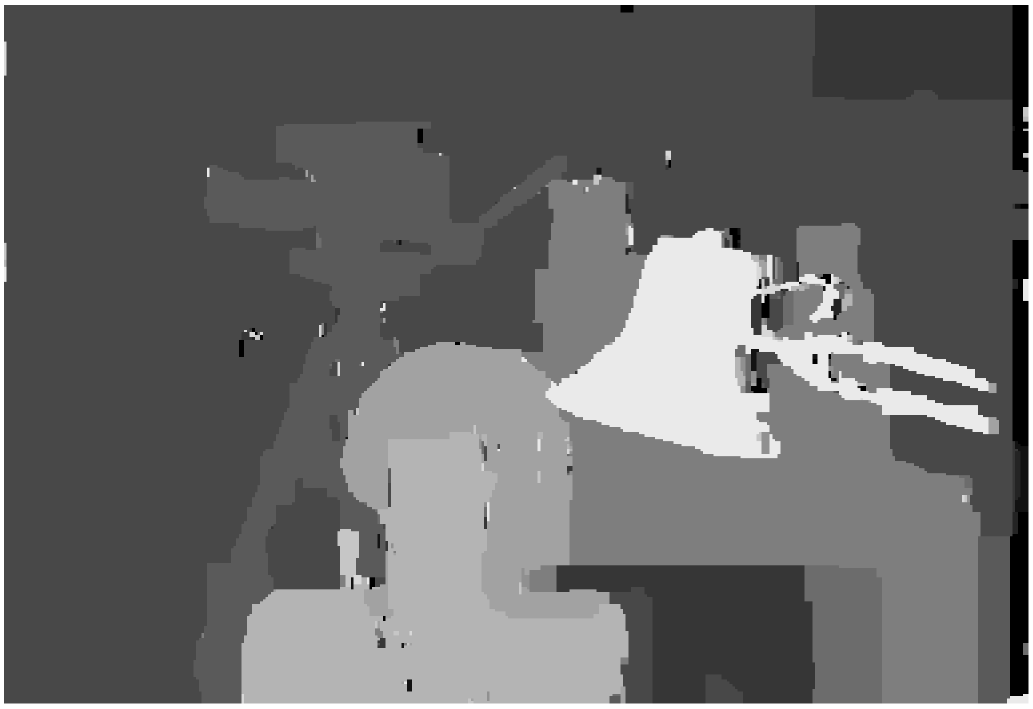
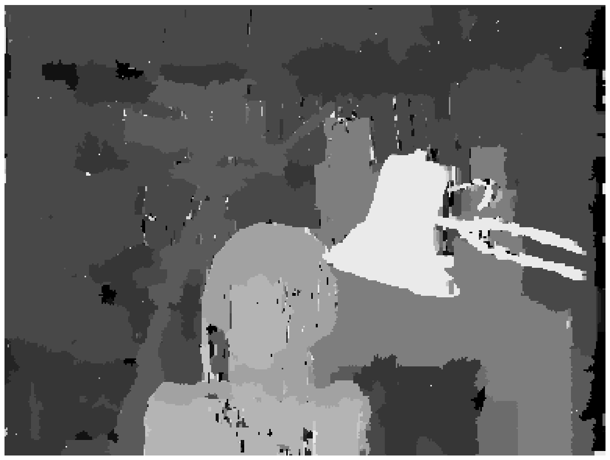
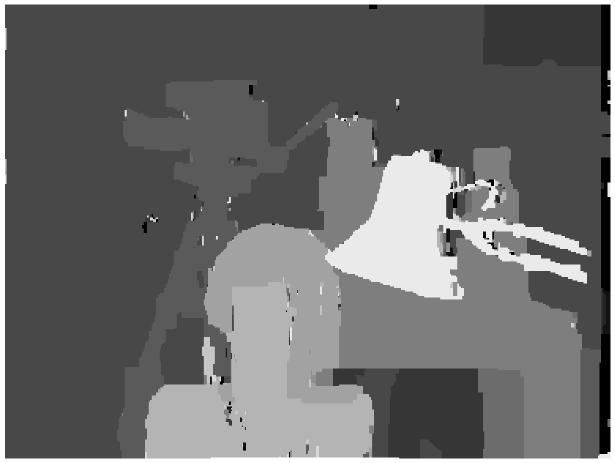

(c) Our Result

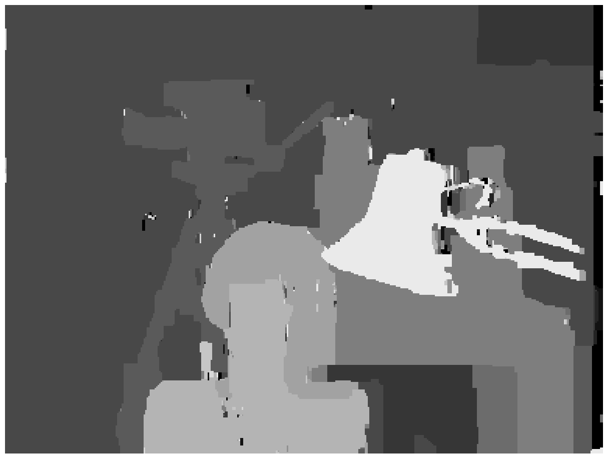
Figure III: Final Results

The final result is the 5th iteration of $\alpha - \beta$ swap, the previous iteration result looks like this:


1st Iteration


2nd Iteration

3rd Iteration


4th Iteration

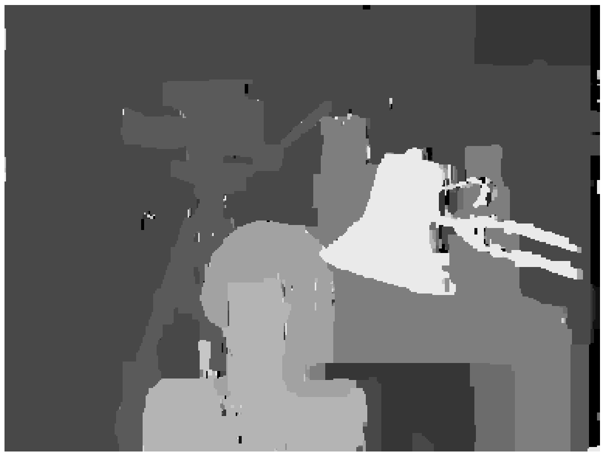

5th Iteration (Same as 4th Iteration)

Figure IV: Results from Iterations

Analysis and Summarize

Our implementation has a good result on disparity matching. The objects are mainly positioned in the right disparity but still kind of away from the ground truth. Here is our analysis on our result and some side problems.

Our Results

From our result image, it's easy to identify the camera, the desk, the statue and the lantern. However, there are some noise occur at high explore spots like the highlight on the lantern and cans on the desk. These spots have large intensity difference comparing to its neighbors which can be a difficulty for $\alpha - \beta$ swap to justify. Moreover. the pixels on the boundary of the image are failed to match their disparity. The main reason is, we choose to use right image as our base but the pixels on the right boundary don't have any reference pixels on the left image, which makes our algorithm doesn't work on these pixels.

Coding Notes

During our implementation, there are several challenges to overcome includes debugging, improve performance, and testing.

1. *cv2* and *pillow* has different image data structure in python, keep using one package in coding can reduce hardness of maintaining.
2. *list.index()* has bad performance when dealing with large *ndarrays* as the complexity of it is O(n2).
3. Avoid re-calculation plays an important role in improving performance
4. If *dictionary* is used in code, use *dictionary.get()* anytime you can.
5. Although Python has its own memory management system, still remember to delete useless data.
6. Every algorithm has its limit, improve algorithm is the only way to break the bottleneck if facing it.

To get our final result, we put effort in several jobs include finding and understanding resources, variable and interval testing, and performance improvement. By implementing $\alpha - \beta$ swap and use Min-Cut/Max-Flow method, we find these are very intelligent ideas and have outstanding performance comparing to classical stereo matching algorithms. However, as the complexity still in NP-Hard and the unsolved NP problem, this algorithm still has its limitations on running time (our best-case running time is 1660 seconds on Intel i7 processor).

Project Experience Summary

Bowen Wang:

- Code testing on variable choosing
- Restructure energy function and performance improvement
- Report structuring and writing
- Demo Presentation

Enhong Han:

- Resources gathering and mechanism understanding
- Code implement and develop
- Code testing and bug fixing
- Report revising and editing

Reference

Scharstein, D. (2011, Feb 11). *2001 Stereo Datasets*. Retrieved from http://vision.middlebury.edu/stereo/data/scenes2001/

Yuri Boykov, O. V. (2001, Nov). Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yuri Boykov, V. K. (2004, Sep). An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.