

Computer Vision Final Project Report

B05202061 電機四 陳威旭

B05501032 電機四 林士鈞

B05901027 電機四 詹書愷

B04502040 電機四 徐瑞擇

Abstract

我們選擇使用傳統方式，即分成四個步驟：Cost Computation, Cost aggregation, Optimization 和 Refinement。為了使結果更為準確，我們首先增加了 preprocessing 的步驟，並集合了三種 cost 以增加 cost 的準確度。最後，我們使用 edge detection 以及 segmentation 增加提升輸出影像的品質。

Observations

1. 演算法在 Synthesize data 上只要夠平滑就能拿高分，但套用在 Real data 上時，不能太依賴 filter，否則產出的 disparity 很容易太過模糊。故我們再 Synthetic data 上只有採用部分 refinement 與 aggregation 方式，而在 Real data 上則是採用了以下提到的所有做法。
2. 在 cost aggregation 階段若太早使用 filter 會使得 LR check 的結果較差。
3. 使用 LR check 之前先用 edge detection 找出 border，可以讓輸出效果更好。
4. 由於演算法在判斷一些 Real data 時偵測到的 maximum disparity 值太低，因此我們事先將太低的 maximum disparity 調高。

Algorithms

Preprocessing

1. Histogram Equalization

對圖片做直方圖均衡化，增加圖片的對比度，可有效增加判斷 disparity。

2. Max disparity：

先使用 SURF，得到一些對應點，算對應點兩者 point 位置差距，然後排序全部的差距大小，然後從最大值的往下看，選擇一個密集的差距值，來當作 Max disparity，例如，差距大小為：76, 60, 59, 59 時，會選擇 Max disparity 為 60，因為 76 可能是 SURF 誤判，故捨去。

Cost Computation

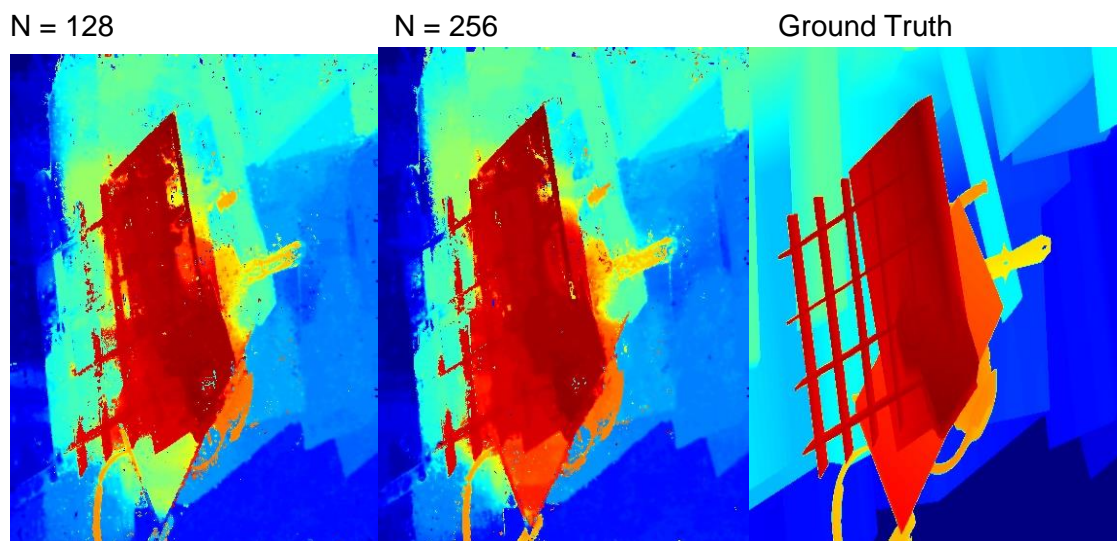
1. Baseline cost

我們使用 L1 cost，並直接對相同 disparity 的 cost 用 guided filter 做平滑化。計算出來的 cost 較為平滑。

2. Binary Stereo Matching (BSM) cost

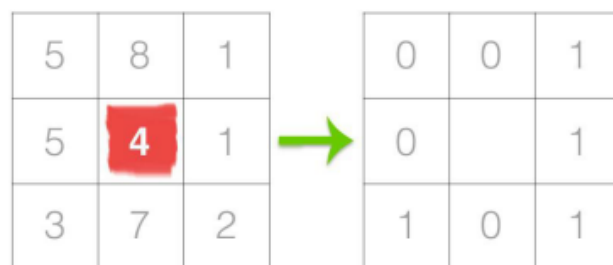
在所有 pixel 周圍 sample 兩兩一對的 pair 共 N 個,並依條件產出一串 N -bit 的數列當作此 pixel 的特徵向量，最後對所有在 max_disp 以內的 pixel 計算 cost。如此計算出來的 cost 會有一些顆粒狀的雜訊。

使用的 N 值越大，產出的 Disparity Map 理論上會越精準。以下是 $N=128$ 和 $N=256$ 的比較：



3. Census Cost

Census Cost 與 BSM 之概念類似，皆使用 pixel 之間的關係作為算 cost 的依據。而 census cost 是針對某一個 pixel 周圍與其之大小關係，而生出相對應的 binary pattern，代表此 pixel 的特徵向量。左圖與右圖會用此向量計算不同 disparity 的 Hamming distance (cost)。



Local binary pattern

4. Cost Merge

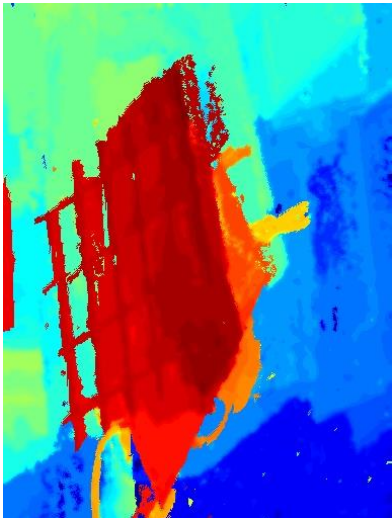
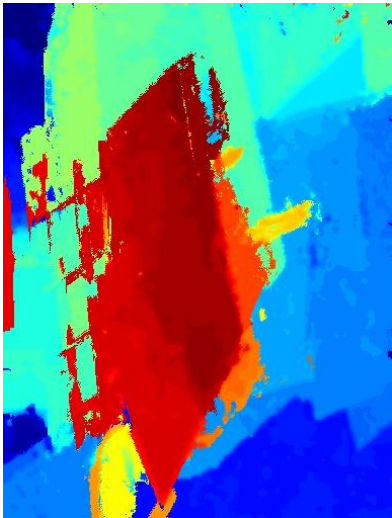
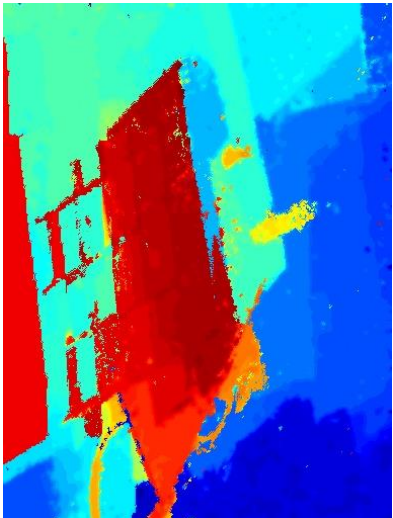
我們將以上三種方式算出來的 Cost 用 $\frac{1}{N} \sum_{i=1}^3 (1 - e^{-c_i/w_i})$ 做加權平均，其中 c_i 是原本的 cost, 而 w_i 則是加權平均的權重。經過試驗後我們使用的權重為：

Census cost: 350

Base cost: 100

BSM cost: 100

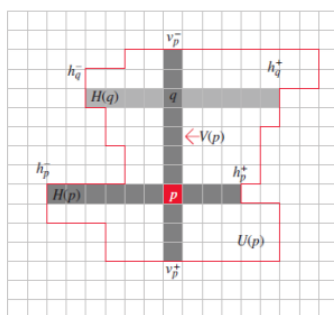
三種 Cost 的比較：

Base	BSM	Census
		

Cost Aggregation

我們 Aggregation 使用的是 Cross-based cost aggregation，其可以根據每一個 pixel 所位於的區域，為其找到最適合做 aggregation 的區域。其演算法主要有三步驟：

1. 對於每個 pixel，找到其上下左右，在顏色相近的條件下，能延伸最遠的距離 (arm distance)。對於一 pixel p ，他所要 aggregate 的區域 $U_l(p)$ ，便是其上下的 arm 中的每一個 pixel，沿著自己左右的 arm 延伸出去後所覆蓋的區域。細節如下圖：



Find the largest arm span:

$$r^* = \max_{r \in [1, L]} \left(r \prod_{i \in [1, r]} \delta(p, p_i) \right)$$

$$\delta(p_1, p_2) = \begin{cases} 1, & \max_{c \in \{R, G, B\}} (|I_c(p_1) - I_c(p_2)|) \leq \tau \\ 0, & \text{otherwise} \end{cases}$$

2. 將左圖的區域 $U_l(p)$ 與右圖相對應的區域 $U_r(p-d)$ 取交集，得到 $U(p)$ ，而 p 是對應到的 disparity。
3. 取 $U(p)$ 的平均 cost。

由於前述的演算法皆可使用 `numpy`，以矩陣的方式平行計算，再加上取平均的步驟可以使用 `Integral Image` 的方式加速，故整個 `aggregation` 的運算可在 $O(n)$ 底下完成(假設平行的計算不會有 overhead)。

Optimization

1. WTA

使用萬能的 WTA 解決。WTA 可以說是諸多方式中表現最適中的，不會有太大的錯誤出現。

Refinement

1. 捨去突出值

每個點和附近的九宮格比較進行差距確認，要是和全部的點的差距，都超過一定值的話，就以其為中心，張開一個中型 `window`，看這個 `window` 之中，哪個 `disparity` 出現次數最多，更換成這個值。

2. Left-Right Check and Scan Line

為了減少 WTA 的錯誤，利用 `left disparity map` 與 `right disparity map` 找出不滿足下式的點

$$D_L(p) = D_R(p - D_L(p))$$

這些點在左圖與右圖的偏差數值不一致，將這些點的 `disparity` 更換成 `epipolar line` 上最近之一致點的 `disparity`，最後經過 `weighted median filter` 進行調整。

3. Bilateral Filter

使用 `bilateral filter` 將 `scan line` 的結果進行平滑化處理。

4. Edge Detection

為了明確地分出遠景與近景的邊界，利用 `sobel filter` 計算出 `disparity map` 的梯度，由梯度差值的大小判斷出遠景與近景之邊界點，從 `cost map` 中找出邊界點與邊界附近點的 `cost`，將邊界點的 `disparity` 更換成 `cost` 較相近的點之 `disparity`。

Other Methods We Tried But Failed

1. Denoising

我們嘗試使用 Non-local means denoising, 但是他會將 disparity map 過度平滑化, 導致細節過於模糊, 雖然在處理 Synthesized data 時會有較好的效果, 但考慮到 Real data 要求較細的細節, 我們最終不使用它。

2. Subpixel Enhancement

利用二次多項插值法將 disparity map 中所有的邊界平滑化, 其中包含 cost 誤差導致的錯誤邊界與正確的邊界, 再利用 edge detection 邊界值強化, 然而此方法並沒有造成很明顯的優化, 因此最後並未使用。

$$d_{new} = d - \frac{C(p, d + 1) - C(p, d - 1)}{2(C(p, d - 1) + C(p, d + 1) - 2C(p, d))}$$

3. Using the mean of the 3 smallest disparities for each pixel

在 Optimization 階段, 我們嘗試使用最小的三個 cost 的位移, 取其平均值當作 disparity, 這麼做是希望能夠減少雜訊, 但測試後發現這麼做沒有效果。

Results

Machine SPEC

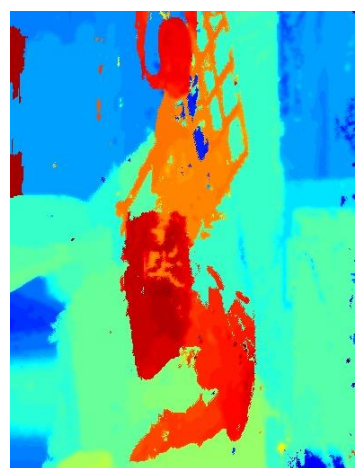
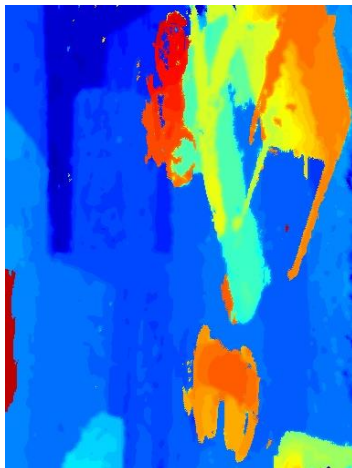
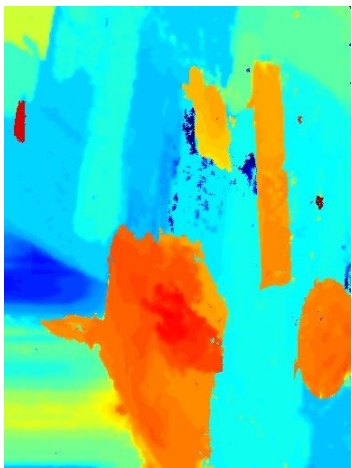
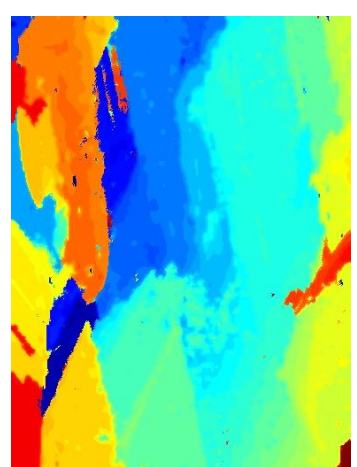
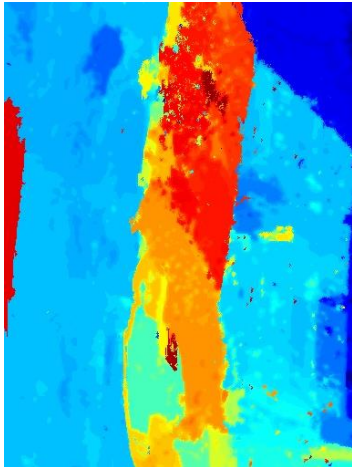
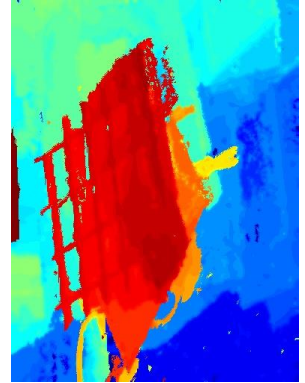
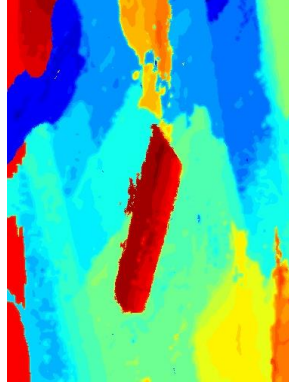
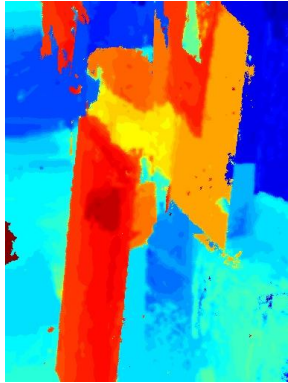
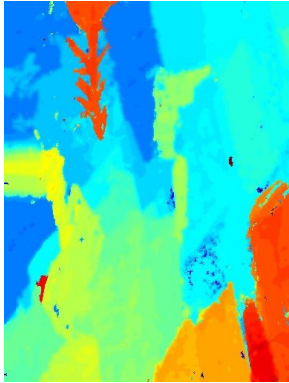
Synthesized: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

Real: Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz

Synthesized Data

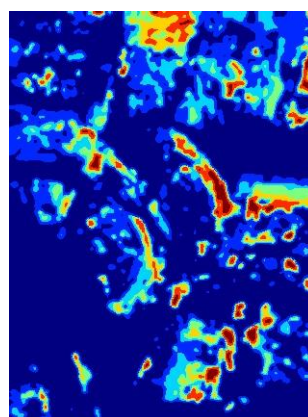
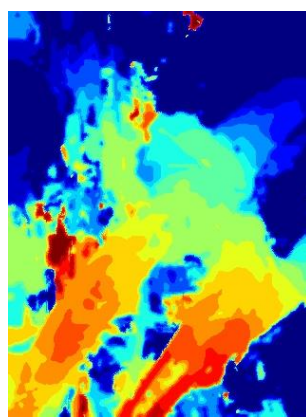
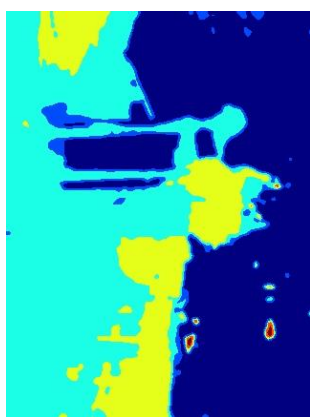
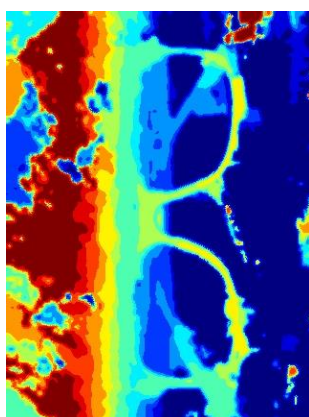
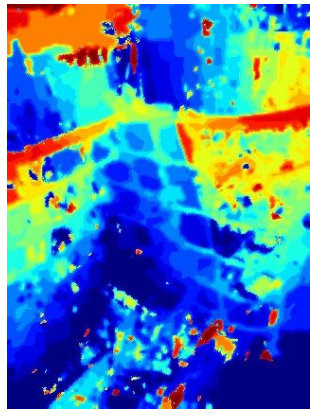
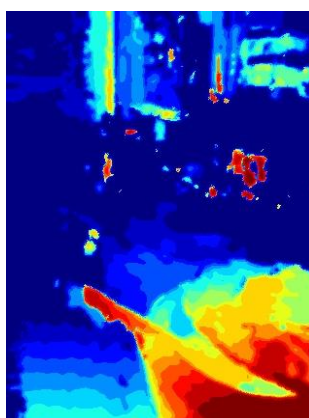
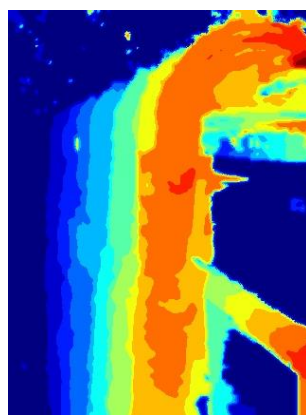
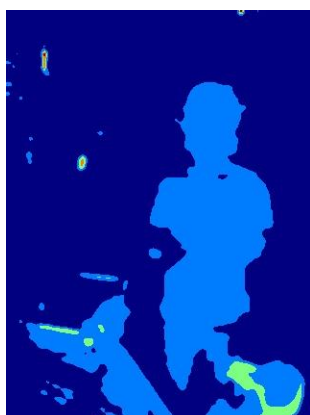
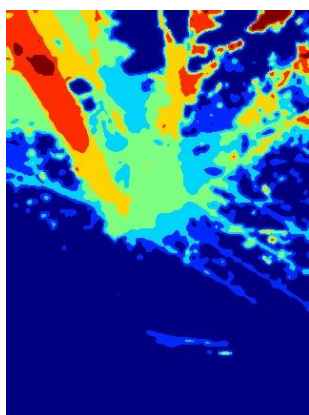
Average Error: 2.60

Image Number	0	1	2	3	4	5	6	7	8	9
Processing Time (seconds)	8.026	7.271	5.686	6.752	8.430	6.376	6.763	7.574	5.863	7.179
Error	1.875	2.393	1.631	2.904	3.555	3.103	2.200	2.655	1.714	2.976



Real Data

Image Number	0	1	2	3	4	5	6	7	8	9
Processing Time (seconds)	40.04	41.96	43.65	38.78	38.89	41.28	39.80	43.10	39.40	43.22



References

HEQ: <https://www.jianshu.com/p/9a9000d226b6>

SIFT: <https://segmentfault.com/a/1190000015735549>

BSM: <https://arxiv.org/pdf/1402.2020.pdf>

Cross-Based Stereo Matching:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.7579&rep=rep1&type=pdf>

More Robust Stereo Matching:

http://www.nlpr.ia.ac.cn/2011papers/gjhy/gh75.pdf?fbclid=IwAR2RfzesNLXFFMXMoS5df0IMN1A9Vpco1Sk_UroCH3Y5Ag-G8ltTs6EJUVY