國立清華大學 電機工程學系

實作專題研究成果報告

FPGA Implementation of Visual Odometry by Using High-Level Synthesis

使用 FPGA 及高階合成技術
應用於視覺里程計

專題領域：資工領域

組別：第 A288 組

指導教授：賴瑾(Jiin Lai) 教授

組員姓名：林奕杰、李承澔、鄧文瑜、林妤諠、郭朝恩

研究期間：112 年 6 月 30 日至 112 年 5 月 8 日止，共 12 個月

Abstract

The goal of this project is to utilize High-Level Synthesis (HLS) technology for hardware acceleration to increase development efficiency and simplify the design process. Compared to traditional hardware description languages such as Verilog and VHDL, HLS can quickly translate high-level programming languages such as C, C++, SystemC into RTL, and generate higher-performance hardware architectures through different optimizations, allowing designers to develop complex hardware systems more quickly.

In this project, we use Xilinx Vitis development system to rewrite algorithms in OpenCV and implement the Visual Odometry (VO) algorithm using HLS technology. We will use knowledge of computer vision, system design, hardware design, HLS development process, and HLS technology to achieve efficient real-time computation of the VO accelerator on FPGA.

摘要

本專題旨在運用高階合成(High-Level Synthesis，HLS)技術進行硬體加速，以提高開發效率並簡化設計流程。相對於傳統使用 Verilog、VHDL 等硬體描述語言進行開發，HLS 技術能夠將高階程式語言(C、C++、SystemC)快速轉譯為 RTL，並透過不同的優化產生更高效能的硬體架構，讓設計者能夠更快速地開發複雜的硬體系統。

在此專題中，我們使用 Xilinx 的 Vitis 開發系統，將 OpenCV 的演算法改寫，利用 HLS 技術實現視覺里程計（Visual Odometry，VO）演算法。我們將使用電腦視覺、系統設計、硬體設計、HLS 開發流程、HLS 技術等相關知識，以實現 VO 加速器於 FPGA 的高效即時運算。

章節目錄

一、前言

Visual Odometry 是一種通過分析相機捕捉到的連續圖像來估計相機運動的技術，其應用範圍涵蓋了機器人、自動駕駛、擴增實境等領域。而在 VO 演算法中包含深度估計 (Stereo Matching)、特徵點擷取 (Feature Extraction)、特徵點配對 (Feature Matching)、以及位移估測 (Motion Estimation)四個步驟。深度估計是指從一對相機捕捉到的圖像中，通過匹配兩者之間的對應點來估計相機的運動。特徵點擷取是指從圖像中提取特徵點，如角點、邊緣等，以用於後續的運動估計。特徵點配對是指在多個圖像或視頻幀中找到對應的特徵或關鍵點的過程，在計算機視覺應用中，常應用於物體識別、跟蹤和三維重建中。而位移估測則是通過使用先前提取的特徵點及其軌跡，從而估計相機的運動。

FPGA 具有可重構性、低延遲、高平行化等優點，因此被廣泛應用於許多領域，包括數位信號處理、計算機視覺等。而 High Level Synthesis 則是一種將高階程式碼轉換為硬體設計的技術，通過自動化設計流程來提高硬體設計的效率和品質。相對於傳統的硬體描述語言設計，High Level Synthesis 可以大大簡化設計流程。因此，本研究旨在運用 High Level Synthesis 技術將 Visual Odometry 演算法實現於 FPGA 上，加快視覺里程計在 FPGA 的硬體開發流程，未來更能提高其運算效率和實時性。

二、視覺里程計演算法

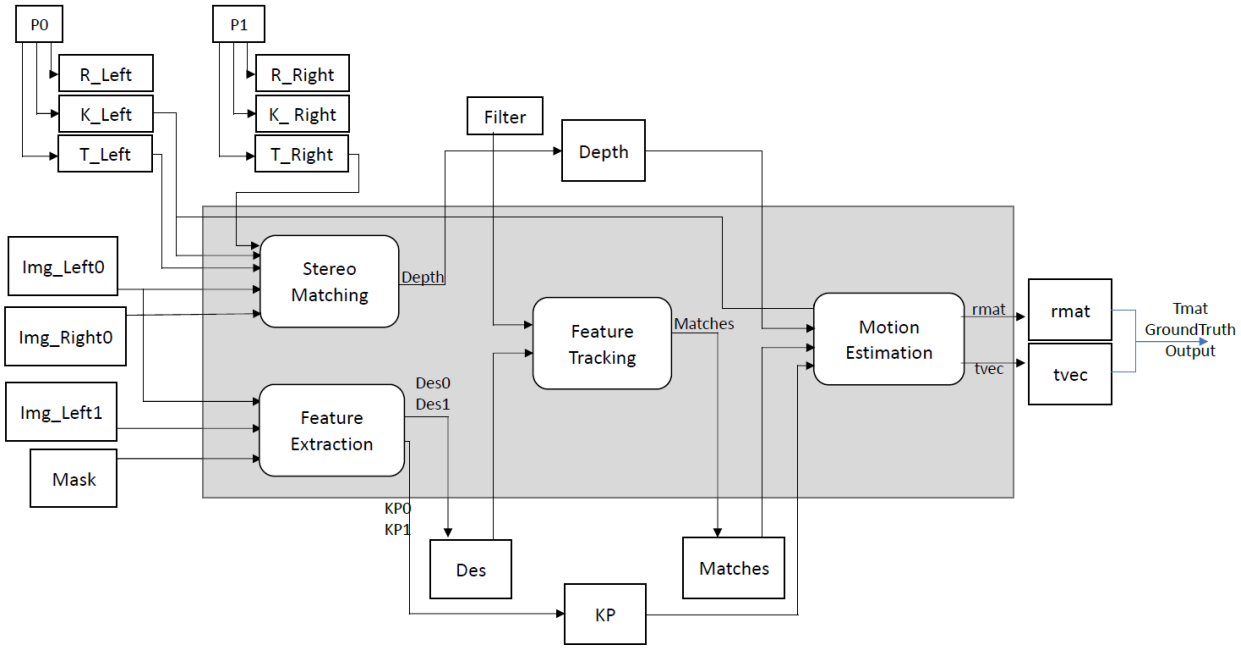

Figure 1 Visual Odometry Block Diagram

Visual Odometry 的目標是藉由位移前後擷取的兩張影像，估算出位移量，其中位移量以旋轉矩陣 R、平移向量 t 表示。如上圖示，由影像轉換到位移量的過程會經過深度估計 (Stereo Matching)、特徵點擷取 (Feature Extraction)、特徵點配對 (Feature Matching)、以及位移估測 (Motion Estimation)四個步驟。在每次位移的估計中，首先需要輸入位移前左、右兩部相機擷取的圖片 (image left 0、image right 0)，配合相機參數 (P0、P1)，用左右視差計算出深度 (depth)。同時分別對左相機位移前後擷取的圖片 (image left 0、image left 1)提取特徵點 (kp0、kp1)以及其描述子 (des0、des1)。在提取特徵點後，給定閥值 (filter)利用描述子配對特徵點 (matches)。最後利用深度、配對的特徵點、相機參數，估計出旋轉矩陣 (rmat) 以及平移向量 (tvec)。

2.1 Stereo Matching

SGBM (Semi-Global Block Matching)演算法是一種用於立體視覺和深度估計的演算法，它可以在左右相機圖像之間進行視差計算，用於計算左右相機圖像之間的位移。SGBM 演算法使用窗口匹配的方法，在左右圖像中匹配特定大小的區塊。它計算每個區塊在另一個圖像中的對應區塊的相似度，以確定每個像素的視差值。它通過考慮每個像素的整個代價圖，並將其與經過平滑處理的代價圖進行比較，以選擇最佳匹配。

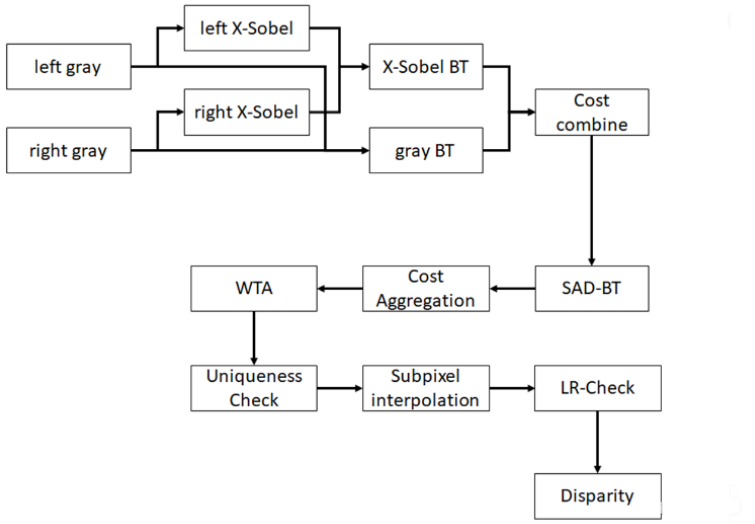

Figure 2 Stereo Matching Block Diagram

1. **輸入/輸出**

   輸入 ： 將左右兩張圖片以及相機的矩陣參數 k、r、t 讀進來

   輸出 ： 經過計算後輸出深度

2. **匹配代價計算 (matching cost computation)**

   (1) 輸入圖像經過 X-Sobel 處理後，計算 BT 代價。

   (2) 輸入圖像直接計算 BT 代價值。

   (3) 將上面兩步的代價值進行融合。

   (4) 對上述步驟得到的代價值進行成塊處理，在 SGBM 演算法中，成塊計算就是對每個圖元的代價值用周圍鄰域代價值的總和來代替。

3. 代價聚合 (cost aggregation)

$$L_r(p,d) = c(p,d) + min\begin{cases} L_r(p-r,d) \\ L_r(p-r,d\pm 1) + P_1 \\ min\ L_r(p-r,i) + P_2 \end{cases} - minL_r(p-r,i) \qquad s(p,d) = \sum_r L_r(p,d)$$

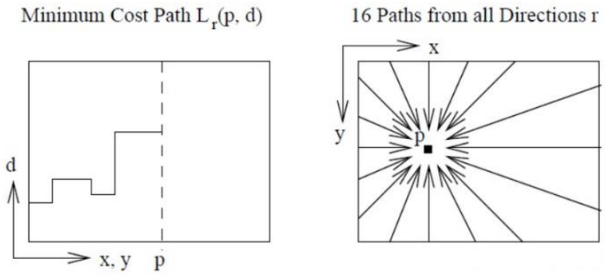

Figure 3 Cost aggregation

L 為當前路徑累積的代價函數，P1、P2 為圖元點與相鄰點視差存在較小和較大差異情況下的平滑懲罰(P1<P2)，第三項是為了消除各個方向路徑長度不同造成的影響。將所有 r 方向的匹配代價相加得到總的匹配代價(代價聚合)。

4. 視差計算或優化 (disparity computation/optimization)

(1) SGBM 演算法中關於視差的選擇，通常選擇使得聚合代價最小的視差。

(2) 每個像素的視差選擇都只是簡單通過 WTA（Winner Takes All）決定。

(3) 最後進行視差改良(disparity refinement)後處理。

2.2 Feature Extraction

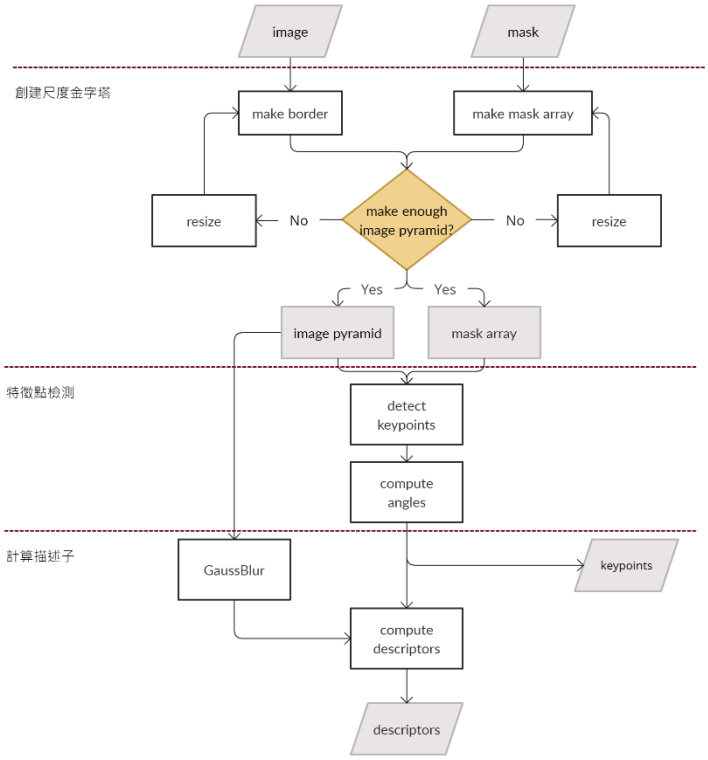

Figure 4 Feature Extraction Block Diagram

1. **輸入/輸出**

   輸入：要計算的 image 與給定的 mask。

   輸出：偵測到的 keypoints 與計算出的 descriptors。

2. **創建尺度金字塔**

   將原始圖片進行不同尺度的縮放，以確保特徵點檢測的尺度不變性，也就是在不同尺度下的檢測結果相同。我們採用 8 層尺度金字塔。

3. **特徵點檢測**

   採用 oriented FAST (oFAST)的檢測方式，oFAST 結合了 FAST 與 Harris 算法。

   (1) FAST 算法檢測特徵點的方式為當一像素其周圍圓上有連續 n 個像素的灰度值大於或小於給定的臨界值，此像素即為特徵點。oFAST 採用半徑為 8 的圓、連續 9 個像素，且臨界值為 20。

   (2) Harris 算法為偵測圖像中的角點，角點是兩條邊緣的交點，因此檢測小圖像片段中灰度值的變化程度，即可判斷特徵點。

   (3) 最後將每個特徵點加入角度，以確保特徵點的旋轉不變性，也就是在不同方向下的檢測結果相同。角度的計算為灰度質質心位置到中央與 x 軸所產生的夾角。

4. **計算描述子(Descriptor)**

   (1) 將圖片做高斯模糊，使得每一鄰近像素之間的落差不會太大，以確保描述子計算的正確性。

   (2) 以特徵點為中心，在 31*31 的圓內選取 256 對點做比較，每一對點會得到 1 與 0 的結果，此 256 個 bit 即為描述子。

2.3 Feature Matching

1. **輸入/輸出**

   輸入 ： 由 Feature Extraction 得到的兩張圖片的描述子

   輸出 ： 兩組特徵點(描述子)配對後的索引值

2. **knn match**

   輸入兩組特徵點的描述子，找出第二組描述子中，與第一組描述子第一及第二相近的。由於 Orb 使用的描述子為二元描述子(binary descriptor)，因此將使用 hamming distance 計算相似度。

3. **ratio test**

   給定閥值，若 (第一相近的描述子 >= 第二相近的描述子×閥值)，則意味第一相近的描述子足夠特別成為配對的描述子。

2.4 Motion Estimation

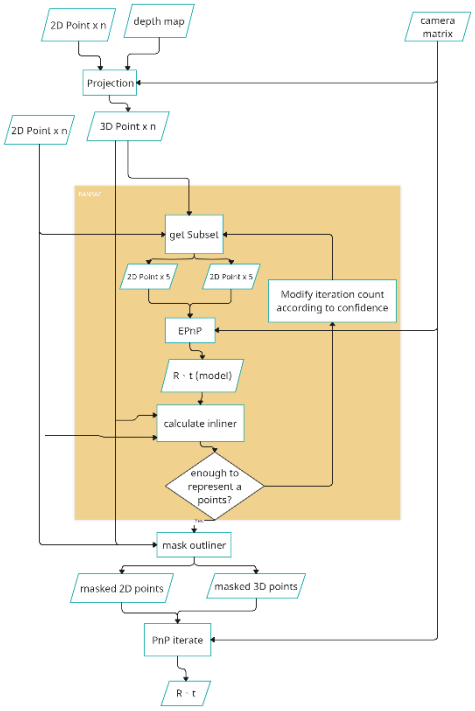

Figure 5 Motion Estimation Block Diagram

1. **輸入/輸出**

   輸入 ：stereo matching 輸出的深度、feature extraction 輸出的兩組特徵點、feature matching 輸出的特徵點配對、以及相機參數。

   輸出 ： 旋轉矩陣 R、平移向量 t

2. **特徵點投影**

   進入 EPnP 前，先配對兩組特徵點，接著藉由深度、相機參數，將兩特徵點分別投影成 2D 和 3D 點。

3. **RANSAC-PnP**

   (1) 隨機提取投影點子集合(subset of 2D, 3D points)，分別 5 個點，稍後用於 EPnP。

   (2) 用提取的投影點做 EPnP 計算(Efficient Perspective-n-Point Camera Pose Estimation)。EPnP 將線性計算 3D 點和 2D 點之間的幾何誤差，並使用 Gauss-Newton 迭代優化係數，最後輸出兩特徵點場景的旋轉矩陣(R)和平移向量(t)。

   (3) 將使用投影點子集合算出的 R 和 t，代回將全部的 2D 點投影成 3D 點，並計算與原始 3D 點間的誤差，將超出 threshold 之投影點定為 outliers。

   (4) 重複步驟，直到用 R 和 t 算出誤差的低於給定的值。

   (5) 迭代完成後，將投影點扣除最終決定的 outlier(mask with outlier)。

4. **PnP iterative**

   以 mask 過的投影點作為輸入，與 EPnP 類似，用線性方法(Direct Linear Transform)，算出 R 和 t，並使用 Levenberg-Marquardt method 迭代優化誤差值，最後輸出旋轉矩陣(R)和平移向量(t)。

三、開發流程

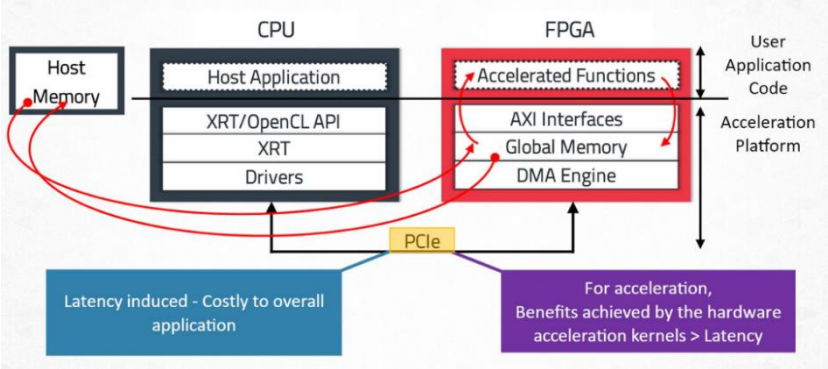

Figure 6 Host-Device Architecture

1. HLS 應用加速程式由兩個部分組成：

(1) 軟體程式以 C/C++編寫，執行於傳統 CPU（x86 或 ARM）上。其次，軟體程式使用由 Xilinx Run Time（XRT）或 PYNQ 函式庫實現的 user-space API 與 FPGA 中的 kernel 進行資料傳輸。

(2) 硬體加速 kernel 可以用 C/C++或 RTL（Verilog 或 VHDL）撰寫，並在 FPGA 上執行。這些 kernel 使用標準的 AXI interface 與 Vitis 硬體平台集成。通過 AXI-PCIe IP，PCI 加速卡使用 PCIe 總線與 x86 服務器傳輸資料。

在本次的專題中，我們使用 KITTI Visual Odometry 作為 Code Base，並以此為參考將其呼叫的 OpenCV API 轉為 self-contained library，自行改寫內部程式並測試，接著再改寫為 High-Level Synthesis code，使用 Xilinx Vitis 系列軟體做軟體及硬體驗證，最後在 u50 上執行。[4]

2. Work flow：

(1) 使用 Visual Studio 運行：

i. 在使用 OpenCV library 的環境下運行 Visual Odometry 演算法，並以 KITTI dataset 作為測資。

ii. 區分 Host 與 Kernel 各自負責的部分。

iii. 從 OpenCV library 中找到所有演算法的 C++ source code，將所有會使用到的 function 與 class 整理出來成為 self-contained code，各 Kernel 即不再使用到 OpenCV library。

(2) 使用 Vitis HLS 或 Vitis IDE 運行：

i. 運行 C-simulation，Host 與各 Kernel 各自利用測資驗證正確性，Host 使用 KITTI dataset；各 Kernel 從 OpenCV 環境下輸出自己會用到的測資來使用。

ii. 運行 Synthesis，各 Kernel 改寫成 HLS 可以合成為硬體的形式，並閱讀 Synthesis Report 進行初步優化。

iii. 運行 Co-simulation，確保所有介面可以正常串接且模擬硬體運算結果正確。

(3) Host 採用 OpenCL 架構：

i. Host code 使用 OpenCL 撰寫，並確認 Host 與各 Kernel 間傳輸的資料型態及介面。

ii. Host 程式除了能完整串聯所有 Kernels，也需允許各 Kernel 個別驗證。

iii. Host 程式能使各 Kernel 運行在兩種模式下：C code 模式或 FPGA kernel 模式。

(4) Host 與 Kernels 進行驗證：

i. Host 先個別驗證各 Kernel，且一次僅有其中一 Kernel 跑在 FPGA 上，其餘部分皆先跑在 CPU 上。

ii. Host 整合驗證各 Kernel，將所有 Kernel 串聯且都跑在 FPGA 上。

iii. 依序跑 SW-Emulation -> HW-Emulation -> Hardware，驗證每個步驟皆正確執行。

(5) Host 與 Kernels 進行優化：

確認資源使用量及比較時間(FPGA time v.s CPU time)，並進行優化。

四、 實作內容

4.1 Host Program

1. 維護基於 OpenCV Library 的 Visual Odometry program，後續所有的硬體行為及輸出皆會與此 program(下稱 OpenCV Host Program)的輸出比較。
2. 且基於 OpenCV 新加入的功能皆會在此 OpenCV program 先行測試，才會加入下列的 Host program，以限縮除錯時的範圍。
3. 開發整合各 Kernel pure(self-contain) C Visual Odometry code 的 Host program(下稱 pure C Host program)。並分別在 Visual Studio 與 Vitis IDE 進行兩階段的 C-Simulation，驗證兩 IDE 編譯器的行為是否一致。
4. 雖然稱作 pure C，但 pure C Host program 運行在 PC/Server 上，仍然可以調用 OpenCV Library 進行讀圖、圖像縮放、顯示圖片等非加速目標之功能。
5. 開發整合各 Kernel HLS Visual Odometry code 的 Host program(下稱 HLS Host program)。此 Host program 基於 pure C Host program 開發，加入 OpenCL 溝通 FPGA kernels。

   為了方便驗證我們制定並以 define macro 方式選擇以下 6 種編譯模式：

```
//==================================Macro Section=====================================
/**
 * 6 compile modes:
 * `_PURE_C_`                  // Run all function in pure C code
 * `_ONLY_K_StereoMatching_`   // Run all function in pure C code excluding `StereoMat
 * `_ONLY_K_FeatureExtraction_` // Run all function in pure C code excluding `FeatureEx
 * `_ONLY_K_FeatureMatching_`  // Run all function in pure C code excluding `FeatureMa
 * `_ONLY_K_MotionEstimation_` // Run all function in pure C code excluding `MotionEst
 * `_ALL_KERNELS_`             // Run all function in HLS code
 */
//================================Macro Section END===================================
```

Figure 7 Compilation Mode

- **_PURE_C_Host 的行為與 pure C Host program 相同，所有 function 都運行在 Host PS side。**

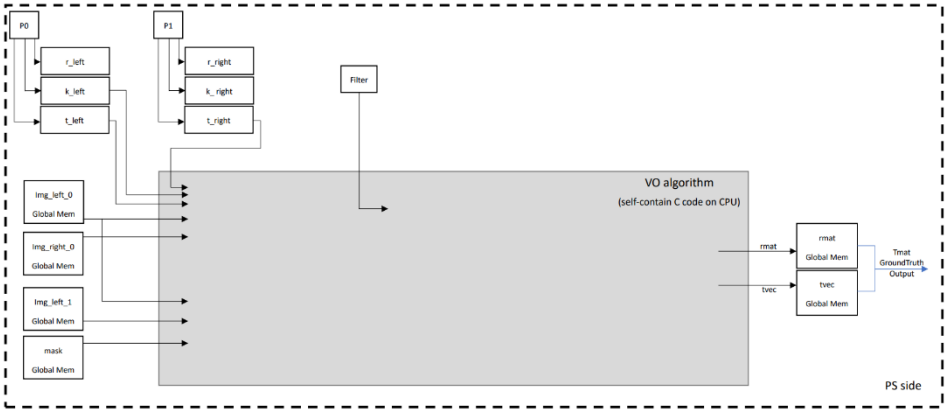

Figure 8　Pure C Host Program

- **_ONLY_K_<Kernel_Name>_則會以 Xilinx v++ compiler 編譯、OpenCL 溝通該 kernel，於 FPGA PL side 運行。其餘的 kernel function 仍是在 Host PS side 以 pure C program 運算。**

  一旦建立單個 kernel 的 compile mode，各個 kernel 的開發可更具有獨立性。kernel 開發者可基於先前 pure C Host program 的前級輸出，作為後級 kernel 的輸入，以獨立開發各自 kernel。(例如開發 Feature Matching kernel 時，不必等待前級輸入 Feature Extraction kernel 開發完成。可同時開發。)且 kernel 的驗證也是與 pure C Host program 的結果比對，4 個 kernel 可同時驗證，大幅縮短開發流程。

- **_ALL_KERNELS_則是以 Xilinx v++ compiler 編譯、OpenCL 溝通 4 個 kernel，於 FPGA side 運行。**

  在先前單個 kernel 驗證完畢後，只要在 Host 端正確連接前後級對應的輸入輸出，即可在不修改 kernel code 的情況下同時運行 4 個 kernel，達成所有演算法都透過 FPGA 加速的目標。且 kernel 開發者不必擔心在不同的運行模式下介面定義不相同。

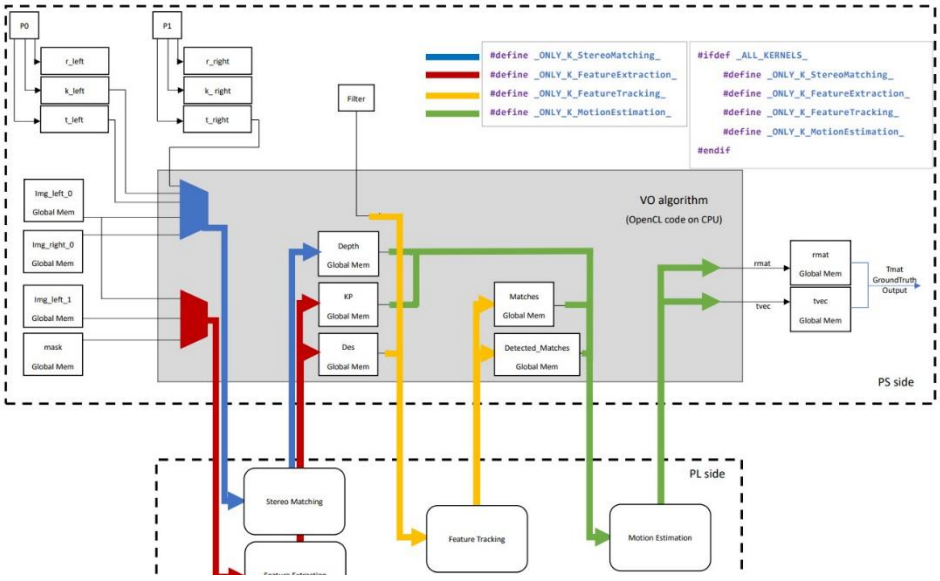

Figure 9 Entire Block Diagram

4.2 Kernels

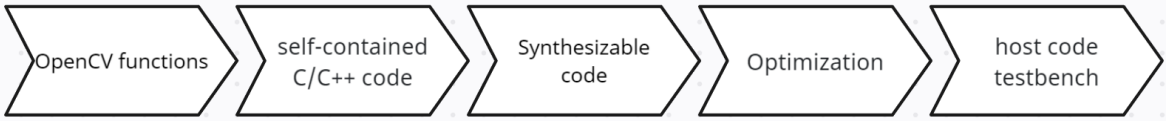

Figure 10 Kernel Development Flow

對於四個 kernel 的設計流程，從 OpenCV 實作四種演算法所呼叫的 function 著手，首先將內部使用的無論數學運算、資料儲存結構，節錄及實作成完全沒有呼叫外部函式庫的 self-contained code。順利實作完成後，HLS 編譯器會出現不支援部分程式碼的情形，kernel 介面也沒有硬體定義，因此要針對可合成性修改程式，成為 synthesizable code。

接著，儘管程式以符合硬體合成需求，還是可能出現資源使用過多、運行時間過長等等的問題，此時便要研讀各項 report，針對問題做優化。當優化完成並通過模擬測試後，最後即可輸出成 binary file，交付 host program 完成應用端測試。

4.2.1 Pure (self-contained) C/C++ code

使用 OpenCV 實作演算法時，為了使用方便，通常只會呼叫一、兩個頂層的函式，然而這些函式背後會呼叫大量其他的底層函式。此外，也會出現為了防止 exception 出現而設計的檢查性程式，但硬體的規格限制不需這些檢查。因此這步驟的目的是將實際實作演算法的程式碼擷取出，同時使擷取出的程式碼不會呼叫外部函式庫(OpenCV library)，並驗證演算法輸出正確。

- 運行環境 : Visual Studio
- Steps:
  1. 從 OpenCV 函式庫中找到並清理出會使用到的 class 與 function，由於程式碼眾多且定義複雜，過程較為費時。若針對真正會使用的部分清出，可以避免最終要處理大量不相關的程式碼，過程中需確定輸出結果正確。
  2. 清理程式碼時，遇到龐大定義的 class 與 function，直接將整個 class 與 function 搬來使用並不明智，可以自行改寫為所需的資料型態或運算方式。
  3. 製作屬於自己 kernel 的 testbench，實作方式為利用 OpenCV code 跑一張圖片，將所需數據寫入 txt 檔作為 testbench。

4.2.2 Synthesis

當手上已經有了純粹實作演算法、不依賴外部函式庫的程式碼，這還是不足以將程式碼直接合成為硬體，因為程式中出現不可合成為硬體的行為，例如使用動態配置的資料結構、system call、向量計算或對指標的運算，也有可能是 kernel 介面無定義清楚，因此需花蠻多的時間在改寫程式內容，以下將探討部分原因。此步驟目標是將手中的程式碼修改為可合成為硬體的形式，並驗證正確性，再通過 Vitis HLS 的 synthesis 並產生 report，接著根據 report 的資訊加上適當的 #pragma 做優化。

- 運行環境 : Vitis HLS
- Steps:
  1. 在 top function 加入 interface pragma，並確認 buffer depth。在介面上若有多

筆資料要傳輸可以加上不同 bundle，使資料能夠同時傳輸以減少傳輸時間。

2. FPGA 不能動態配置記憶體大小，因此在硬體實作上所有 array 的大小都要事先給定，並改寫動態分配記憶體的資料型態，如：std::vector。
3. 過程中會遇到許多不能合成的情況需做改寫，如：Pointer to pointer，或是某些程式碼無法對應硬體行為。
4. 確認能夠合成之後，就可以修改程式碼或是加上#pragma 以進行優化，例如：pipeline, unroll, array partition 等等，且需注意#pragma 有確實生效。
5. 由於 report 產生的資源使用量有時無法真實反映使用情形，因此放到 FPGA 進行驗證時，可能還是會因為資源使用超過而無法成功驗證，此時就要再進行優化以減少資源使用。

4.2.3 Co-sim

儘管手邊的程式碼已經可以順利合成為硬體，其執行結果可能會與預期結果不符，這是因為硬體執行時，並不會像 CPU 依序執行指令，而會平行運行，這可能導致資料間的相依性與運行在 CPU 上的程式不同。此外，各函式間的接口也會被賦予硬體特性，例如 stream 接口便會有容量，若操作不當，可能導致 deadlock，使程式 stall 住。因此這步驟的目的是驗證合成出的硬體經過實際的 routing 後，運行結果是否與使用 CPU 執行時相同，並確保當部屬到板子上時，不會出現錯誤。

- Steps:
  1. 由於 Co-sim 的時間相當長，因此各 kernel 可以設計出使用較少的運算資料來進行驗證的方式，或是將程式分段測試，以快速找到問題所在。
  2. 根據 Co-sim 後產生的 timeline trace、schedule viewer 以及 waveform，可以確認演算法內部運作的流程是否符合預期。
  3. 當硬體模擬結果與軟體模擬結果不同時，以下提供可能出錯的原因:
     (1) 暫存器未歸零(尤其是有做 +=、*= 等的運算)，將使下次使用相同暫存器時，內容與預期不符。
     (2) 合成硬體時，編譯器會對程式做平行化的優化，此時若編譯器不慎將有 data dependency 的運算做平行化，也會使得運算結果與預期不符。
     (3) 內部演算法 buffer 容量開得太小，使計算時超出記憶體空間，會發生運行時間跑不完的情況。

4.2.4 on FPGA

當手上的程式通過層層驗證，便可以輸出成 binary file (xclbin)，透過自己寫的 OpenCL host code testbench 將各個 kernel 部屬到 FPGA 上串接起來，並由執行出的 report 評估是否繼續後續優化，最後再交由 host program 端做後續應用的部屬。

- 運行環境 : Vitis IDE
- Steps:
  1. 確保硬體資源不會超過 FPGA 板上可使用的量，如超過的話在跑 Hardware-link 時會出現錯誤，需回到前面步驟修改程式以解決資源使用過多的問題。
  2. 產生 xclbin，根據實際在 FPGA 上運行的時間，與 CPU 對比，並持續優化。

五、 結果

5.1 測試資料集

在我們的實驗中使用 KITTI dataset，KITTI dataset 是一個被廣泛使用的計算機視覺資料集，其中包含了從一個行駛中的車輛上拍攝的影像序列、雷達數據、慣性測量單元 (IMU) 等資料。它主要用於研究和評估各種計算機視覺任務，如視覺里程計 (visual odometry)、立體視覺 (stereo vision)、物體偵測與辨識 (object detection and recognition) 等。

- Poses

這個資料夾中的檔案包含了每一幀畫面的的 6D 相機參數，其紀錄相機與起始點的相對旋轉與平移。

- Sequences

其包含 22 組雙目相機的 png 檔案作為訓練資料，其中 11 組資料(00-10)有 Ground truth，而剩餘的 11 組(11-21)無 Ground truth。

Figure 11 Dataset

5.2 實驗環境

CPU	11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz
RAM	49112592 kB
OS	Ubuntu 20.04.4 LTS
FPGA	Xilinx u50

Figure 12 Environment

5.3 成果比較

5.3.1 測試方法

由 Pure C program 及 HLS program 分別輸出 2000 張圖像的軌跡(trajectory)，各自與真實路況軌跡(ground truth)比對，並比對 Pure C program 和 HLS program 中演算法的實際行為是否相符，或是釐清誤差的原因。

由於部分演算法使用過多 BRAM，導致無法將 4 個演算法合成 binary file 同時

燒錄進 FPGA 上。因此 HLS program 的驗證改採存讀檔、分別運行 4 個 kernel 的方式，將前一級 kernel 算完 2000 張圖像對應的輸出存檔至 mass storage，接著下一級 kernel 由 mass storage 讀取前級結果作為本次輸入，直至 4 個 kernel 分別運行完畢。同時比較 Pure C program(on CPU)及 HLS program(on FPGA)的 10 張圖像平均運行時間。

5.3.2 10 組圖片的平均運行時間比較

kernel	FPGA time (Baseline Version)	FPGA time (Optimized Version in 2023/5/7)	CPU time
Stereo Matching	8290.4 ms (216.3MHz)	7880 ms (190.3MHz)	439.3 ms
Feature Extraction	910.3 ms (300MHz)	321.6 ms (104.7MHz)	58.6 ms
Feature Matching	11.6 ms (300MHz)	5.13 ms (300MHz)	4 ms
Motion Estimation	60.6 ms (300MHz)	408.6 ms (258.1MHz)	2.5 ms

Figure 13 Compare CPU time and FPGA time

5.3.3 2000 組圖片對應的真實路徑 (Ground truth)

在 KITTI Dataset 中，XZ 平面解釋為生活當中的 2D 水平面(長度、寬度/前後、左右)；Y 軸則是解釋為海拔高度。由於此資料集是在相對平整的平面路段採集而成，無過多不規則的高低起伏，因此在後續的結果分析應主要聚焦於 XZ-plane。

5.3.4 結果

以下分別為 **Ground truth (**Blue**)** v.s **Pure C Program** (Orange) v.s **HLS kernel Baseline (**Red**)** v.s **HLS kernel Optimized (**Green**)** 跑出來的路徑圖：

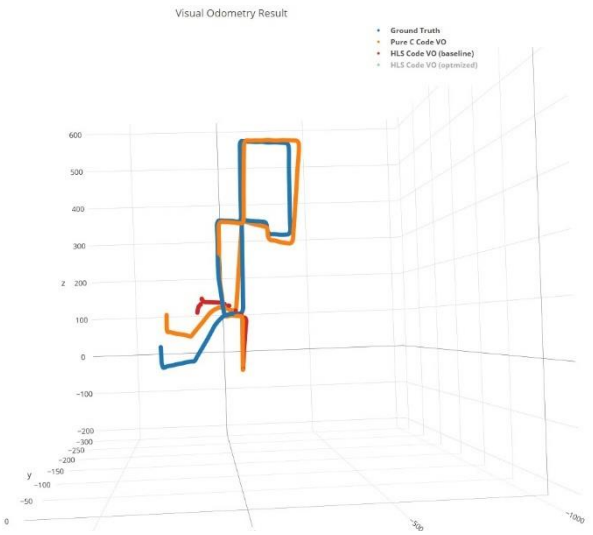

Figure 14 Trajectory Comparison (Baseline)

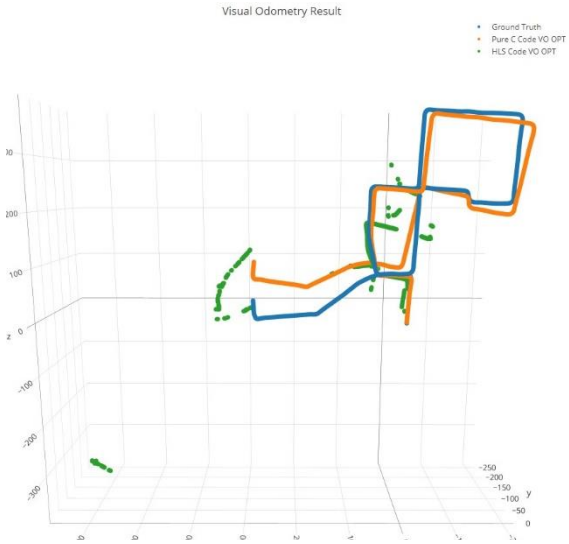

Figure 15 Trajectory Comparison (Optimized)

5.4 結果分析

5.4.1 Ground truth 與 Pure C program 的結果誤差

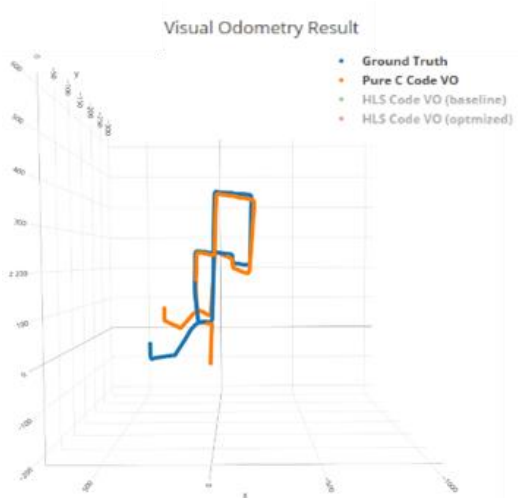

Figure 16 Track discontinuity and matching segment between ground truth and pure c program

Figure 16 中 Pure C program 計算出的軌跡與真實路徑(Ground Truth)間存在些微偏轉，此為 VO 演算法中已知難以避免的問題，因此後續應著重比較 Kernel 與 Pure C program 計算出的軌跡。

5.4.2 Pure C program 與 HLS kernel (baseline)的結果誤差

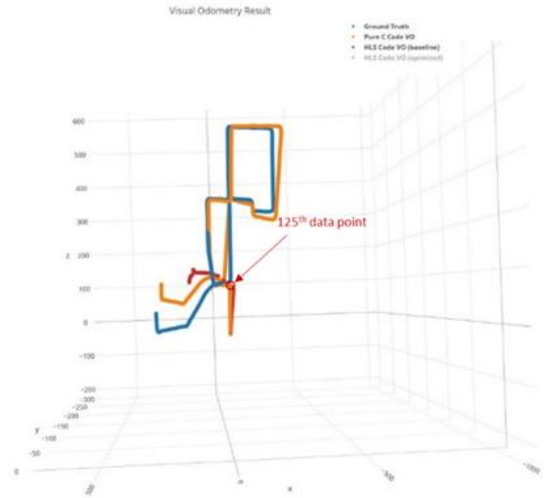

Figure 17　Track discontinuity and matching segment between HLS program and pure c program

由 Figure 17 可知，baseline kernel 在約前 125 個數據點與 Pure C program 的結果相近，接著開始轉彎數據點卻出現不連續、移動幅度過大、行進方向偏差的情況。

開發此階段 Kernel 時，Pure C program 中為了因應不同路況變化而有對應的軌跡估計演算法，但若移植所有情境下的演算法至 FPGA 將導致資源使用率過高，因此選擇捨棄部分算法。

5.4.3 Pure C program 與 HLS kernel (optimized)的結果誤差

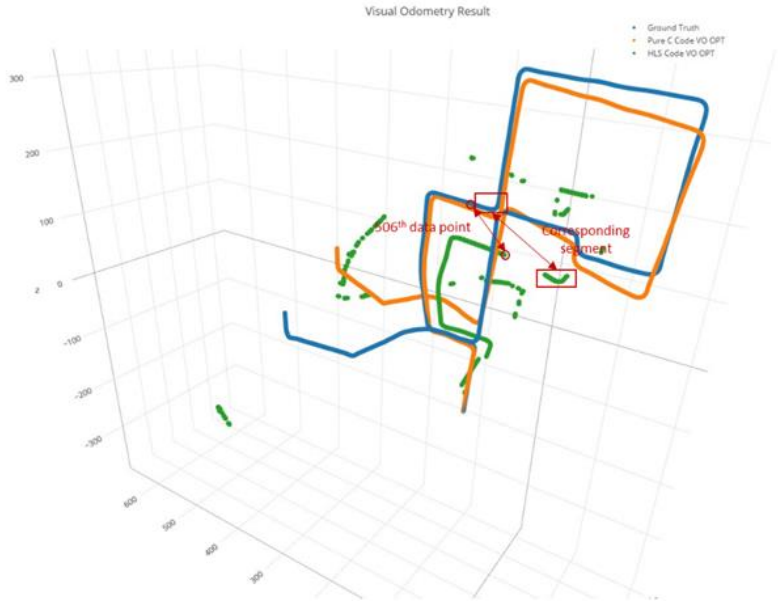

Figure 18 track discontinuity between optimized HLS program and pure c program

在本階段 kernel 開發階段時重新加入部分演算法以因應轉彎的路況變化。

由 Figure 18 可看出，optimized kernel 在約前 506 個數據點軌跡與 Pure C program 的軌跡相似。接著即因誤差累積過大，後續數據點開始出現不可辨識的軌跡與斷點。可估計的路徑長度較前一階段 baseline kernel 多出近 400 個 iteraions。

5.5 結果討論

5.5.1 誤差的原因

(1) 誤差累積

由前兩節比較發現在直線行進時，我們的演算法可以算出不錯的擬合路徑(Pure C program 及 HLS program 在前 125 數據點的結果)，直到轉彎時才產生錯誤。由於 VO 具有誤差累積性：

$$e_i = {}^{i}T_{i+1} \cdot {}^{i}\hat{T}_{i+1}^{-1}$$

每一 iteration 的誤差e_i都是第(i, i+1)次的 ground truth 轉換矩陣$i_{T_{i+1}}$與 VO 計算出的逆轉換矩陣$i_{\hat{T}_{i+1}^{-1}}$內積。意即一旦誤差發生，只要 VO 計算的路徑越長，最後的誤差也會因累積而隨之增大。因此在第一次轉彎出現極大的誤差後，後續的路徑即以相當可觀的程度偏離。

(2) 斷點形成的原因

從圖中可見路徑出現斷點，其成因主要來自於 Motion Estimation kernel 中使用的 Levenberg-Marquardt (LM)迭代優化演算法。與梯度下降法(gradient descent) 、高斯牛頓法(Gaussian-Newton)類似，LM 作為迭代優化演算法，自然會面臨陷在 local minima 的問題，接著便可預見，計算出的結果超出預期許多，這便是出現斷點的主因。

陷入 local minima 的原因與迭代的起始點息息相關，在優化過的 Motion Estimation kernel 中，LM 的起始點來自以下三處:

i. inlier 點經過直接線性變換法(Direct Linear Transform, DLT)，計算出之 rmat、tvec。
ii. 前一級 RansacPnP 中，由 EPnP 計算出之 rmat、tvec。
iii. 上一輪 Motion Estimation kernel 計算出之 rmat、tvec。

這三組 rmat、tvec，經投影後計算誤差，選取誤差最小作為 LM 迭代優化的起始點。

然而以上任何一組結果作為初起始點仍有可能使結果陷入 local minima:

i. 作為線性解法得出之 rmat、tvec，DLT 的結果自然不適合用於轉彎處。
ii. EPnP 為隨機抽樣 5 點計算，雖然經過 Ransac 篩選，但仍不可靠。
iii. 上一輪 Motion Estimation kernel 計算出之 rmat、tvec 在連續動作也許效果很好，但如果遇到急轉彎等情況，還是不適合作為起始點。

綜合以上所述，在不改變 LM 演算法的情況下，當遇到動作變動劇烈時(例如轉角)，便可能發生斷點。

(3) 未來優化方式

我們觀察到 ground truth 一開始並不是走 z 直線的方向，因此未來可以在系統的最後加上校正的實作，我們覺得可以採用 Bundle Adjustment 的技術修正，目標是通過優化相機參數和特徵點的三維位置，從而使重建的三維模型與實際場景盡量一致，達到更精確的三維重建效果，以最小化在所有影像中的重投影誤差。

5.5.2 FPGA 速度較 CPU 慢的原因

(1) Common Reason:

- **未做到最高頻率運算：**

目前頻率未能做到最快的 300MHz，觀察 report 可以發現還有許多造成 timing violation 的運算。

Future work: 將所有無法達到高頻的運算修改以達成 300MHz。

- **讀取資料未完整做到 burst：**

在 kernel interface 部分，並未對 m_axi 做 max burst length、number of outstanding 的測試，使得僅僅在讀取資料的表現，kernel 就輸給有做完整 caching 的 CPU。

- **未優化完全及未高度平行化運算：**

目前尚無法使整個演算法以 dataflow 的形式計算，且有些運算還可以再修改成能夠做到 pipeline 及 array partition 的形式，使運算高度平行化。

Future work: 改寫變數型態與運算方式以達成更有效的運算。

(2) Stereo Matching

- **尚無實作出 cache 的分配資源寫法：**

由於內部演算法計算複雜，且有許多層迴圈的結構，使用 cache 需確認每個 buffer 的實際使用情形，且需避免 random access 的情況，因此在 cache 的實作上

有一定的困難度；且在有時間壓力的情況下，目前並未採取此解法改寫。

Future work: 使用聰明的資源分配技巧將資料 cache 起來，以解決資源不足所導致的成果誤差並縮短運行時間。

- **切圖運算需呼叫多次 (FPGA 資源限制)：**

由於 FPGA on-chip buffer 資源不足，因此我們將圖片切成三等分，並在內部 kernel 呼叫了五次該演算法，最後再進行疊圖的運算以縮小誤差。因此在時間上比起原先只呼叫一次演算法、且不用進行後面疊圖計算的時間還慢上許多。

Future work: 優化演算法以縮小 buffer 的使用，達到不需切圖與疊圖的運算。

(3) Feature Extraction

- **未符合硬體適合的運算形式：**

由於從 C++的程式碼做修改，原先的運算較符合 CPU 的執行效率。因有許多運算可以做 Divide and Conquer 並修改成能發揮硬體優勢的運算方式，而我目前所做的修改未能達成最好的硬體運算效率。

Future work: 重新改寫成硬體能高效運算的程式。

- **FPGA 資源限制：**

原先應可以在偵測 keypoint 時同時做 GaussBlur 的運算，然而存放圖片的 BRAM 有限，導致無法同時進行運算。

Future work: 優化資源使用量並改寫成並行運算。

(4) Motion Estimation

- **新增演算法：**

除了實作原始 OpenCV 中的演算法，也額外增加演算法以優化運算結果，例如：比較許多組先前算的結果做誤差計算，因此新增的計算步驟導致了整體的運行時間增加。

Future work: 優化演算法以降低運行時間。

六、結論

本專題除了要清楚軟體層面的演算法，更需要去細究其對應到的硬體行為，盡量消滅資料間的 Dependency 以及盡可能利用 FPGA 高平行度的優勢。此外，在 Memory 的使用上需要格外小心，避免定義過大的 Buffer，以避免合成出超過實際可使用的硬體資源，導致合成出的硬體無法使用。

本次實作成功將 VO 演算法應用於 FPGA 上進行運算。不斷迭代優化後，除 Motion Estimation 為增加路徑估計精準度而增加用時，已大幅縮短整體 FPGA 的運行時間。儘管其運行時間仍不及 CPU，但與最初版的運行時間相比已有顯著進步。

由於 self-contained C code 計算出的軌跡與真實路徑(Ground Truth)間的偏差實屬 VO 演算法已知難以避免的問題。因此我們著重比較 Kernel 與 self-contained C Program 計算出的軌跡，可發現前 506 個 iterations 中，兩者具有相似的軌跡，說明我們的系統在較短的路徑內可以做出相對準確的軌跡估計。

在此次專題實作中，即便硬體資源與 CPU 平台相比較為貧乏許多，我們仍成功在 FPGA 上開發出整套系統。我們學習到如何應用 HLS 技術於系統的設計開發、驗證硬體行為，並找到可能導致結果不如預期的原因，未來將進一步持續平行優化。

七、參考文獻

[1] T. Gao et al., *"I ELAS: An ELAS-Based Energy-Efficient Accelerator for Real-Time Stereo Matching on FPGA Platform,"* 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)

[2] W. Fang, Y. Zhang, B. Yu and S. Liu, *"FPGA-based ORB feature extraction for real-time visual SLAM,"* 2017 International Conference on Field Programmable Technology (ICFPT) https://ieeexplore.ieee.org/document/8280159

[3] Maio, A. D., & Lacroix, Simon. (n.d.). "*On Learning Visual Odometry Errors*.", GitHub RoboticVisionChallenges.,https://nikosuenderhauf.github.io/roboticvisionchallenges/assets/papers/IROS19/deMaio.pdf

[4] "*KITTI_visual_odometry*", https://github.com/FoamoftheSea/KITTI_visual_odometry

八、專案管理及組內分工

1. 專案管理

我們將各自負責實作的 source code 儲存於 GitHub，作為團隊協作的平台。我們使用 Excel 定義了各核心模塊的 I/O interface，以便更好地進行模塊之間的協作。同時，我們使用 Slack 和 Google Meet 作為與教授溝通以及進行每周會議的平台，以確保團隊成員之間的有效溝通和協作。撰寫報告則採用 HackMD，以共同編輯與討論。

以下是我們的 GitHub 連結：https://github.com/bol-edu/robotics-computing
我們將所有程式碼、燒錄的 binary code、開發流程、相關實作檔案皆放置在上述平台，包含整個實作專題期間的 Issues and Fix 等等。

2. 組內分工

這次的專題小組中，除了包含本次報名的同學外，也包含來自陽明交通大學的郭朝恩同學，我們主要將整個題目分成以下五個部分進行：

(1) Host Program & System Integration: 李承澔、林奕杰
(2) Stereo Matching：林妤諠
(3) Feature Extraction：鄧文瑜
(4) Feature Matching：郭朝恩
(5) Motion Estimation：郭朝恩

且每週皆會與教授、助教於線上討論各自的進度及遇到的問題。