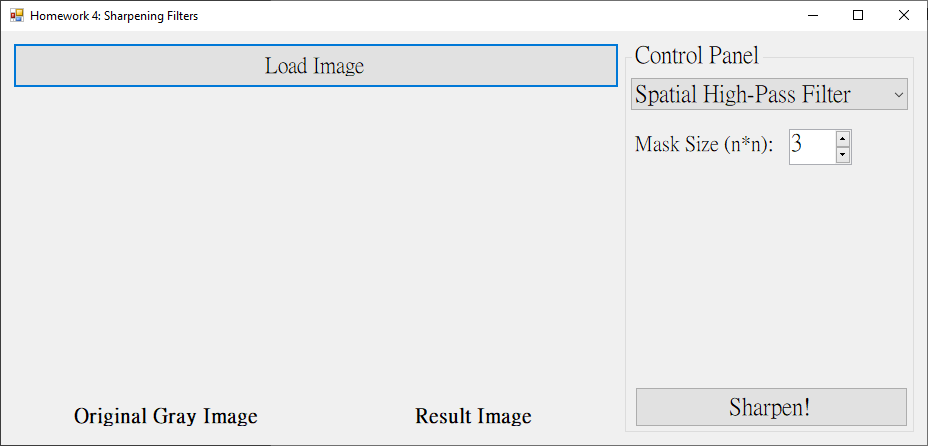
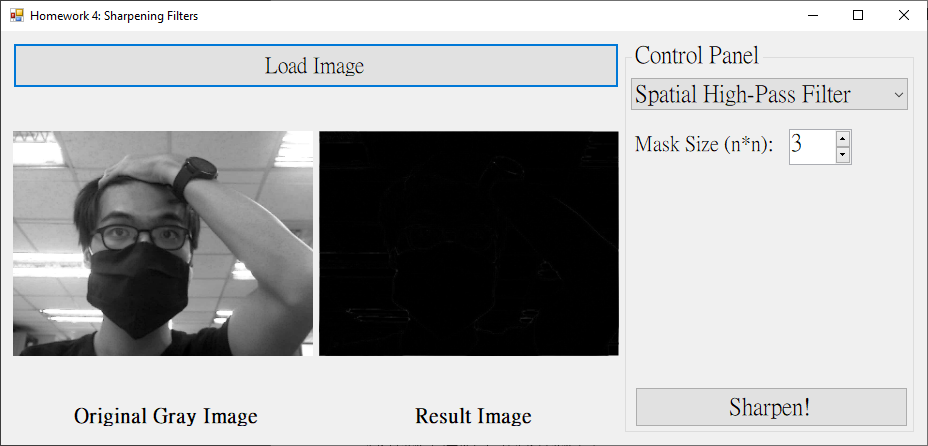
四資工三甲 C108151131 賴冠綸

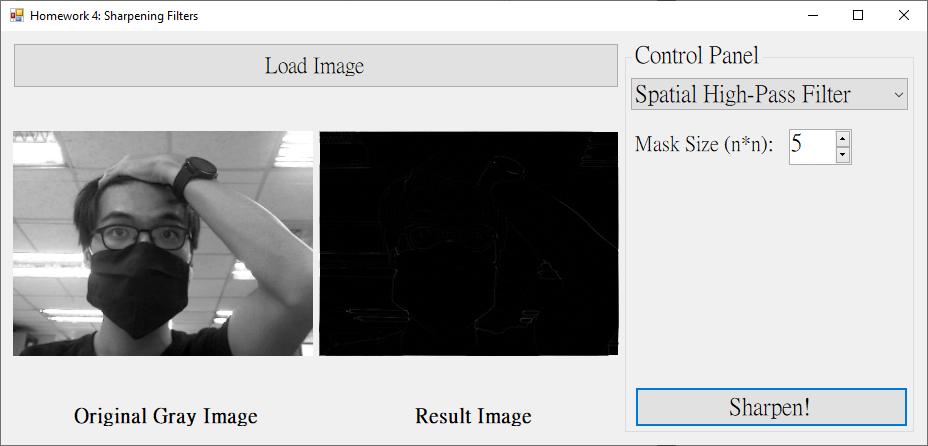
1. 程式執行畫面
   * 1. 程式起始畫面



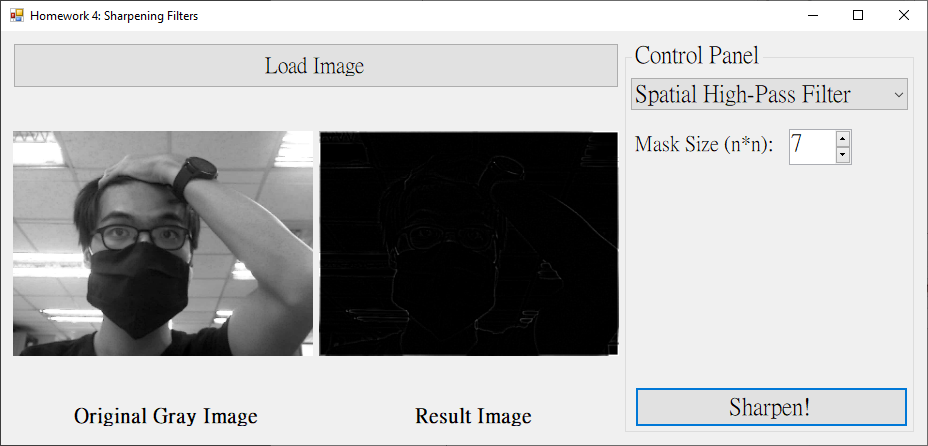
* + 1. High-Pass Filter
       1. Mask Size = 3x3



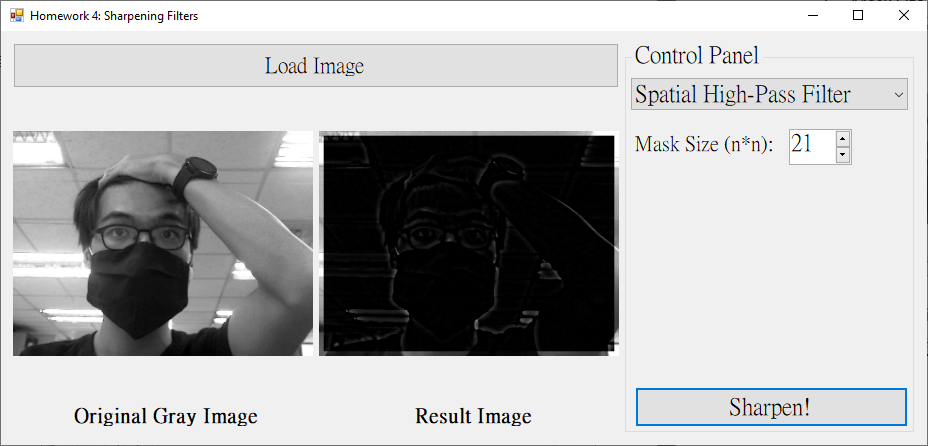
* + - 1. Mask Size = 5x5



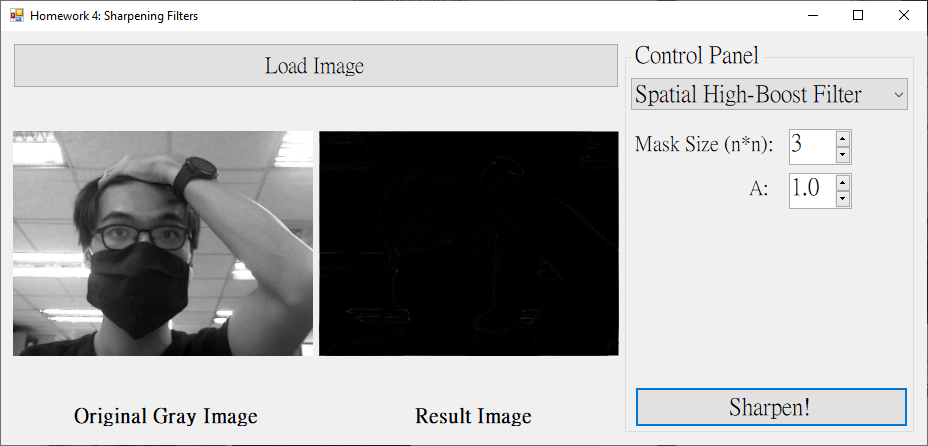
* + - 1. Mask Size = 7x7



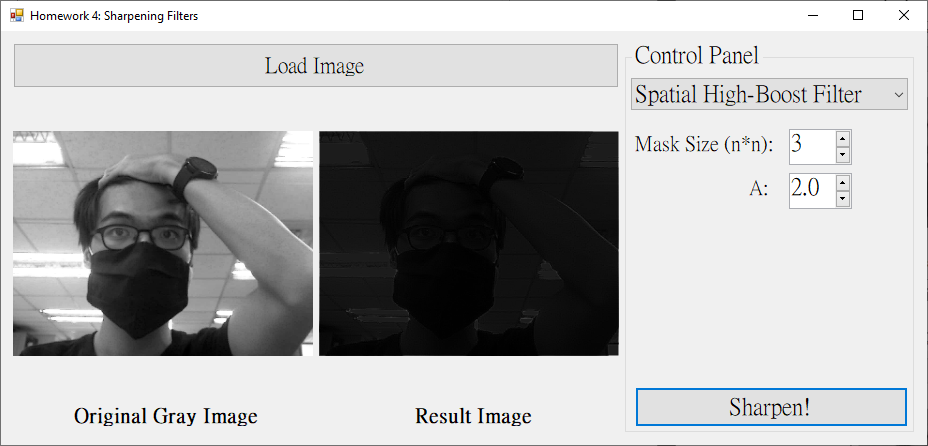
* + - 1. Mask Size = 21x21



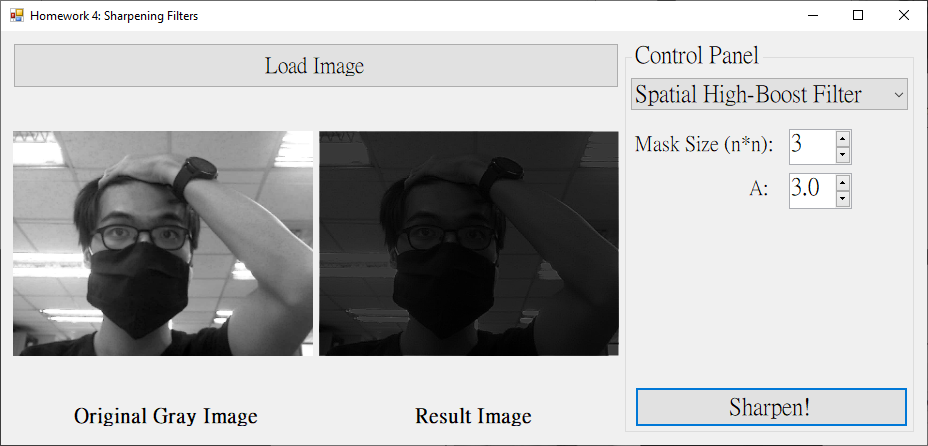
* + 1. High-Boost-Filter
       1. Mask Size = 3x3, A = 1 (As same as High-Pass Filter)



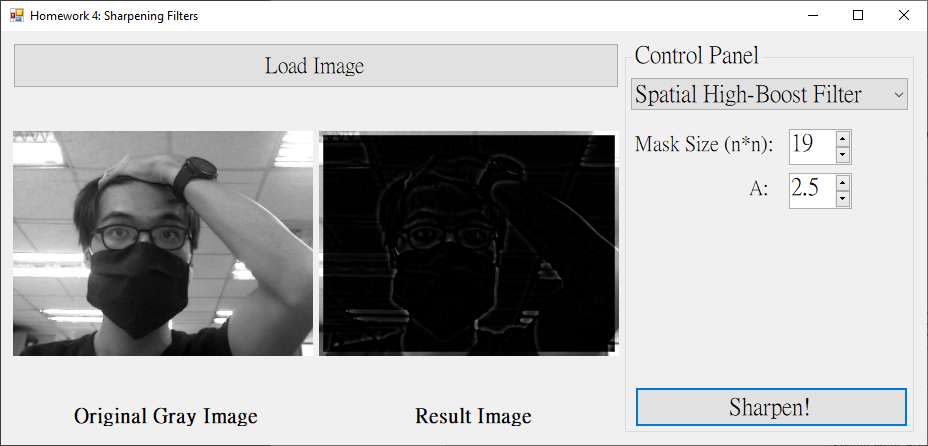
* + - 1. Mask Size = 3x3, A = 2



* + - 1. Mask Size = 3x3, A = 3



* + - 1. Mask Size = 19x19, A = 2.5



1. 程式碼（僅附主要程式碼）
   * 1. High-Pass Filter

|  |
| --- |
| private: Bitmap^ HighPassFilter(int n, Bitmap^ gray)  {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadWrite, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  int blank = n / 2;  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  // avoid out of the range  if (x >= blank && x < grayBD->Width - blank  && y >= blank && y < grayBD->Height - blank)  {  // vector template needs import library "vector"  std::vector<int> b;  std::vector<int> g;  std::vector<int> r;  for (int i = -blank; i <= blank; i++)  {  for (int j = -blank; j <= blank; j++)  {  Byte\* ptr = grayPtr + (x + j) \* 3 + (y + i) \* grayBD->Stride;  b.push\_back(ptr[0]);  g.push\_back(ptr[1]);  r.push\_back(ptr[2]);  }  }  int setRate;  int bSum = 0, gSum = 0, rSum = 0;  for (int i = 0; i < n \* n; i++)  {  if (i == n \* n / 2) setRate = n \* n - 1;  else setRate = -1;  bSum += b[i] \* setRate;  gSum += g[i] \* setRate;  rSum += r[i] \* setRate;  }  if (bSum < 0) bSum = 0;  if (gSum < 0) gSum = 0;  if (rSum < 0) rSum = 0;  resultPtr[0] = bSum / (n \* n);  resultPtr[1] = gSum / (n \* n);  resultPtr[2] = rSum / (n \* n);  }  else // if is edge  {  Byte\* ptr = grayPtr + x \* 3 + y \* grayBD->Stride;  resultPtr[0] = ptr[0];  resultPtr[1] = ptr[1];  resultPtr[2] = ptr[2];  }  resultPtr += 3;  }  resultPtr += resultByteskip;  }  gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |

* + 1. High-Boost Filter

|  |
| --- |
| private: Bitmap^ HighBoostFilter(int n, double A, Bitmap^ gray)  {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadWrite, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  int blank = n / 2;  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  // avoid out of the range  if (x >= blank && x < grayBD->Width - blank  && y >= blank && y < grayBD->Height - blank)  {  // vector template needs import library "vector"  std::vector<int> b;  std::vector<int> g;  std::vector<int> r;  for (int i = -blank; i <= blank; i++)  {  for (int j = -blank; j <= blank; j++)  {  Byte\* ptr = grayPtr + (x + j) \* 3 + (y + i) \* grayBD->Stride;  b.push\_back(ptr[0]);  g.push\_back(ptr[1]);  r.push\_back(ptr[2]);  }  }  double setRate;  int bSum = 0, gSum = 0, rSum = 0;  for (int i = 0; i < n \* n; i++)  {  if (i == n \* n / 2) setRate = A + n \* n - 2;  else setRate = -1;  bSum += b[i] \* setRate;  gSum += g[i] \* setRate;  rSum += r[i] \* setRate;  }  if (bSum < 0) bSum = 0;  if (gSum < 0) gSum = 0;  if (rSum < 0) rSum = 0;  resultPtr[0] = bSum / (n \* n);  resultPtr[1] = gSum / (n \* n);  resultPtr[2] = rSum / (n \* n);  }  else // if is edge  {  Byte\* ptr = grayPtr + x \* 3 + y \* grayBD->Stride;  resultPtr[0] = ptr[0];  resultPtr[1] = ptr[1];  resultPtr[2] = ptr[2];  }  resultPtr += 3;  }  resultPtr += resultByteskip;  }    gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |