```cpp
#include <iostream>

#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>

#define BGR_BLUE  0
#define BGR_GREEN 1
#define BGR_RED   2


using namespace cv;

void deBayer(Mat *rawImg, Mat *outImg) {
  // the Daheng camera datasheet specifies a GRBG pattern (page 88). Our input values will be 8 bit
  // Split 1 channel image into 3 channels according to bayer pattern
  //   0  1
  //0  G  R
  //1  B  G

  //I will create a new "Mat" with OpenCV that contains three channels. The splitting and processing is done by hand.
  if (rawImg->type() != CV_8UC1)
    throw("Sorry, only 1 8-bit channel should be used");

  (*outImg) = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); // Fill output buffer with zeros with the correct geometry
  MAT outImgMID = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); // Fill output buffer with zeros with the correct geometry
  //Size s = rawImg->size(); I will be using rows and colums rather than height and width
  long row, col;

  std::cout << "Bayer splitting to 3 channels" << std::endl;

  for(row=0; row<rawImg->rows; row++){  //todo: make this evaluation smaller to increase speed
    for(col=0; col<rawImg->cols; col++){
      if (row % 2 == 0 && col % 2 == 0) //odd row, odd column
        outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>(row,col);
      if (row % 2 == 0 && col % 2 == 1) //odd row, even column
        outImgMID->at<Vec3b>(row, col).val[BGR_RED] = rawImg->at<uint8_t>(row,col);
      if (row % 2 == 1 && col % 2 == 0) //even row, odd column
        outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] = rawImg->at<uint8_t>(row,col);
      if (row % 2 == 1 && col % 2 == 1) //even row, even column
        outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>(row,col);
    }
  }

  // Interpolate green channel by taking the value of the nearest neighbour that is green
  std::cout << "Interpolating green channel" << std::endl;
```

```
  for(row=0; row<outImgMID->rows; row++){
    for(col=0; col<outImgMID->cols; col++){
      if (outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] == 0){
        if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_GREEN] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
(row, col-1).val[BGR_GREEN];
        else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
col+1).val[BGR_GREEN] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
(row, col+1).val[BGR_GREEN];
        else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_GREEN]
!= 0)
          outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
(row-1, col).val[BGR_GREEN];
        else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
col).val[BGR_GREEN] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
(row+1, col).val[BGR_GREEN];
      }
    }
  }

  // Interpolate red channel by taking the value of the nearest neighbour
that is red
  std::cout << "Interpolating red channels" << std::endl;

  for(row=0; row<outImgMID->rows; row++){
    for(col=0; col<outImgMID->cols; col++){
      if (outImgMID->at<Vec3b>(row, col).val[BGR_RED] == 0){
        if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_RED] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
(row, col-1).val[BGR_RED];
        else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
col+1).val[BGR_RED] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
(row, col+1).val[BGR_RED];
        else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_RED] !=
0)
          outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
(row-1, col).val[BGR_RED];
        else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
col).val[BGR_RED] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
(row+1, col).val[BGR_RED];
      }
    }
  }

  // Interpolate blue channel by taking the value of the nearest neighbour
that is blue
  std::cout << "Interpolating blue channels" << std::endl;

  for(row=0; row<outImgMID->rows; row++){
    for(col=0; col<outImgMID->cols; col++){
      if (outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] == 0){
        if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_BLUE] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
(row, col-1).val[BGR_BLUE];
```

```
        else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row, col+1).val[BGR_BLUE] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>(row, col+1).val[BGR_BLUE];
        else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_BLUE] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>(row-1, col).val[BGR_BLUE];
        else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1, col).val[BGR_BLUE] != 0)
          outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>(row+1, col).val[BGR_BLUE];
      }
    }
  }

  // done?
  std::cout << "Done?" << std::endl;

}

int main() {
  // Read the image (in 8bit grayscale)
    Mat img = imread("test_RAW.png", CV_8UC1);
    if(img.empty()) {
        std::cout << "Could not read the image: " << std::endl;
        return 1;
    }

    Mat result;
    deBayer(&img, &result);


    imshow("Display window", result);
    waitKey(0); // Wait for a keystroke in the window
    return 0;
}

```