REAL TIME IMAGING

Final Project
PixelStreams: Object Tracking

GARCIA, Frederic (u1038431)
Objectives

The aim of this final project is to develop an object tracking algorithm in real-time using Handel-C and the PixelStreams library. A default workspace and a set of test images will be given which will help to both develop and evaluate your algorithm.

Introduction

In spite of having a clearly description about what we have to develop in this lab exercise, the main handicap is the necessity of change our way to perform applications. This subject is centred in real time imaging and we are not use to program for real-time applications.

The previous lab exercises were guide and they were based on follow a list of steps to obtain the expected results. Here we have to consult and understand how Handel-C works, aspects such as its format conversion types or the way to treat the memory space.

To sum up, to achieve the lab exercise goal, the code has to be compiled to work over a Celoxica RC10 board (view Figure 1.1) and has to track in real-time the target over the screen.

![Celoxica RC10 board](image)

Figure 1.1. Celoxica RC10 board.

a) Approach to solve the tracking problem

The first step to perform this lab exercise was load the default workspace and observes how it works.

It was observed also that depending on the testType selected, the behaviour of the target was easier or more difficult to follow.

The simplest case consists on a black background where the target is navigating slowly and without getting out of the screen coordinates (view Figure 1.2a). On the other hand, the hardest case presents a background which confuses with the target (due to its contrast) and, furthermore, the target can disappear and moves according with a free trajectory, velocity and acceleration (view Figure 1.2b).
After observe the differences between the RTI_EXTREME and the RTI_EASY test types it was decided to work only with the extreme case. The reason of this decision was the supposition that if the code works in the extreme case, it has to work also in the easier cases.

The lab exercise performed can be divided in two main steps; the former consists on transform the RTI_EXTREME case to the RTI_EASY case while the latter consists on draw the bounding box and the cursor which follows the target.

**Converting the hardest case to the easiest case**

The significant difference between the hardest case and the easiest case is the background contrast where we have to follow the target (view Figure 1.2). However, following the next steps it can be solve:

- The first step consists on apply a threshold to binarize the image.

From figures 1.3a and 1.3b (a simulation done with Adobe Photoshop of the response after applying the threshold to the original image) can be observe that the areas of the image which are white (as the target to follow) doesn’t disappear.

- To obtain a consistent background it is necessary to delete these white areas. A possible solution consists on applying an erode process followed by a dilate process. After apply erode, all the white areas will disappear but, consequently the target dimensions will be also reduced. Then, it is necessary to apply the dilate process to obtain the real dimensions of the target to follow.
There is a Handel-C command which implements that function and it is `PxsClose`. Then, applying the following Handel-C code, the extreme background will be converted to the easy background (also occurs with the difficult and medium backgrounds).

```c
// Convert the complex backgrounds to make easy the centroid computation
PxsThreshold(&OStr,&OStrTh,0,254); // Binarize the original stream
// With PxsClose all the noisy points are deleted (erode+dilate)
PxsClose(&OStrTh,&OStrThClose,Width);
```

**Handel-C**

**Following the target**

This second main step has been the most difficult to achieve. There are as minimum two different possibilities to perform this step:

- The first possibility, and also the easiest, consists on save the first coordinate of the target (top-left coordinate) and the last coordinate of the target (bottom-right coordinate). Once these coordinates are saved, the mean of both will be the centroide of the target where the cursor will be drawn and also the bounding box at certain distance of it.

Although this technique works, it only works for targets which are rectangular or square. If the target has an irregular form, then this technique is not suitable.

- The second possibility, and the performed one, is based on the centroide calculation of the target (independently of the target form). Once the centroide coordinates are computed, the cursor can be drawn and also the bounding box.

To compute the target centroide it is necessary to sum all the \((x,y)\) coordinates of the target and divide by the number of target pixels:

\[
coord_x = \frac{\sum x}{N}, \quad coord_y = \frac{\sum y}{N}
\]

This is a simply concept; however the real-time implementation is not such easy. It is necessary to acquire the target top-left and bottom-right coordinates according to the synchronism of the stream after being computed (threshold + close process) and also the number of pixels that the target has. At the same time (in a parallel process thread), at each cycle (when all the pixels of the target have been computed), the mean of them is computed to obtain the centroide coordinates and to can draw the cursor and also the bounding box.

The cursor is directly drawn in the centroide coordinates while to draw the bounding box it is necessary to make few computations. The target top-left and bottom-right coordinates are known. Then, a 10 pixels bigger and a 5 pixels bigger rectangles are drawn (with the same centroide as the target). With the `PxsSubSat` Handel-C command can be subtracted the smaller rectangle to the bigger one and the resulting will be the bounding box for the target (view Figure 1.4).
The Handel-C code implemented for this section is:

```
// Generates the stream with the outer and inner rectangle
// which will be the bounding box of the target
PxsRectangle(&OStr,&OutRectStr,TL_x-10,TL_y-10,RB_x+10,RB_y+10,1); // Outer rectangle
PxsRectangle(&OStr,&InRectStr,TL_x-5,TL_y-5,RB_x+5,RB_y+5,1); // Inner rectangle
PxsSubSat(&OutRectStr,&InRectStr,&BoundBoxStr); // Makes the difference between the previous both streams (bounding box)
// Adding the bounding box to the original stream
PxsAddSat(&OStr,&BoundBoxStr,&TargetStr);
// Draw the cursor in the centroid of the target object
PxsCursor(&TargetStr,&CentroidCursorStr,mean_x,mean_y,PXS_CURSOR_CROSS);
```

```
while (1)
{
    par
    |
    if (OStrThCloseInv.Valid && OStrThCloseInv.Active &&
    OStrThCloseInv.Pixel.U1[PXS_M]==1)
    {
        par
        |
        // Compute the target coordinates
        // Find the maximum x coordinate
        if (OStrThCloseInv.Coord.X > coord_max_x)
        {
            coord_max_x=OStrThCloseInv.Coord.X;
        }
        else delay;
        // Find the minimum x coordinate
        if (OStrThCloseInv.Coord.X < coord_min_x)
        {
            coord_min_x=OStrThCloseInv.Coord.X;
        }
        else delay;
        // Find the max y coordinate
        if (OStrThCloseInv.Coord.Y > coord_max_y)
        {
            coord_max_y=OStrThCloseInv.Coord.Y;
        }
        else delay;
        // Find them minimum y coordinate
        if (OStrThCloseInv.Coord.Y < coord_min_y)
        {
            coord_min_y=OStrThCloseInv.Coord.Y;
        }
        else delay;
        // Necessary values for the centroide computation
        current_x=OStrThCloseInv.Coord.X;
        current_y=OStrThCloseInv.Coord.Y;
        sum_x=sum_x+(0@current_x);
        sum_y=sum_y+(0@current_y);
        nPixels=nPixels+1;
    }
    else delay;
}
while(1)
{
    // At each cycle
```
As can be observed from the previous code, there is the macro Divide which has been modified to can perform divisions between signed 16 bits variables.

b) Handel-C code used and the code

```handel-c
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"
#include "moveBall.hch"

macro proc Divide (X, Y, ResultPtr);

 ['/********************** GLOBAL VARIABLES **********************/
 static signed int 32 sum_x = 0;
 static signed int 32 sum_y = 0;
 static signed int 16 current_x = 0; // Current x coordinate
 static signed int 16 current_y = 0; // Current y coordinate
 static signed int 16 nPixels = 0;   // Number of pixels counted (the target pixels)
 signed int 16 mean_x, mean_y;  // Mean coordinates (to save the centroid)

 /******************************************************************************
 ************************ MAIN **********************************************/
 void main (void)
 {
   // Macros definitions
   macro expr ClkRate = PAL_ACTUAL_CLOCK_RATE;
   macro expr Mode = SyncGen2GetOptimalModeCT (ClkRate);
   macro expr Width = SyncGen2GetHActivePixelsCT (Mode);
   macro expr Height = SyncGen2GetVActiveLinesCT (Mode);
   macro expr testType = RTI_EXTREME; //RTI_EASY, RTI_MEDIUM , RTI_DIFFICULT, RTI_EXTREME

   // Streams declaration
   PXS_PV_S (VGASync, PXS_EMPTY);  // Initial signal
   PXS_PV_S (OStr, PXS_MONO_U8);  // Original stream (with the moving ball)
   PXS_PV_S (OStrTh, PXS_MONO_U1); // Original stream after apply the threshold
   PXS_PV_S (OStrThClose, PXS_MONO_U1); // Thresholded stream after being closed
   PXS_PV_S (OStrThCloseInv, PXS_MONO_U1); // Stream which recover the original colors lost in the threshold process
   PXS_PV_S (OStrThCloseInvUnsync, PXS_MONO_U1); // Stream which recover the original colors lost in the threshold process
   PXS_PV_S (OutRectStr, PXS_MONO_U8); // Outer rectangle, used to generate the bounding box
   PXS_PV_S (InRectStr, PXS_MONO_U8); // Inner rectangle, used to generate the bounding box
   PXS_PV_S (BoundBoxStr, PXS_MONO_U8); // Stream after generating the bounding box
   PXS_PV_S (TargetStr, PXS_MONO_U8); // Target envolved by the bounding box

   // Centeroid computation (mean_x & mean_y are the centroid coordinates)
   Divide(sum_x, nPixels, &mean_x);
   Divide(sum_y, nPixels, &mean_y);
   // Reset the values for the next cycle computation
   sum_x = 0;
   sum_y = 0;
   nPixels = 0;
   coord_min_x = 9999;
   coord_max_x = 0;
   coord_min_y = 9999;
   coord_max_y = 0;

   // Assign the target top-left and bottom-right coordinates
   TL_x=coord_min_x;
   TL_y=coord_min_y;
   RB_x=coord_max_x;
   RB_y=coord_max_y;

   // Macros definitions
   macro proc Divide (X, Y, ResultPtr);

   } // main
```

RTI-FINAL PROJECT

PixelStreams: Object Tracking

PXS_PV_S (CentroidCursorStr, PXS_MONO_U8); // Target with a cross in its centroide and
envolved by the bounding box
PXS_PV_S (OutputStr, PXS_RGB_U8);  // Final output with the tracking object

// Variables
static signed int 16 TL_x = 0;
static signed int 16 TL_y = 0;
static signed int 16 RB_x = 1024;
static signed int 16 RB_y = 768;
static signed int 16 coord_min_x = 9999;  // Necessary as a maximum and non possible
value
static signed int 16 coord_max_x = 0;  // Necessary as a minimum possible value
static signed int 16 coord_min_y = 9999;
static signed int 16 coord_max_y = 0;

// Code
par
{
  PxsVGASyncGen(&VGASync,Mode);  // Generates the VGA Signal
  rtiMoveBall(&VGASync,&OStr,Width,Height,testType); // Creates the moving ball

  // Generates the stream with the outer and inner rectangle
  // which will be the bounding box of the target
  PxsRectangle(&OStr,&OutRectStr,TL_x-10,TL_y-10,RB_x+10,RB_y+10,1); // Outer
  rectangle
  PxsRectangle(&OStr,&InRectStr,TL_x-5,TL_y-5,RB_x+5,RB_y+5,1);    // Inner
  rectangle
  PxsSubSat(&OutRectStr,&InRectStr,&BoundBoxStr); // Makes the difference
  between the previous both streams (bounding box)

  // Adding the bounding box to the original stream
  PxsAddSat(&OStr,&BoundBoxStr,&TargetStr);
  // Draw the cursor in the centroid of the target object
  PxsCursor(&TargetStr,&CentroidCursorStr,mean_x,mean_y,PXS_CURSOR_CROSS);

  // Convert the complex backgrounds to make easy the centroid computation
  PxsThreshold(&OStr,&OStrTh,0,254);    // Binarize the original stream
  PxsClose(&OStrTh,&OStrThClose,Width); // With PxsClose all the noisy
  points are deleted (erode+dilate)
  PxsInvert(&OStrThClose,&OStrThCloseInv);

  while (1)
  {
    par
    {
      if (OStrThCloseInv.Valid && OStrThCloseInv.Active &&
      OStrThCloseInv.Pixel.U1[PXS_MJ]==1)
      {
        par
        {
          // Compute the target coordinates
          // Find the maximum x coordinate
          if (OStrThCloseInv.Coord.X > coord_max_x)
          {
            coord_max_x=OStrThCloseInv.Coord.X;
          }
          else delay;
          // Find the minimum x coordinate
          if (OStrThCloseInv.Coord.X < coord_min_x)
          {
            coord_min_x=OStrThCloseInv.Coord.X;
          }
          else delay;
          // Find the max y coordinate
          if (OStrThCloseInv.Coord.Y > coord_max_y)
          {
            coord_max_y=OStrThCloseInv.Coord.Y;
          }
          else delay;
          // Find them minimum y coordinate
          if (OStrThCloseInv.Coord.Y < coord_min_y)
          {
            coord_min_y=OStrThCloseInv.Coord.Y;
          }
          else delay;
          // Necessary values for the centroide computation
        }
      }
    }
  }
current_x = OStrThCloseInv.Coord.X;
current_y = OStrThCloseInv.Coord.Y;
sum_x = sum_x + (0 @ current_x);
sum_y = sum_y + (0 @ current_y);
nPixels = nPixels + 1;
}
}
else delay;
}
while(1)
{
    // At each cycle
    PxsAwaitVSync(&OStr);
    par
    {
        // Assign the target top-left and bottom-right coordinates
        TL_x = coord_min_x;
        TL_y = coord_min_y;
        RB_x = coord_max_x;
        RB_y = coord_max_y;
        // Centroid computation (mean_x & mean_y are the centroid coordinates)
        Divide(sum_x, nPixels, &mean_x);
        Divide(sum_y, nPixels, &mean_y);
        // Reset the values for the next cycle computation
        sum_x = 0;
        sum_y = 0;
        nPixels = 0;
        coord_min_x = 9999;
        coord_max_x = 0;
        coord_min_y = 9999;
        coord_max_y = 0;
    }
    PxsConvert(&CentroidCursorStr, &OutputStr); // Convert the response stream to the VGA out format
    PxsVGAOut(&OutputStr, 0, 0, ClkRate);  // Show the response
}

/************************** END MAIN ******************************/
/******************************************************************/
// Macro modification to can calculate the centroid computation

macro proc Divide (X, Y, ResultPtr)
{
    signed 48 A, B, C;
    signed 6 Bits;
    if (Y != 0)
    {
        par
        {
            A = (signed)(0 @ X);
            B = (signed)(0 @ Y);
            C = 0;
            Bits = 1;
        }
        while (B @ (signed 1)0 <= (signed 1)0 @ A)
        {
            par
            {
                B <<= 1;
                Bits++;
            }
        }
        do
        {
            par
            {
                if (A >= B)
                {
                    par
                    {
                        // Code...
                    }
                }
            }
        }
    }
}
c) Other aspects such as the problems experienced and the discussion of the results.

- **RTI_EASY** test type:
- **RTI_MEDIUM** test type:

![RTI_MEDIUM](image1)

*Figure 1.6. RTI_MEDIUM test type execution.*

- **RTI_DIFFICULT** test type:

![RTI_DIFFICULT](image2)

*Figure 1.7. RTI_DIFFICULT test type execution.*
RTI_EXPERT test type:

Figure 1.8. RTI_EXPERT test type execution.

As has been said in the introduction of this report, the difficulty of this lab exercise was related with the necessity of changing the way to think about how to program. The fact of been working in real-time make me to know how to program in parallel, how to obtain several values at the same cycle and how to work with them. Once this concept was understood, the rest of difficulties were related with Handel-C, which is quite similar to C or C++ but with significant differences when you want to convert types of variables or save them.

It is an interesting lab exercise, where you can test several possibilities and improve your code, be more familiar with real-time, etc. However, during the last weeks the work related with the rest of subjects make impossible to dedicate more time to this lab exercise. In spite of this aspect, the aim of the exercise has been achieved.