Programming Project GM 2018: 'Air Hockey'

- REPORT -

Authors: M. Zemp (Sciper-nr. 284044) & I. Scholl (282558) GM BA 2

Platform: Windows 10, CodeBlocks (for C++)

Files: bin – Folder crated by CodeBlocks
obj – Folder created by CodeBlocks
Seq1m – Folder with the PNG images of the first sequence
Seq2m – Folder with the PNG images of the second sequence
AH_score – PDF file containing puck's graphs created by Matlab
AirHockey.vi – Main VI to open first, from which the whole program can be launched
ComputeScore.m – Matlab script created by LabVIEW and executed by Matlab
main.cpp – 'Main' code part of 'Pix2Pos' C++ executable
Pix2pos.depend – Additional file to the executable created by CodeBlocks
Pix2pos.exe – Executable for analysis of puck's coordinates, called by LabVIEW
Pix2Pos.layout – Additional file to the executable created by CodeBlocks
pixmap.bin – Binary file created by LabVIEW and read by the C++ executable
pos.txt – Text file written by C++ and read by LabVIEW
Report_SchollZemp – This report to the project
SearchCoord.vi – Sub-VI retaining the puck's coordinates in every picture
WriteScript.vi – Sub-VI creating the script later executed by Matlab

Introduction

This project has as a goal to analyse and trace the movements of two pucks being put on an Air Hockey table. For this, a combination of the three programs C++, LabVIEW and Matlab is used. The user only must choose the parameters he likes, the rest is executed by the programs themselves and the result is displayed as well as saved into a PDF file if no errors occur. Possible errors are treated directly by the program - depending on their effect on the program's execution - and stated to the user.

Working with other groups: Some parts especially of the LabVIEW code, as for example the error treatment and the 'WriteScript' sub-VI, have been developed directly with the group of *Sofya Aman and Baptiste Bayle*, as we have been working on that part a lot together. Therefore, parts might be resembling each other or even the same. Also, we have shared some of our ideas - especially for the LabVIEW part too – with the group of *Dominik Kunz and Clément Lorée*.

C++

The C++ part must find the positions of the pucks in the picture. The program receives the document 'pixmap.bin' as an input, in which the two colours 0 and 1, the length and the height, and all the pixels are saved. To find the pucks it runs through every pixel and checks its colour. Once the colour is 0 or 1, which corresponds to puck 1 and 2 respectively, the program checks if the puck pattern of 5x5 is fulfilled. If it is the case, the program saves this position and checks if none of the pucks is found twice. In the end the found positions are saved into a document, 'pos.txt'.

During the research and the calculations of the program there are several checks for errors, some of them make the program stop, others work just as warnings. But all of them are given out as a standard error output. In the following list one can see, how we dealt with each problem and what our program does if the specific cases occur:

Warnings (the program can still be executed; a warning message is transmitted to the LabView interface as a standard error output (*cerr*))

- *pixel not valid*
  The pixels should code a number between 0 and 255, if a pixel is outside this range, there occurs a warning. Because we're only looking for pixels of values 0 and 1 to find the pucks, this does not hinder us to reach our objective.
- *colours are not correct*
  The colours should be 0 and 1, for the pucks 1 and 2 respectively. If the values in 'pixmap.bin' differ from them, a warning occurs. The program continues and looks still for pucks of colour 0 and 1.
- *too many pixels*
  The number of pixels is given by length times height. If we have too many pixels, we continue nonetheless (and check just those pixels that are in our range), but the program gives out a warning.

Errors (the program must be stopped; an error code is transmitted to the LabView interface as a standard error output (*cerr*))

1. *'pixmap.bin' can't be opened*
   The file can't be opened, so we don't have any data to deal with.
2. *'pixmap.bin' can't be read*
   The file can be opened, but there occurs a problem with reading its data, so again we have no data to deal with.
3. *largeur not valid*
   We want as width a value between 10 and 1000. If it isn't we stop the process as the picture is not appropriate for our execution of the program.
4. *hauteur not valid*
   Same as for width (largeur).
5. *not enough pixels*
   If we don't have length times height pixels we have to break, because our program would not work properly if we wouldn't fill our vectors totally.
6. *'pixmap.bin' can't be closed*
   The file can't be closed, so something went wrong, and we can't continue.

7 *two pucks of colour 0*
   If we have two pucks of the same colour, we stop. There must be an error in the 'pixmap.bin'.
8 *two pucks of colour 1*
   See 7
9 *could not save positions of puck 1*
   An error occurred while saving the located positions inside the program, so LabVIEW wouldn't run properly.
10 *could not save positions of puck 2*
   See 9
11 *positions of pucks are not valid*
   The pucks must not touch the borders of each other, if they do, their positions aren't valid.
12 *problem with creating/writing into/closing file 'pos.txt'*
   The process of creating, writing and closing the file 'pos.txt' was not successful, so LabVIEW can't read from it.

If no error occurred the program gives out as a standard output (*cout*) the word "success", to illustrate its successful running. A document 'pos.txt' has then been created and can be read by LabVIEW.

For further detailed information on single parts of the code refer to the code directly, that is itself commented line by line.

LabVIEW

Our LabVIEW part consists of the main VI 'AirHockey' and two sub-VIs 'Search Coord' and 'WriteScript'.

When you open the main VI 'AirHockey' the user interface appears on top left, from which you can control the whole execution. You can observe the pictures that are analysed one by one and see their corresponding frame number below the bottom right corner of the picture frame.

Also, you can choose the colours ('C0' & 'C1') in which the graphs of the two pucks should be created, as well as for how many seconds ('Pause') you want to display the graph when Matlab is executed. Additionally, you can decide whether you want the program to execute Matlab ('Create Matlab Scoresheet') or whether you just want to analyse the pictures and prepare the Matlab script. In the top right corner, you can choose the sequence to be analysed.

In the bottom right corner, the 'Warnings & Errors' interface allows the user to check whether the run was successful or whether there have been errors given out by C++, LabVIEW or Matlab.

Once all the parameters have been chosen the VI will open the sequence to analyse. With every picture it executes a loop containing the sub-VI 'Search Coord', that gives each picture's puck's coordinates as output. These are then given on together with the chosen parameters to the sub-VI 'Write Script' who itself prepares the Matlab script and gives it as an output. The script is saved as 'ComputeScore.m' and then directly called again by Matlab (if chosen so in the interface), which is executed by LabVIEW. Once the execution of Matlab is finished the process is ended.

The sub-VI 'Search Coord' analyses every PNG picture given in and reads all its important data. With this it writes the binary file 'pixmap.bin' in the appropriate format. Next it launches the C++ executable who will read from that binary file and create itself the file 'pos.txt'. The VI then goes to read and retain the coordinates of the two pucks from this text file to later give them back to the main VI.

The sub-VI 'WriteScript' prepares the Matlab script. For this it creates vectors out of the puck's coordinates in every single picture, analyses the values for the colours, the pause and the chosen sequence and adds the rest of the Matlab code developed by us. This is then the complete script that is given as an output to the main VI and that will later be executed directly by Matlab.

Errors (are to be solved directly by the user whenever possible)

- *Sequence folder is wrong*
  If the selected sequence folder isn't correct *(can be tested by using the option 'sequence false')* a new window appears where the user is asked to choose an actual sequence folder to be analysed.
- *Wrong/damaged PNG picture*
  If a picture can't be read by LabVIEW the error is given directly into the error wire going through the 'Read PNG File'-VI, and therefore the execution stops, and an error message is displayed in the 'Warnings & Errors' interface.
- *C++ executable can't be found*
  If the C++ 'Pix2Pos' executable cannot be found at the expected path (same as the main VI) or the expected name then a new window appears, and the user is asked to choose the right path to the executable of any name, while he can only choose files of '.exe' format. This path is then remembered as expected path by a feedback node and its name is remembered by another feedback node for the next executions of the sub-VI 'Search Coord', so that the user doesn't have to choose the executable's path for every single execution with all pictures.
- *Check whether C++'s standard error output is warning or error*
  If the error output given by C++ is an error, the execution is stopped, and the error is directly displayed in the user interface. If it is a warning only, LabVIEW keeps running, but a warning will appear in the box 'Warning (C++)' in the 'Warnings & Errors' section of the user interface.
- *Others*
  All the main parts of the LabVIEW code are wired with the error wire, so if there should appear complications with any of the code's parts then the execution is stopped directly, and a message is displayed in the 'Warnings & Errors' interface.

For further information on the LabVIEW code refer directly to the main VI and sub-VIs where every block is commented separately.

MATLAB

In the MATLAB part our goal is to create a PDF-file, which illustrates the movements of our pucks. It shows the rebound at the borders and the collisions between the pucks and is executed based on the script created by our LabVIEW main VI 'AirHockey'.

For the rebounds the condition is simply a certain distance (d<=15 pixels) between the puck and the border, that must be reached or undercut by the puck. If a puck stays in this area for more than one frame, we only indicate the rebounds at the first point that is inside the area. Also, we don't indicate the very first rebounds of the pucks with the border, as they are yet to be launched onto the Air hockey table and not yet rebounding from the walls.

The collision between two pucks can occur under the following conditions: Firstly, the pucks must be in the same area (d<50 pixels) at the same time. And secondly, the deviation of the angle of the

trajectory of puck 1 must be greater than 1 rad. Furthermore, it must be checked, that this point is not a collision with the borders.

All the conditions above are checked by our Matlab code. Based on the data obtained we plot a graph of the two pucks in the colours chosen in the LabVIEW interface indicating rebounds and collisions with circles (see legend of the plot for distinction). Also, the time and date, name of the sequence, the code's authors names and numbers concerning rebounds and collisions are displayed. This whole is saved into a pdf file called 'AH_score'.

If any error occurs with running and executing Matlab, this should be displayed directly in the LabVIEW interface.

For more detailed information on particular parts of the code refer directly to the Matlab script, where every block is commented separately.

Conclusion

We have successfully managed to compute all the parts of the code such that the right result should be received for any sequence given in. Also, our program treats all the errors to be expected, depending on their effect on the execution. Additional features like for example distinguishing between warnings and actual errors given out by the C++ executable have been added and should not only work correctly but also make the usage even more convenient.

Therefore, we are happy and satisfied with the result and of opinion of having fulfilled the given task successfully.

For any questions or comments, please contact:

manuel.zemp@epfl.ch
ian.scholl@epfl.ch