Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

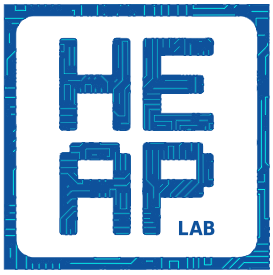
HEAPLab Project Report

The HoughCircles BarbequeRTRM application

Author:
Ayoub Benkhoris

Supervisor:
Dr. Giuseppe Massari

July 14, 2020



Abstract

The Hough Circle Transform is an image processing technique used to detect circles in an image. Its implementation is provided in the OpenCV library. In this project, this application is ported to the BarbequeRTRM project. The application is then modified to make it more adaptive and tested with different configurations to show some functionalities of the Barbeque resource manager and to study the impact of the different parameters on the performances of the application and the quality of the results.

1 Introduction

The Hough Transform is a technique used in image processing to identify instances of a shape in an image. It was patented in 1962 by Paul Hough and was initially designed to identify lines in an image. In this project we work with the Hough Circle Transform which is a specialized Hough Transform able to detect circles in an image.
An implementation of it is available in OpenCV, a library of programming functions aimed at real-time computer vision. In this project, the goal is to port this OpenCV implementation of the Hough Circle Transform to the Barbeque Run-Time Ressource Manager OpenSource Project (BOSP) developed at DEIB, Politecnico di Milano by the HEAPLab Group.

2 Design and Implementation

Once BOSP is downloaded, compiled and correctly working, one is able to create an application following the Adaptive Execution Model or port an existing application to BOSP with the **bbque-layapp** command.
This command generates a template for the application based on its type (C++, OpenCL, OpenCV, Python). In the following subsections, we will explain the design of our application.

2.1 The HoughCircles class

The HoughCircles class is defined in the *HoughCircles_exc.h* header file. It is derived from the BbqueEXC class. Indeed it contains, in addition to the constructor, a declaration of the different functions making it compatible

with BOSP (*OnSetup, OnConfigure, OnRun*, etc.). The *Work* function has been added, its objective is to contain the actual computation code of the application. These functions will be called from inside the class, so they are declared as private functions.
Moreover, in this class definition, a bunch of class variables are declared. They correspond to modifiable parameters or are variables related to resources management. Their purpose will be explained below.

2.2 Functions implementation

If we switch to the source file, *HoughCircles_exc.cc*, we can find the definition of the functions mentionned above:

1. *OnSetup*: this is the part where we initialize our system. First, the application verifies if the slow mode option is enabled. If it is, the program will be allowed to run at a reduced number of cycles per second. This feature is easily implementable in any project thanks to BOSP. The file containing the image to process is read and if the filename provided is pointing to a valid image, then this image is converted to grayscale. Besides, a median blur is applied to it to reduce noise and avoid false circle detection.

2. *OnConfigure*: this function was not modified except to add a line of code checking if any GPU is available. The code was tested on a virtual machine running with one processor core and limited memory so a complex management of resources did not seem useful. But instead of letting the user define a number of threads, we could have adapted the number of threads to use depending on the processing resources available for example.

3. *OnRun*: once the program is correctly setup, this function is called. It will initialize a certain number of concurrent execution threads depending on the user input. These threads will run the same task described in the *Work* function.

4. *OnMonitor*: between each *OnRun* call, a monitoring function is called. The quality of service is computed and logged. Furthermore, the remaining number of jobs to perform is checked and if it is lower than the number of threads to exploit, then the number of threads is reduced.

5. *OnSuspend* and *OnRelease*: when the program is suspended, a message is printed for the user. When it ends, the processed image with the detected circles is shown to the user, the program is fully terminated when the user closes the image window.

6. *Work*: this is where the magic (or computations) happens! The Hough Transform is applied to our blurred gray image. The precision of the detection is adjusted with parameters that the user can input in the command line. The circles found are stored in a three-dimensional vector and then drawn on the processed image. Finally, the number of jobs performed is incremented.

2.3 Command line options

The HoughCircles can run with different parameters provided by the user, they are implemented in *HoughCircles_main.cc*:

1. *help, -h*: prints the help message.

2. *version, -v*: prints the program version.

3. *conf, -C*: to provide a configuration file different from the default one.

4. *recipe, -r*: to provide the recipe name, by default it is "HoughCircles".

5. *filename, -f*: to provide the path to the image to process, by default the program looks for a file called "jo.jpg" in the current directory.

6. *threads, -t*: to change the number of threads to exploit, by default only one thread is executed.

7. *center_threshold, -c*: to change the threshold for center detection in the Hough Transform, equals to 30 by default and cannot be greater than 250.

8. *upper_threshold, -u*: to change the upper threshold for the internal Canny edge detector used by the Hough Transform, equals to 100 by default and cannot be greater than 500.

9. *jobs; -j*: to change the number of jobs to perform, by default *OnRun* is called 50 times.

10. *min_radius, -i*: to change the minimum radius of circles to be detected, equals to 1 by default and cannot be lower than 0.

11. *max_radius, -a*: to change the maximum radius of circles to be detected, equals to 30 by default and cannot be greater than 100.

12. *min_dist, -m*: to change the minimum distance between detected centers, equals to 16 by default and cannot be greater than 64. The higher the value, the lower the minimum distance between detected centers!

13. *slow_mode, -s*: to enable slow mode, is false by default and can be set to true to limit the program to 1 to 2 cycles per second.

3 Experimental Results

Let us test the HoughCircles BarbequeRTRM application. The source code is available at:

`https://github.com/ayoubbenkho/bbque-houghcircles/`

We suppose the application has been enabled in the BOSP configuration file and compiled along with BOSP as explained at:

`https://bosp.deib.polimi.it/`

The BOSP Shell must be started by sourcing the environment configuration script. Then, we will have to start the BarbequeRTRM daemon:

```
$ . ~/BOSP/out/etc/bbque/bosp_init.env
$ bbque-startd
```

The executable can be found at *~/BOSP/out/usr/bin/*. We will work with the "jo.jpg" image located in the same folder. We can now run the application with different options as shown in the following screenshots. We can start by printing the help message:

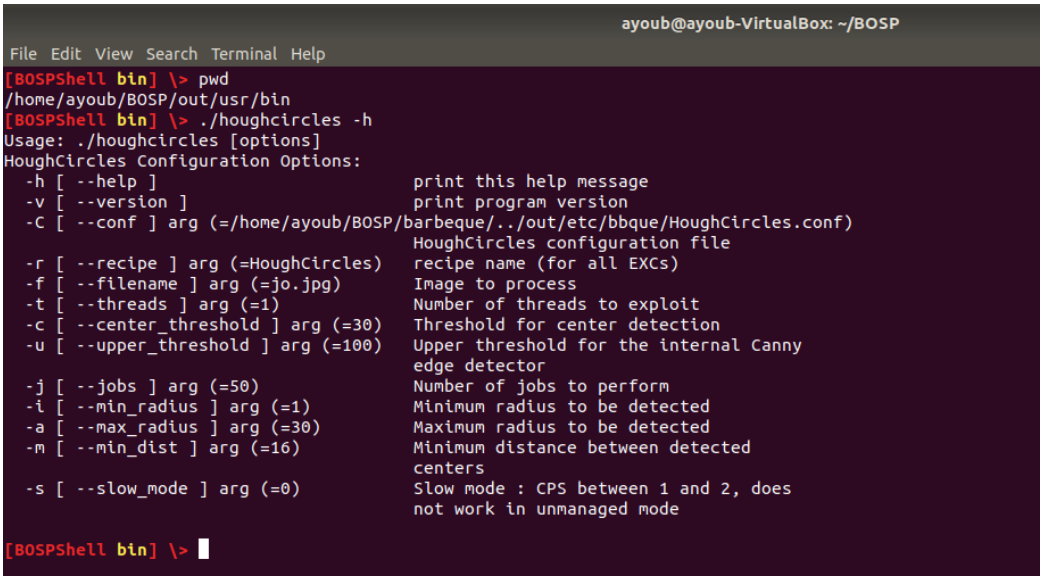


Figure 1: Help message.

Now we can try different options, starting with the slow mode. We notice that there is no significative difference in the unmanaged mode. Indeed, in this mode, the Barbeque scheduler is bypassed:

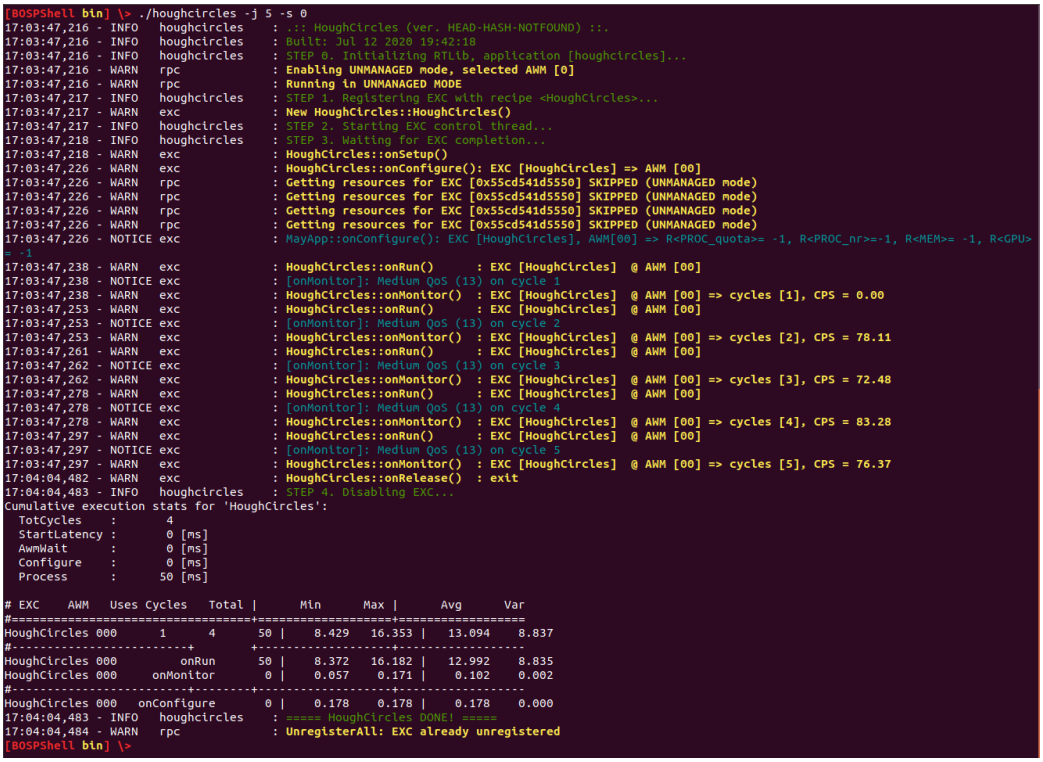


Figure 2: Unmanaged mode without slow mode.

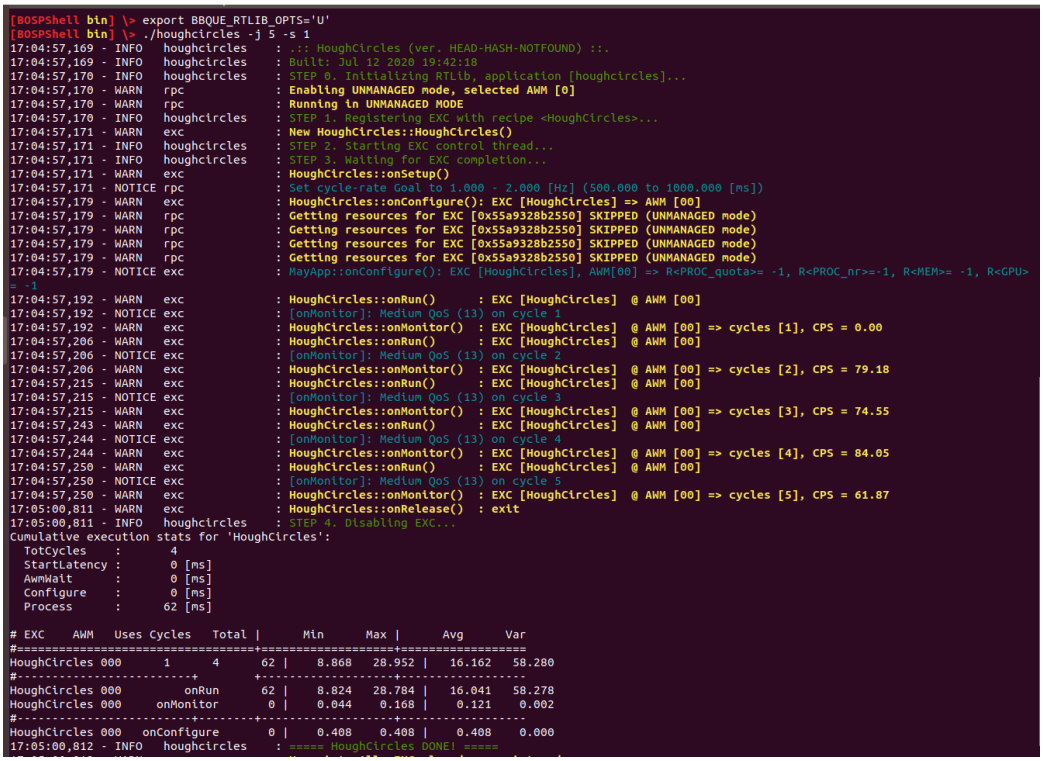


Figure 3: Unmanaged mode with slow mode.

We could exploit the C flag when exporting the "BBQUE _RTLIB_OPTS" variable to have more control on the cgroups. But here we will use the D flag which will stop the application after a certain amount of time or cycles and will let the Barbeque scheduler do its work, we can now see the effects of slow mode on the cycles per second:

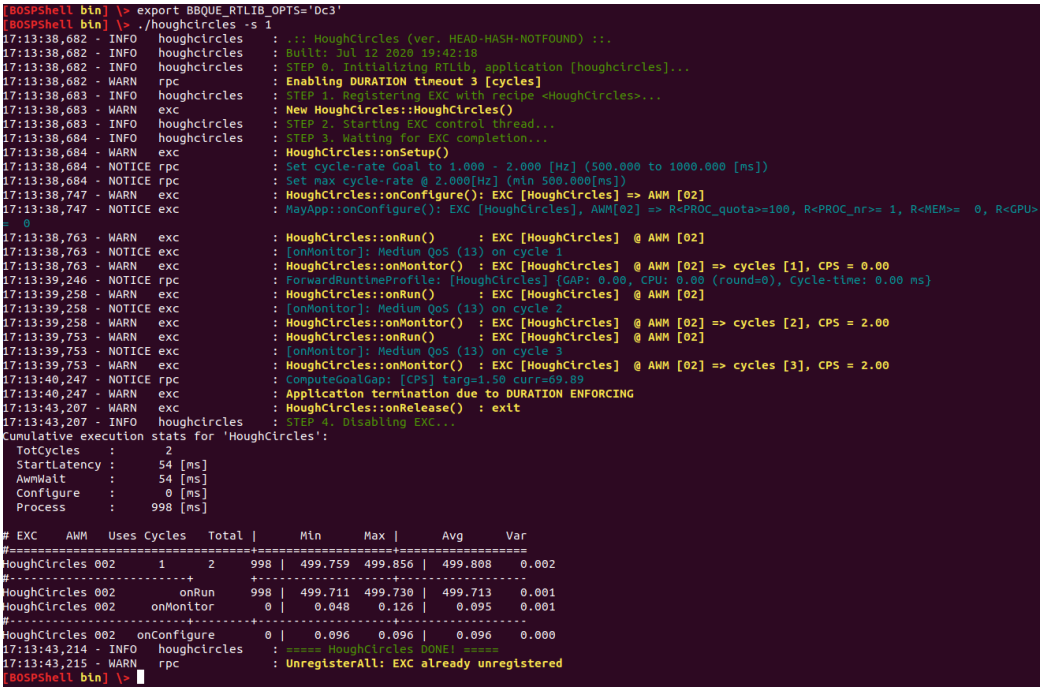


Figure 4: Managed mode with slow mode, stop after 3 cycles.

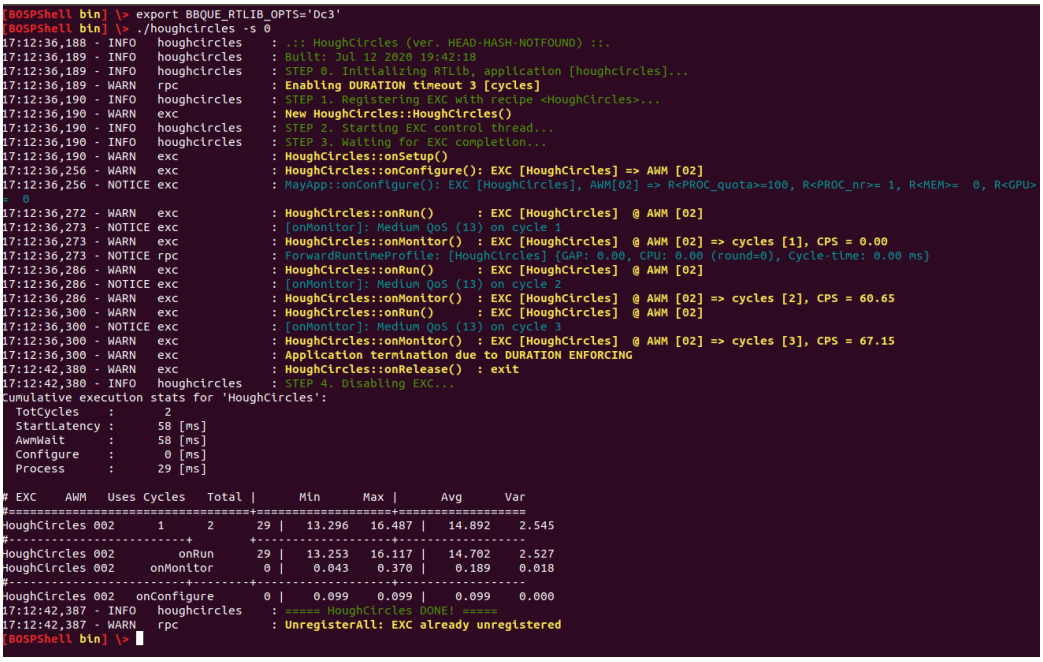


Figure 5: Managed mode without slow mode, stop after 3 cycles.

We can take a look at the impact of the number of threads on the execution:

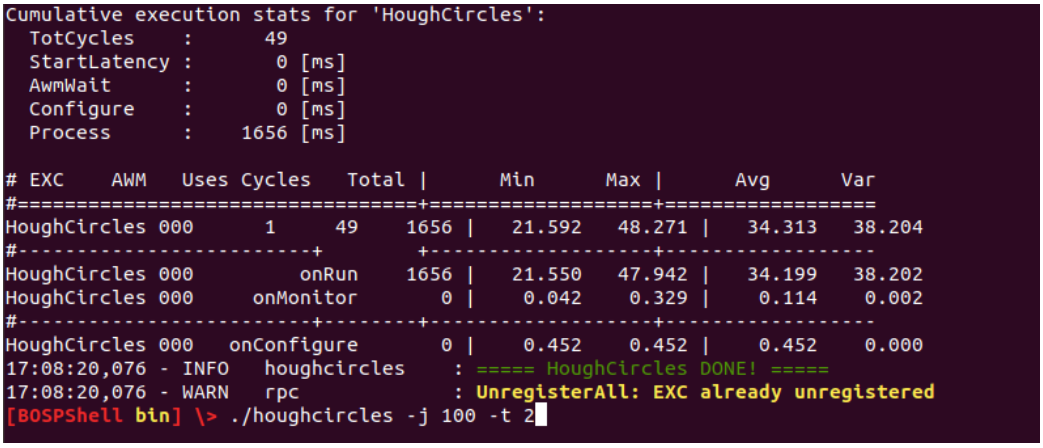


Figure 6: 2 threads to perform 100 jobs.

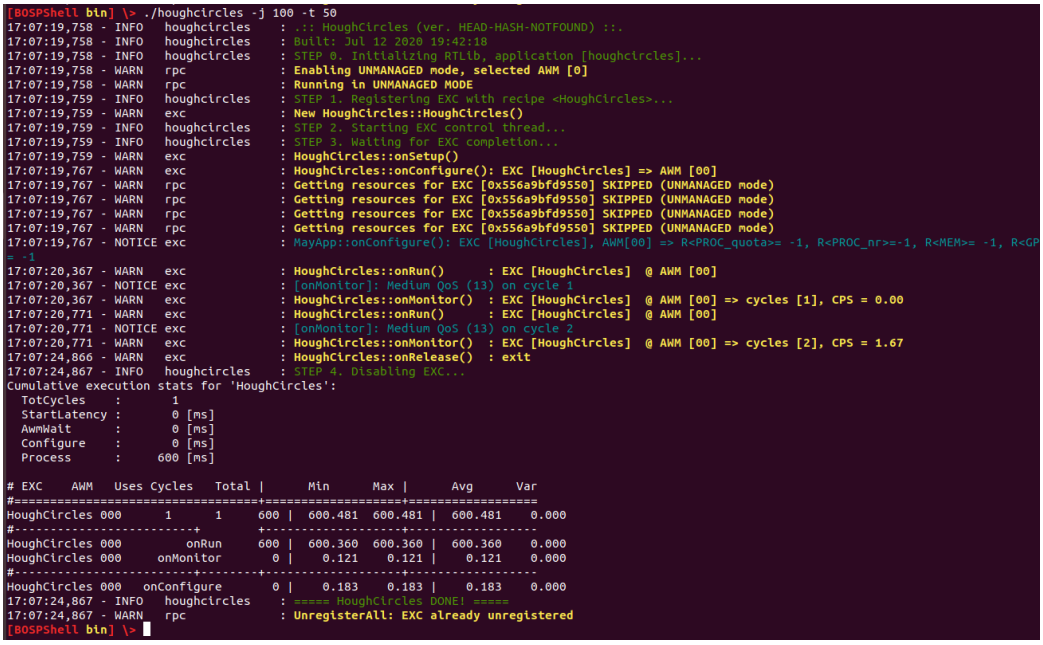


Figure 7: 50 threads to perform 100 jobs.

With 2 threads, at each cycle 2 jobs will be performed. Thus we need 50 cycles to finish the entire workload. With 50 threads, 50 jobs are performed concurrently in one cycle. After only 2 cycles, the program ends. Even though the average cycles per second is way higher with 50 threads, we can notice the processing time goes from 1656 ms to 600 ms.

We can mention the fact that this test was run on a single core machine without a GPU. Indeed we can see on Figure 5: "R<PROC_nr>= 1" and "R<GPU>= 0". If we had multiple cores available, increasing the number of threads to exploit would have a more significant effect.

We can also increase the quality of service or decrease it by increasing or decreasing the range of circles to be detected. This is done by modifying the minimum and maximum radius of the circles and the minimum distance between them:

Figure 8: High quality of service: detected circles.

```
17:19:38,312 - NOTICE exc            : [onMonitor]: High QoS (68) on cycle 20
17:19:38,312 - WARN   exc            : HoughCircles::onMonitor()  : EXC [HoughCircles]  @ A
17:21:27,852 - WARN   exc            : HoughCircles::onRelease()  : exit
17:21:27,852 - INFO   houghcircles   : STEP 4. Disabling EXC...
Cumulative execution stats for 'HoughCircles':
  TotCycles     :      19
  StartLatency  :      54 [ms]
  AwmWait       :      54 [ms]
  Configure     :       0 [ms]
  Process       :     730 [ms]

# EXC    AWM   Uses Cycles   Total |      Min      Max |      Avg      Var
#==================================+==================+==================
HoughCircles 002      1     19     730 |   27.573   56.683 |   39.076   70.192
#--------------------------+        +------------------+------------------
HoughCircles 002         onRun     728 |   27.527   54.301 |   38.740   69.899
HoughCircles 002     onMonitor       2 |    0.047    2.382 |    0.336    0.293
#-------------------------+--------+------------------+------------------
HoughCircles 002   onConfigure       0 |    0.163    0.163 |    0.163    0.000
17:21:27,856 - INFO   houghcircles   : ===== HoughCircles DONE! =====
17:21:27,856 - WARN   rpc            : UnregisterAll: EXC already unregistered
[BOSPShell bin] \>
```

Figure 9: High quality of service: performance results.

Figure 10: Low quality of service: detected circles.

```
17:23:00,732 - NOTICE exc             : [onMonitor]: Low QoS (-22) on cycle 20
17:23:00,732 - WARN   exc             : HoughCircles::onMonitor()  : EXC [HoughCircle
17:23:46,181 - WARN   exc             : HoughCircles::onRelease()  : exit
17:23:46,181 - INFO   houghcircles    : STEP 4. Disabling EXC...
Cumulative execution stats for 'HoughCircles':
  TotCycles     :      19
  StartLatency  :      57 [ms]
  AwmWait       :      57 [ms]
  Configure     :       0 [ms]
  Process       :     303 [ms]

# EXC     AWM    Uses Cycles    Total |      Min      Max |      Avg      Var
#=================================+==================+==================
HoughCircles 002      1     19     303 |   12.218   23.958 |   16.488    7.137
#-------------------------+        +------------------+------------------
HoughCircles 002         onRun     303 |   12.174   23.691 |   16.415    7.134
HoughCircles 002     onMonitor       0 |    0.043    0.266 |    0.073    0.003
#-------------------------+--------+------------------+------------------
HoughCircles 002   onConfigure       0 |    0.286    0.286 |    0.286    0.000
17:23:46,193 - INFO   houghcircles    : ===== HoughCircles DONE! =====
17:23:46,193 - WARN   rpc             : UnregisterAll: EXC already unregistered
[BOSPShell bin] \>
```

Figure 11: Low quality of service: performance results.

We notice a degradation of the quality of service between the first case

where the quality of service is high, 68, and almost all the circles in the image are detected and the second case where the quality of service is very low, -22, and only 5 circles are detected.
We can also mention the fact that in the first case, circles with a radius between 0 and 100 were searched, whereas in the second case, circles with a radius between 10 and 20 were searched. This explains why the first search (High QoS) took longer than the second one (Low QoS).

For the final test we will use the same parameters as the High QoS search but we will lower the threshold for center detection from 30 to 1:

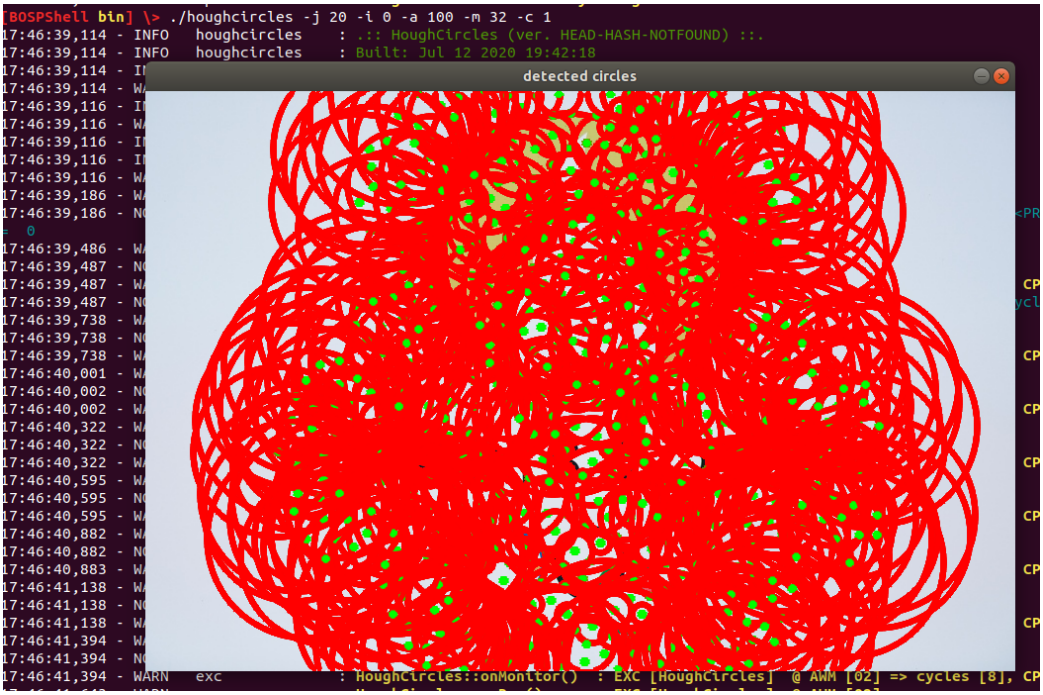


Figure 12: Low center_threshold: detected circles.

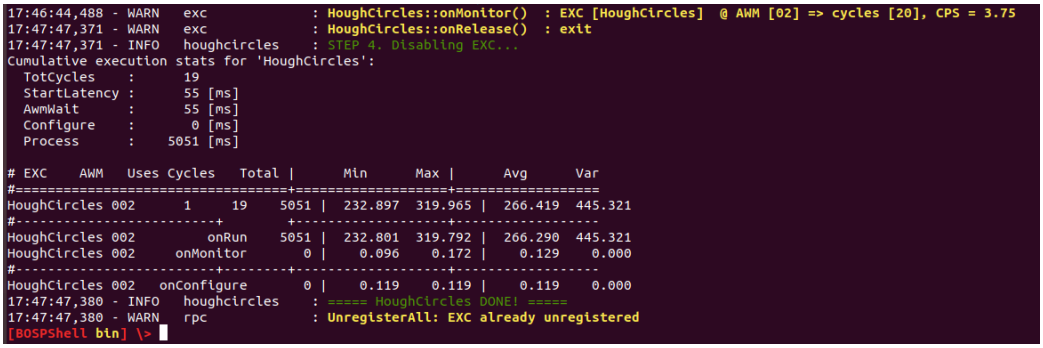


Figure 13: Low center_threshold: performance results.

There are obviously a lot of false positives in the detected circles. We can also see that the program took more time to finish. If we wanted to characterize the application in order to optimize it for a specific system, we could run a script to run the application with different parameters and choose the best combination available.

4 Conclusion

This project has shown how to port and run an existing OpenCV application to BOSP. This has allowed use to make use of the functionalities offered by the Barbeque resource manager such as getting detailed information on the execution of the application in real time and after the execution. Testing the application with different configurations was pretty straightforward and we can imagine that this framework can be really useful with more complex programs running on devices with various computing resources.