Raspberry Pi: GPU Support

Sander Vocke, 24-04-2016
5LIA0 – Embedded Visual Control

2015 EVC group:

Jumana Mundichiparakkal,

Praveen Pujari,

Sethu Jayaraman

Overview

- **Video:** last year result for EVC

- **Why** would you use the RPi's GPU for vision?
- **How** can it be programmed?
- **Example** of a Raspberry Pi GPU+CPU pipeline

- **Where** can you start?

2015 EVC Project:: Tom ‘n Jerry

360-degree
Camera lens:

2015 EVC Project:: Tom ‘n Jerry

- **The 360-degree lens** meant we needed **high resolution** to see anything. (2048x232px)
- We were noticing **low framerates** on the RPi, even at **low resolution**.

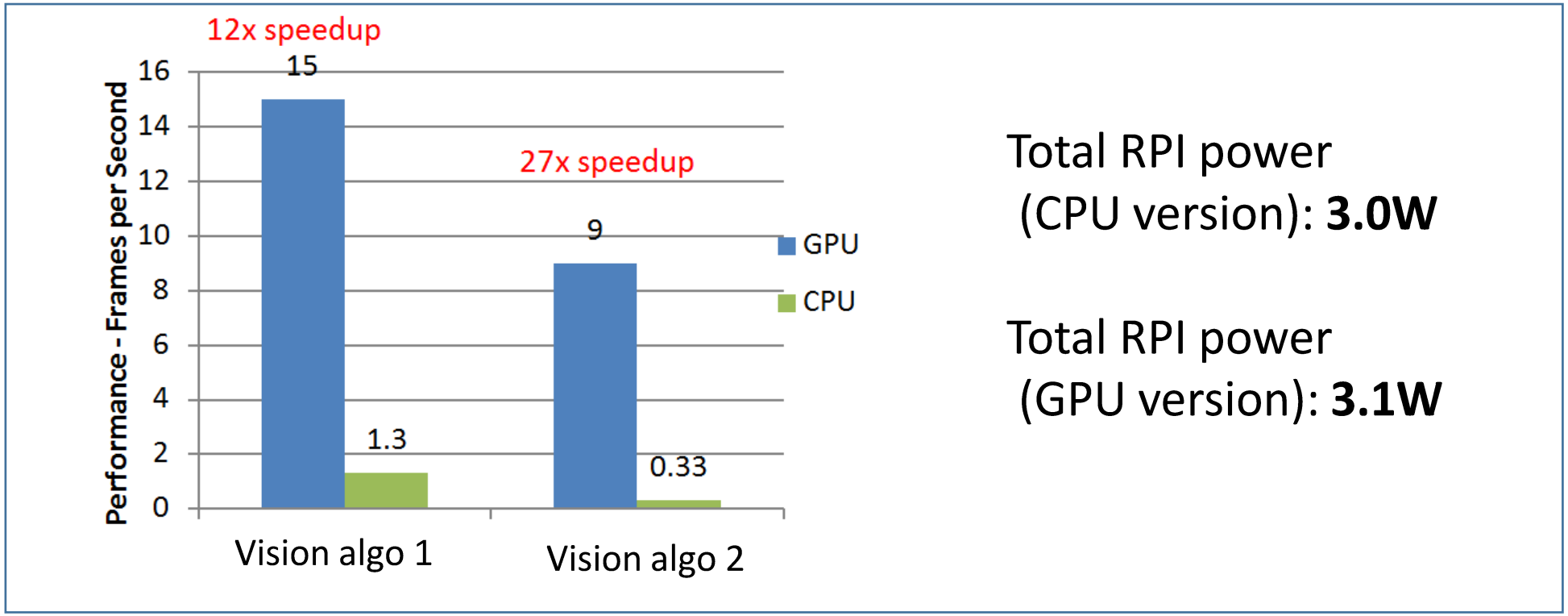
- That’s when we decided to look at **GPU support**.

Raspberry Pi VideoCore GPU

- Purpose: **High-speed video processing @ low power**.
- Typical applications: **video en-/decoding**
- GPU has **4 Processing Subsystems**
  - Each**: 4 Quad Processors (QPU's)**
    - Each: **2 4-way SIMD floating point ALUs**
- **So… A lot of parallel processing power!**
- **And… on 400MHz now** (250MHz on Pi 2)

And… The advantage over CPU is real.

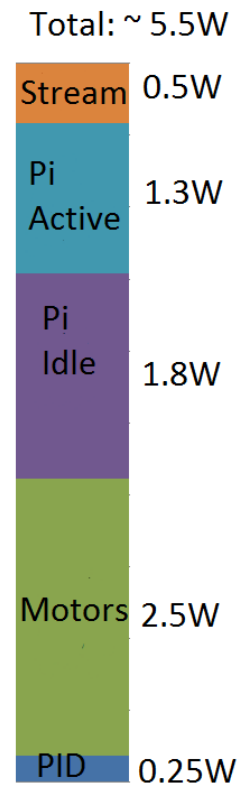
- Last year, our results using GPU, compared to using CPU:



Total RPI power (CPU version): **3.0W**

Total RPI power (GPU version): **3.1W**

Total robot power:



…So, how do we use it?

…So, how do we use it?

- Typical CPU programming methods are **not available**, like:
  - **C/C++, Python, Java**…

And, unfortunately:

That leaves us with:

- **Open Graphics Library. → for rendering (3D) graphics.**

- Takes commands from CPU program and executes them.
- Can run **custom programs**, referred to as **shaders**, which:
  - **May run on GPU**
  - Are programmed in the **GLSL** (OpenGL Shader Language)
  - Operate on images (called **textures** in OpenGL) or 3D models

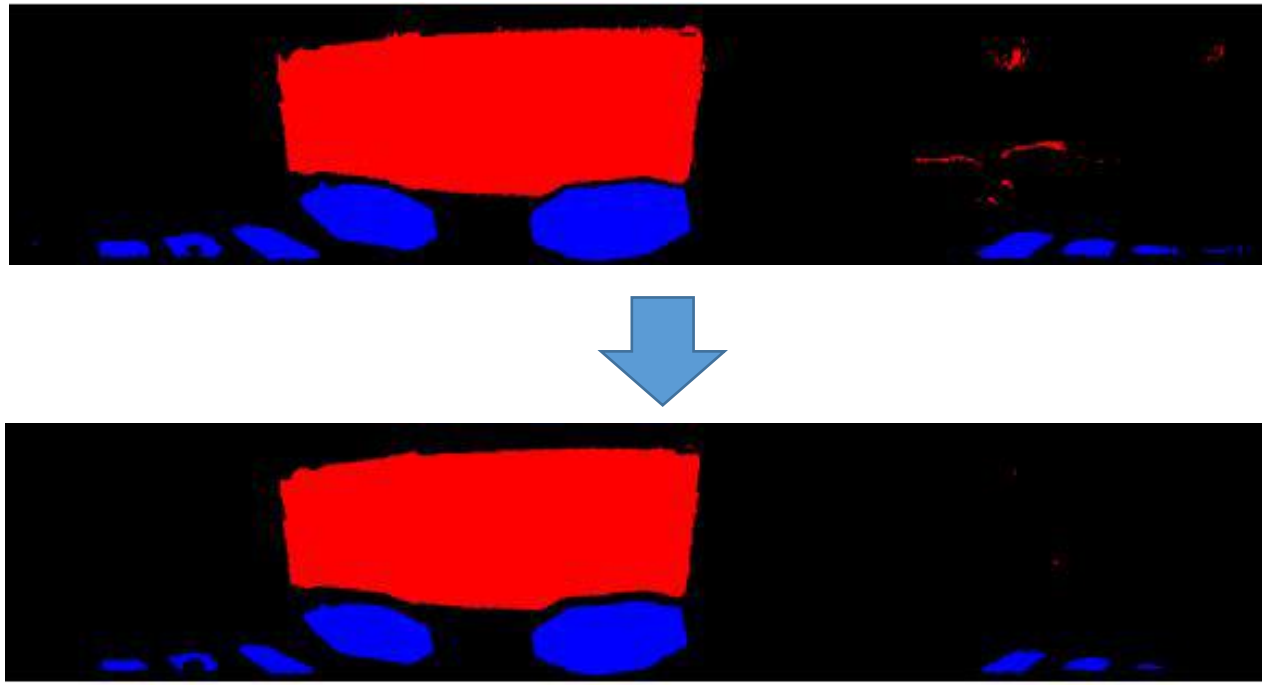
- **GLSL** is **abstract**, **implicitly parallel.**

- The type of shader program we are interested in: **Fragment Shaders**.

Basically:

- A **C-like program**
- Inputs are images (**textures**), plus extra parameters
- Output is the **color of a single pixel at some coordinate**
- OpenGL executes the shader **for every output pixel required**.

- Useful for “pixel-crunching” (for example: **filters**).



Example: Erosion Shader

```
varying vec2 tcoord;
uniform sampler2D tex;

void main(void)
{
        vec4 col = vec4(0);

        for(int xoffset = -3; xoffset <= 4; xoffset++)
                for(int yoffset = -3; yoffset <= 4; yoffset++){
                        vec2 offset = vec2(xoffset, yoffset);
                        col = min(col, texture2D(tex, tcoord+offset));
                }

   gl_FragColor = clamp(col,vec4(0),vec4(1));
}
```

- **Advantages:**
  - Easy to manipulate pixels
  - OpenGL library handles parallel execution scheduling

- **Disadvantages:**
  - Some things are very **hard/impossible to express** in this language:
    - Histogramming
    - Anything that has a data-dependency on neighbouring pixel outputs

- The Raspberry Pi 2 supported only a special OpenGL version:

- This imposed even more restrictions on shaders:
  - (almost) **no if-statements**
  - Strict, very **small limits on code size**, **loop iteration counts**, **memory usage**
  - Getting the OpenGL ES drivers to execute anything is **tedious**

The Good News

- The Raspberry Pi community has not been idle!
- February 2016: **Raspbian OS released with OpenGL 2.1 support!**
- That means “full” OpenGL can now be used **just like on a PC***.
  - For example: using the **PyOpenGL** Python module (?)

The Good News

- Example:

- Because OpenGL ES was so tedious:
  We had a PC app to test our shaders before using them on the Pi (OpenGL).

- With the new Raspbian, it should be possible to **run that Python app directly on the Pi…**

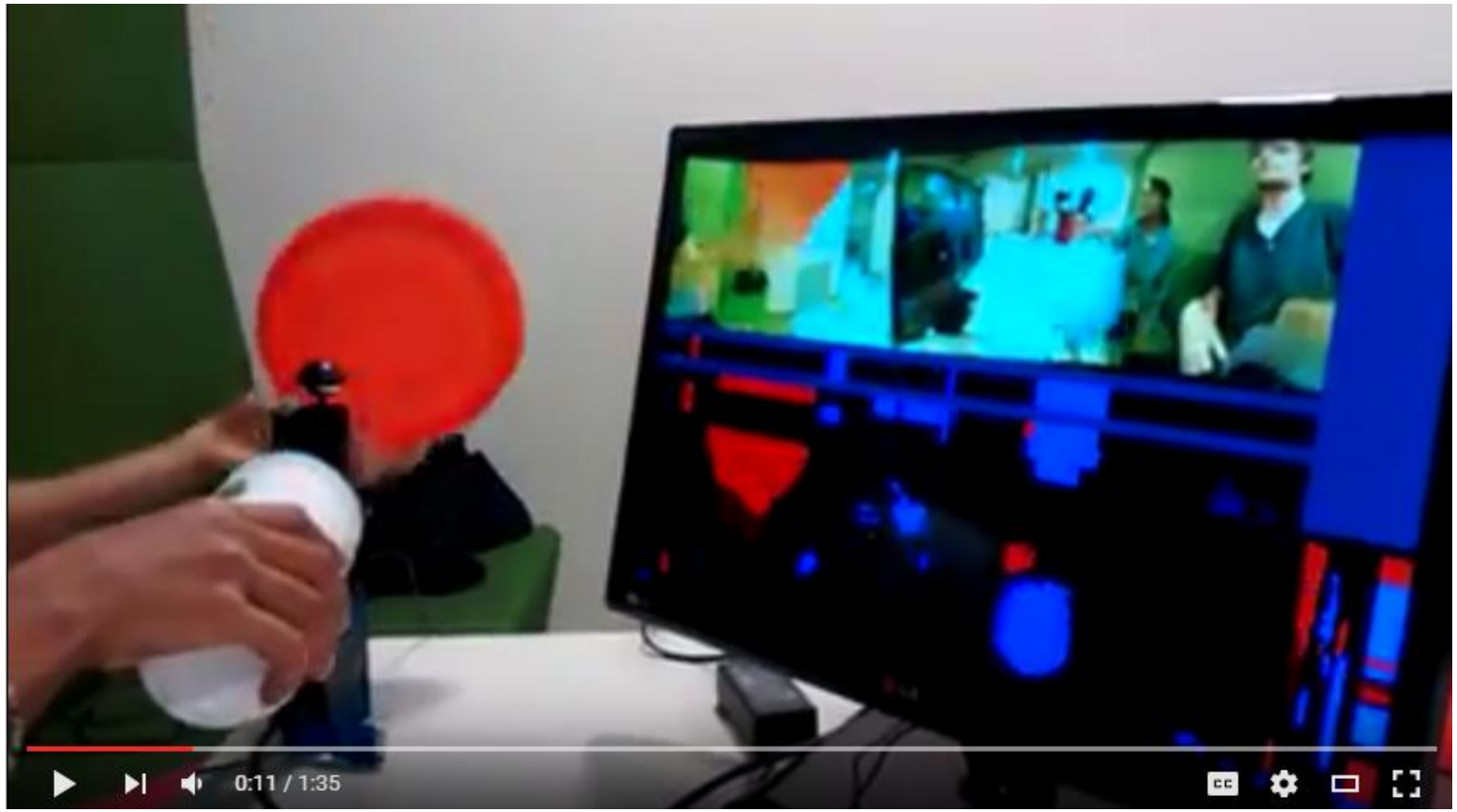
The ? news

- Raspbian release notes:

“this experimental driver may break Raspberry Pi Camera (…) support”

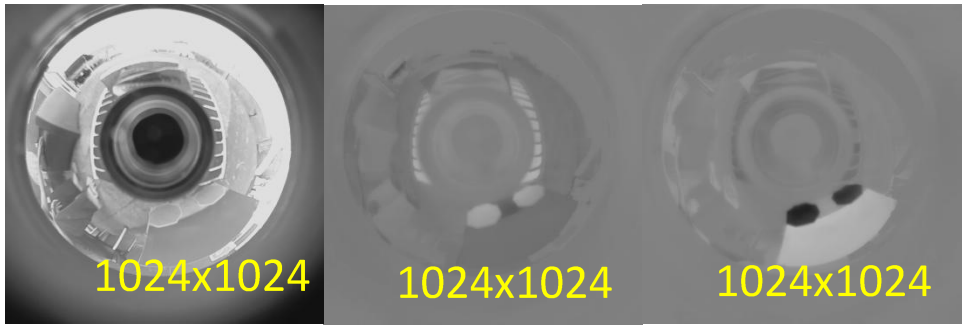
Example: A GPU+CPU pipeline

Video



Pipeline-1

1

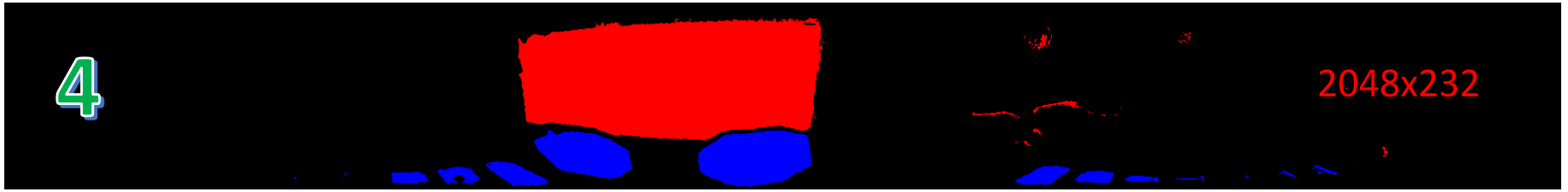


2

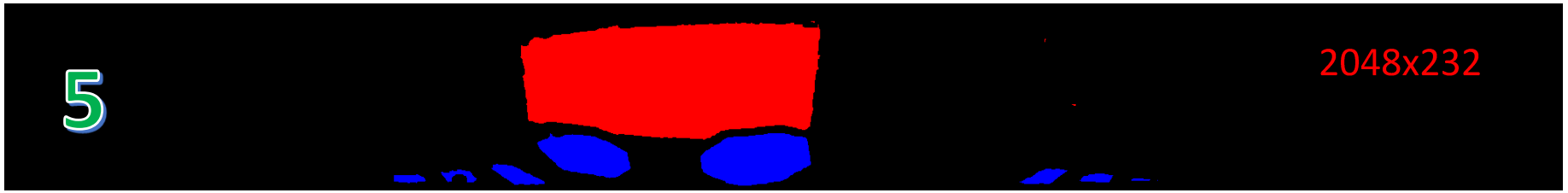
COLOR DETECTION

3

4



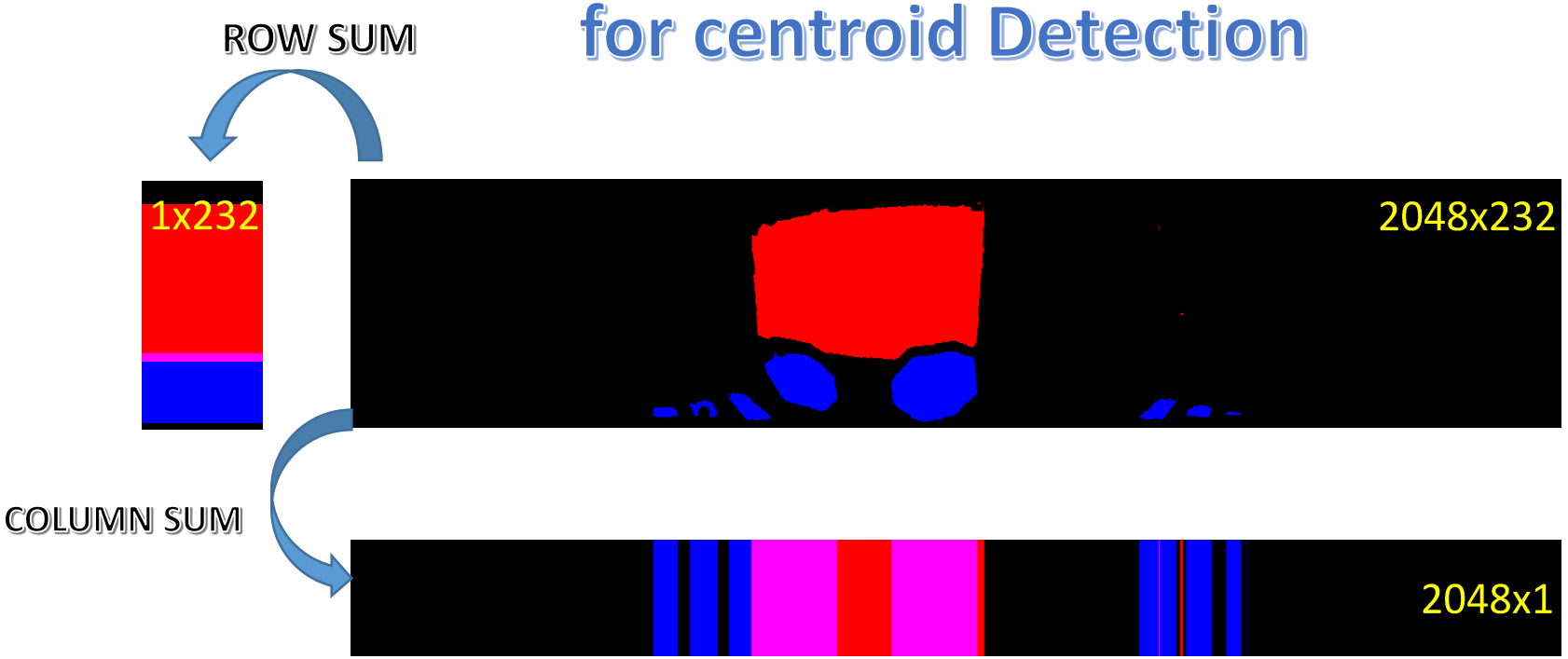
5



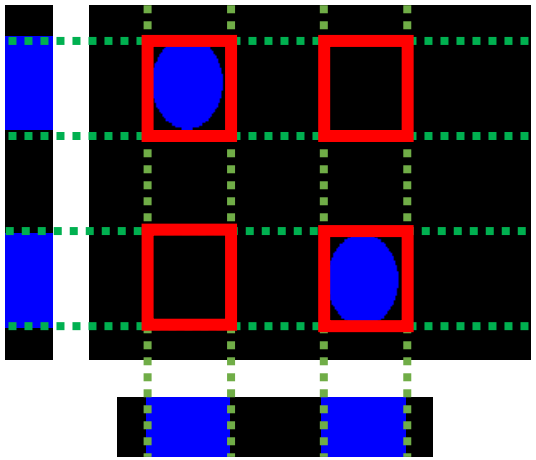
Step No.	Algorithm
1	YUV IMAGES FROM CAMERA
2	YUV-RGB
3	“DE-DONUT”
4	THRESHOLD
5	MORPHOLOGICAL OPERATIONS (ERODE, DILATE)

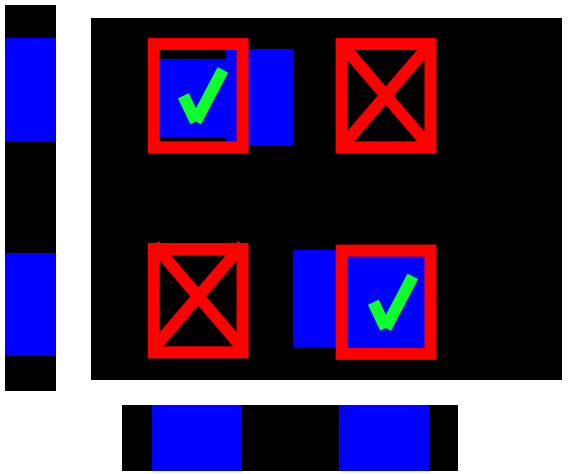
Pipeline-2

Pixel Summation for centroid Detection

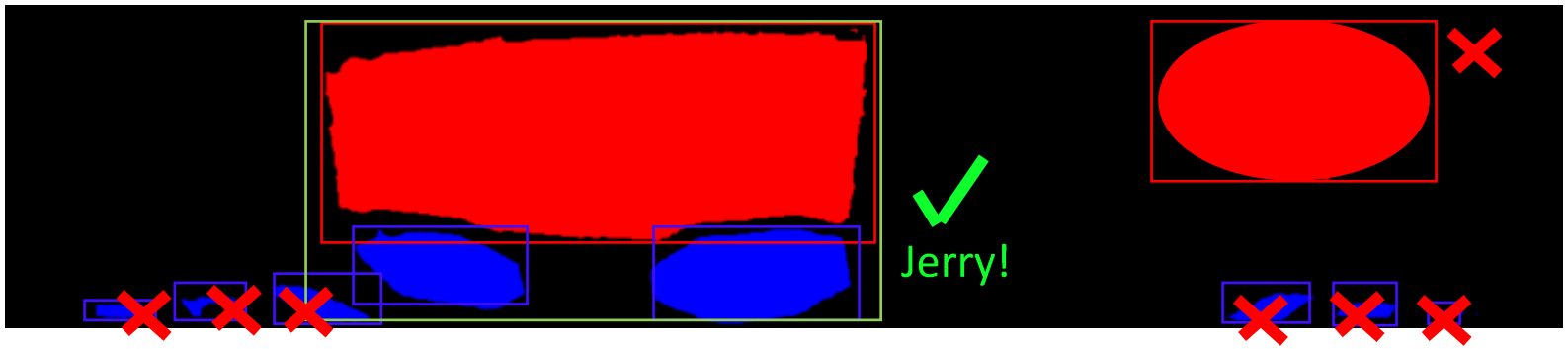


Object ID



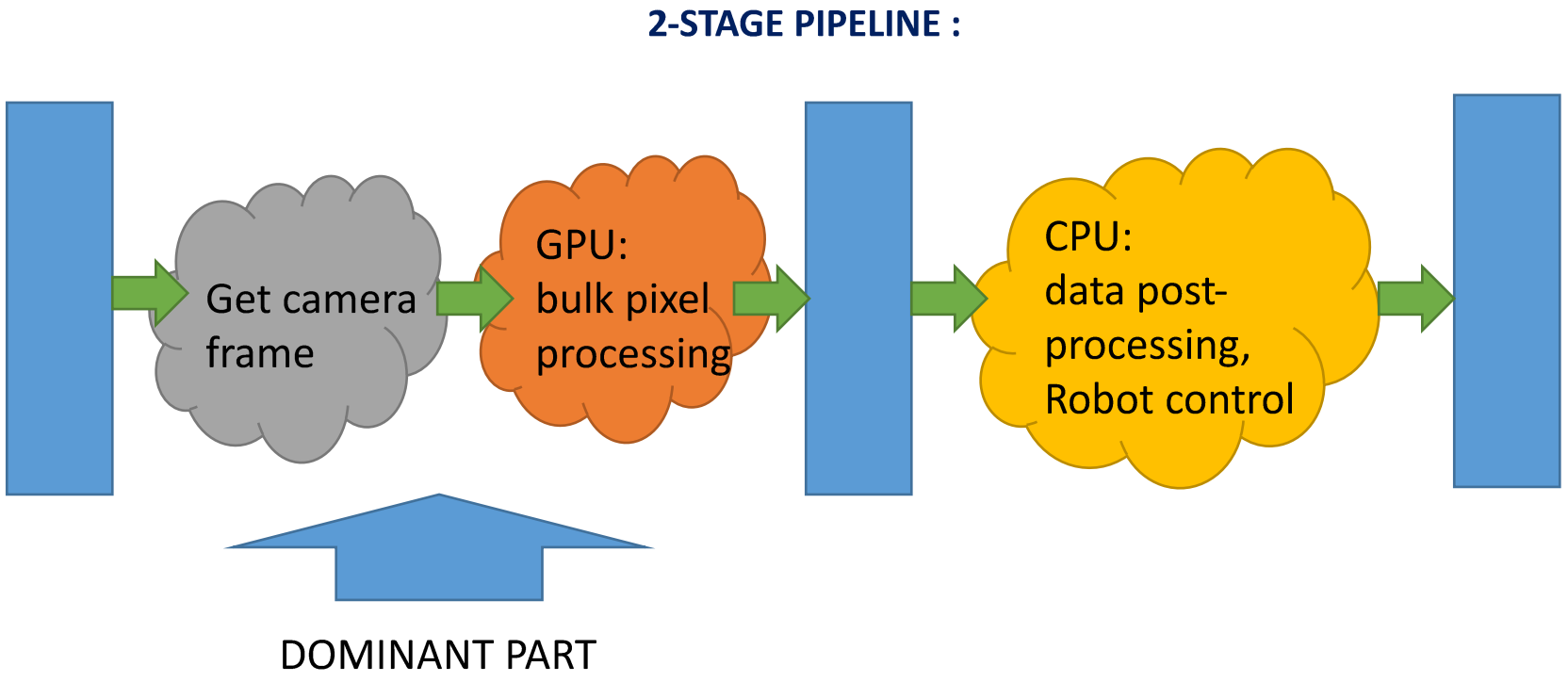


Target Detection



GPU+CPU Cooperation

- Using the GPU for the “hard work” frees up the CPU.



Where to start?

- New OpenGL support means **many Linux OpenGL code examples should now work on Pi**.
  - The OpenGL driver needs to be enabled first

- A tool for writing, using, testing GLSL shaders:

  GLSL Hacker ← supports OpenGL, OpenGL ES modes.

- Or by looking at our code from last year:

https://github.com/SanderVocke/picamgpu_minimal
https://github.com/SanderVocke/Pi_Cam_GPU_Processing
https://github.com/SanderVocke/OpenGL_Pi_Tester

Where to start?

- You can also look at our code from last year:

https://github.com/SanderVocke/picamgpu_minimal
(Minimal example of getting camera data to OpenGL)

https://github.com/SanderVocke/Pi_Cam_GPU_Processing
(Our full robot control code of last year, including OpenGL ES pipe)

https://github.com/SanderVocke/OpenGL_Pi_Tester
(PC application to test OpenGL shaders more easily)

Where to start?

One more link:

- “GPU Accelerated Camera Processing On The Raspberry Pi”
  (where we started our code from last year)

  http://robotblogging.blogspot.co.uk/2013/10/gpu-accelerated-camera-processing-on.html