Exercise 5. Reconstructing Convolved Images

1 (Temporal) Integration of Events and Spatial Convolutions

This is an exercise that we will do collaboratively in one session, building up on Exercise 4. The idea is to extend the image reconstruction method [1] (leaky integrator in Exercise 4) to perform other operations, such as horizontal or vertical edge detection, Laplacian calculation or corner detection. We are familiar with some of these operations because we have seen them working on images in Exercise 1 (Section 2.6): they are performed using spatial convolutions.

This exercise is based on [2]. *Please take a look at Fig. 1 in the paper and read Section III in advance.* The first row of Fig. 1 in the paper refers to image reconstruction from events (Exercise 4 - leaky integrator). In Exercise 5 we want to modify the code we have implemented in Ex. 4 to obtain rows 2 to 5 of such Fig. 1.

The **main idea** is the following. Once we have reconstructed an image from events, we could filter it using a spatial kernel, as shown in Fig 1 (in this document). The alternative that is proposed in [2] is to directly reconstruct the final convolved image (vertical edges in Fig. 1 because a Sobel-x convolution kernel is used), continuously updating it as new events arrive (i.e., event-by-event style of processing). This is possible because convolution and integration are linear operations, i.e., they commute, and so we may convolve before integration (as illustrated in Fig. 2). The key question is then... what does it mean to convolve an event with a kernel mask? (red block in Fig. 2)

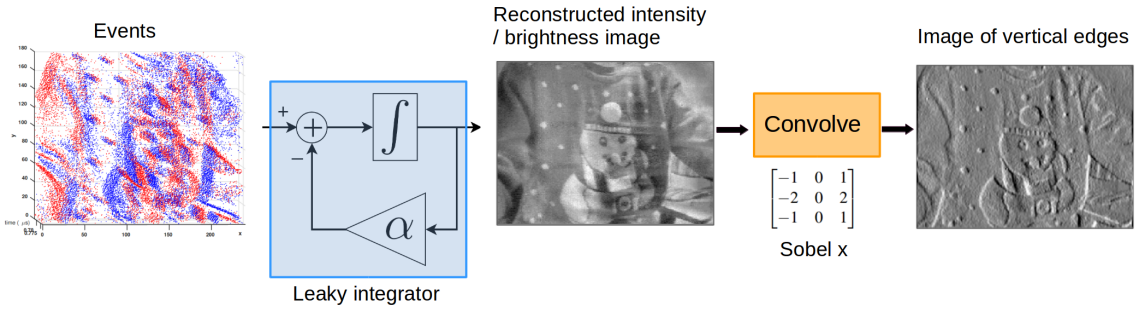

Figure 1: Traditional approach: applying convolution (Ex 1) to the reconstructed image (Ex 4).

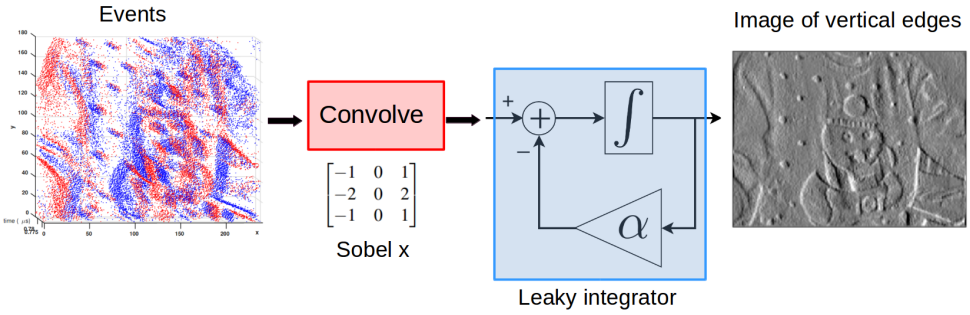

Figure 2: Approach in [2] (Ex 5): applying convolution before the leaky integrator. But what does it mean to convolve events with a kernel mask?

2 Practical Considerations:

Supporting slides are available in the references [3].

Some guiding **questions**:

- What do we need as "internal variables" (i.e., "state") of our method?
- How are these variables updated as we loop through the events? That is, what do we have to change (with respect to the code in Exercise 4), *if needed*, to:
  - update the pixels of the state variables?
  - perform the leaking of the state pixels affected by the kernel mask?
  - compute the output?
- **Remark**: Note that if we use the identity kernel (first row of Fig. 1 in [2]), we should obtain the reconstructed image from Exercise 4... so perhaps we can implement the new functionality that we want in a way that we do not need to change much of the structure of the code we already have.

What to do in this exercise?

- Modify your code (e.g., the event callback function) to include the "new block" (red in Fig. 2).
- Use a pool of predefined kernel masks and switch to convolving with one of them using a parameter, such as the ones used in the dynamic reconfigure GUI (Graphical User Interface). Start easy: first try with only one mask, such as:
  - `const cv::Mat sobel_x = (cv::Mat_<double>(3,3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);`
  - `const cv::Mat sobel_y = (cv::Mat_<double>(3,3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);`

Hints:

- To select a region of an image (or to crop an image) take a look at the OpenCV function cv::Rect(). Example.
- Make sure you do not select a region out of bounds. Leave a margin of unprocessed pixels around the image boundary if needed.
- Try to make sense of your results as you code.
  - Are the Sobel-x and Sobel-y kernels highlighting the edges that you expect (vertical or horizontal, respectively)?
  - What data sequence (ROS bag) would you choose for better prototyping?

Remark: Please note that the word "filter" may refer to a spatial filter (Sobel filter) as well as a temporal filter (the so called "high pass filter" in [1]). The disambiguation should be clear from the context. In this exercise, we perform filtering in both space (the xy image plane of the camera, using convolution kernels) and time (using leaky integration).

3 What to submit?

- **Source code** (ROS Node package), like in Exercise 4. So that it can be copied into a catkin workspace, compiled and executed (tested).
- **Video(s)** (screen capture, for example using SimpleScreenRecorder). See the three example videos provided on ISIS (and Fig. 3).
  - Show all 4 displayed data (events alone -dvs_displayer-, frames, event timestamp map, output convolved brightness).
  - Show also how you interact using the dynamic reconfigure GUI and/or the command line (e.g., if you pause while playing the bag). Show each convolution kernel that you test. Example: one loop through the 3planes sequence for each convolution kernel (see the provided videos).
  - Test on three different sequences, not all simulated. For example, planes_simulation, slider depth, bicycle, etc. Feel free to choose other sequences or sub-clips of sequences.
  - Adjust the cut-off frequency parameter (α) for each sequence, to show some meaningful event integration (because the same α may not be good for all sequences). There is no need to choose the exact same parameters that are shown in the provided videos.
- You may speed up the video using ffmpeg (ffmpeg static builds require no installation). For example, to double the speed of the video:
  - `ffmpeg-4.2.2-i686-static/ffmpeg -i input.mp4 -filter:v "setpts=0.5*PTS" output.mp4`

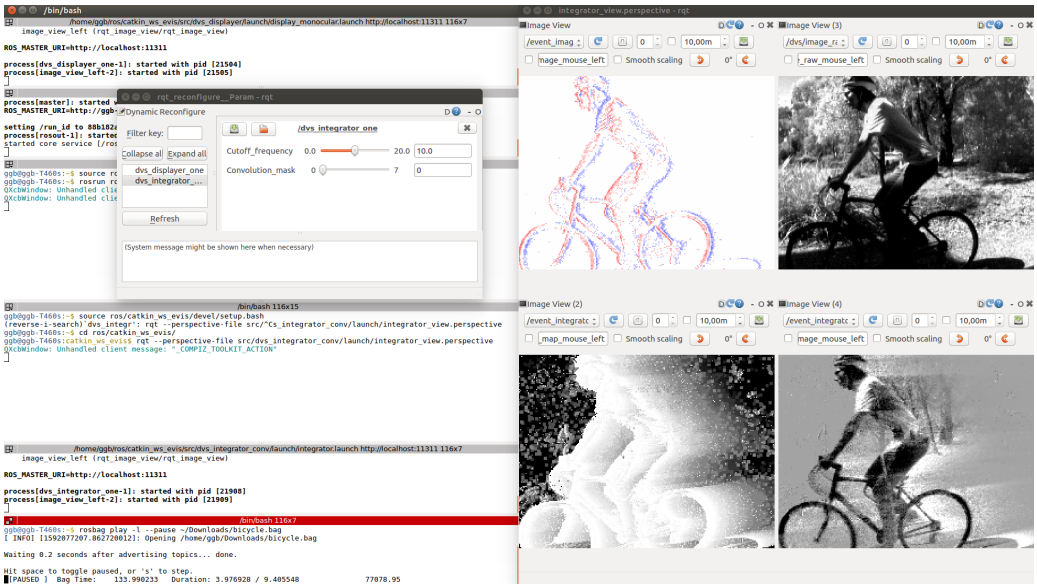

Figure 3: Screenshot of possible video configuration.

References

[1] Scheerlinck et al., Continuous-time Intensity Estimation Using Event Cameras, Asian Conf. Computer Vision (ACCV), 2018.

[2] Scheerlinck et al., Asynchronous Spatial Image Convolutions for Event Cameras, Robotics and Automation Letters, 2019.

[3] Scheerlinck et al. Slides about [2].