Pedestrian Tracking & Corner Detection of An Image Using NXP S32V234 Vision Processor

ECE 59500
Autonomous Embedded Systems

Submitted by:

Abir Saha

Raj Jatin Shah

Md Saiful Islam

IUPUI
INDIANA UNIVERSITY–PURDUE UNIVERSITY INDIANAPOLIS
PURDUE SCHOOL OF ENGINEERING & TECHNOLOGY

Introduction:

Face recognition is one of the key components for future intelligent vehicle applications such as determining whether a person is authorized to operate the vehicle.

To detect vehicles or any other object we need to know what differentiates them from the rest of the image captured by the camera. Colors and gradients are good differentiators, but the most important features will depend on the appearance of the objects.

Vision-based face and corner detection is a core part of making car autonomous.

S32V234 offers an ISP, powerful 3D GPU, dual APEX-2 vision accelerators, ASIL-B security and supports Safe Assure and suited for ADAS, NCAP front camera, object detection and recognition, surround view, machine learning and sensor fusion applications. Especially due to the extensive information an image can hold and the decreasing costs of computational power, it has attracted increasingly interest in the last decade. Since fully autonomous driving needs to take decision, maintain Vehicle-to-vehicle (V2V) and Vehicle-to-infrastructure (V2I) so it is important to ensure that the car can detect object using face and corner around it using sensor fusion technology. Moreover, car can use this technique as an extra caution while crossing pedestrian crossing detecting face of pedestrian and corner of image to detect zebra crossing on road.

In this project, we are using face detecting and edge detecting algorithms with S32V234 Vision Processor for detecting face and corner of an image. Car having captured images will process it using S32V234 Vision Processor and detect face and corner which will be taken as input to take decisions.

Hardwire Specifications and Technical Details:

- NXP S32V234 board Evaluation Board
- S32 Design Studio for Vision v2.0 IDE
- Micro SDHC card
- USB Micro B to USB A cable, Ethernet to USB cable
- Power supply for NXP S32V234 board
- Laptop

System Block Diagram:

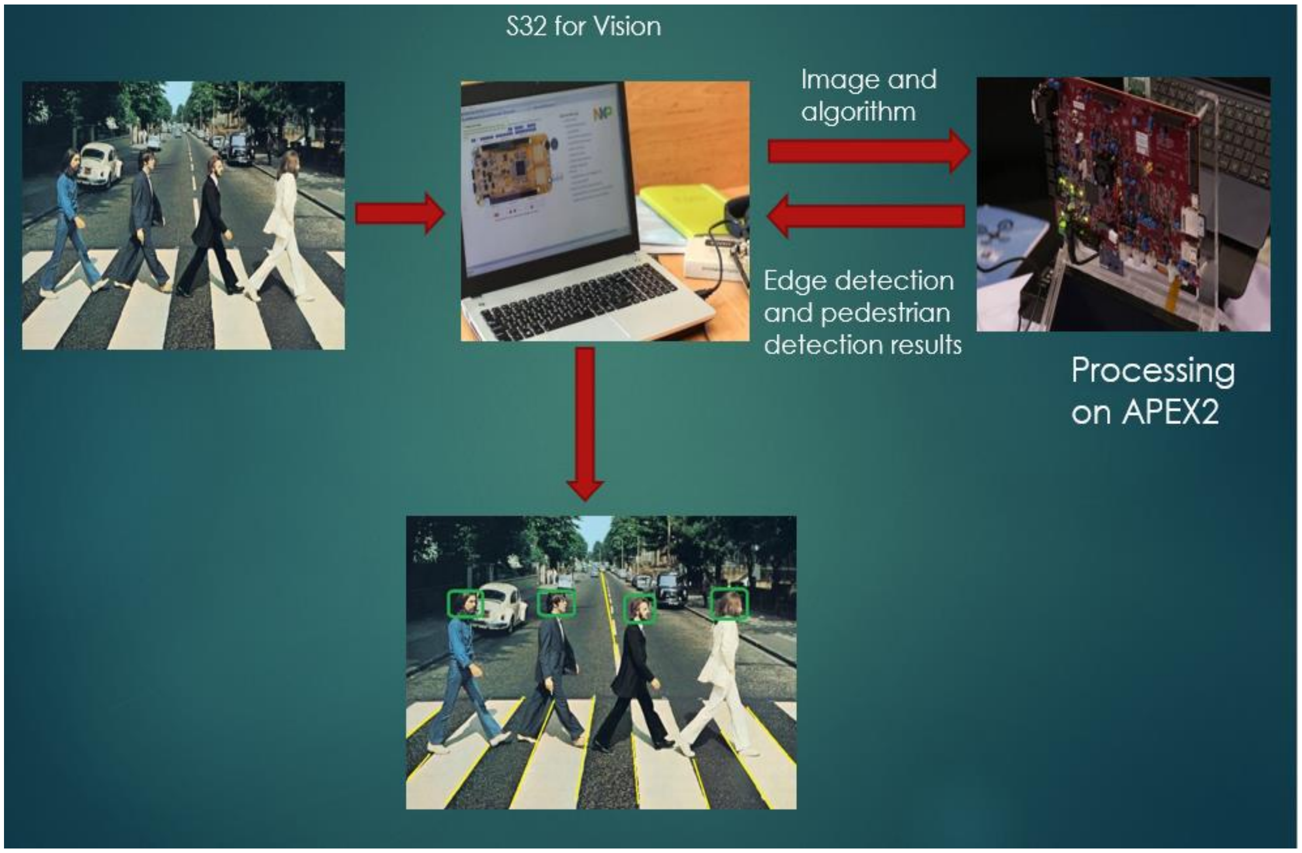


Figure 1: Block diagram of the system

NXP S32V234EVB Evaluation Board:

The S32V234 is our 2nd generation vision processor family designed to support computation intensive applications for image processing and offers an ISP, powerful 3D GPU, dual APEX-2 vision accelerators, security and supports Safe Assure. S32V234 is suited for ADAS, NCAP front camera, object detection and recognition, surround view, machine learning and sensor fusion applications. S32V234 is engineered for automotive-grade reliability, functional safety and security measures to support vehicle and industrial automation.

S32V234 has a complete enablement platform supported by S32 Design Studio IDE for Vision which includes a compiler, debugger, Vision SDK, Linux BSP, and graph tools.

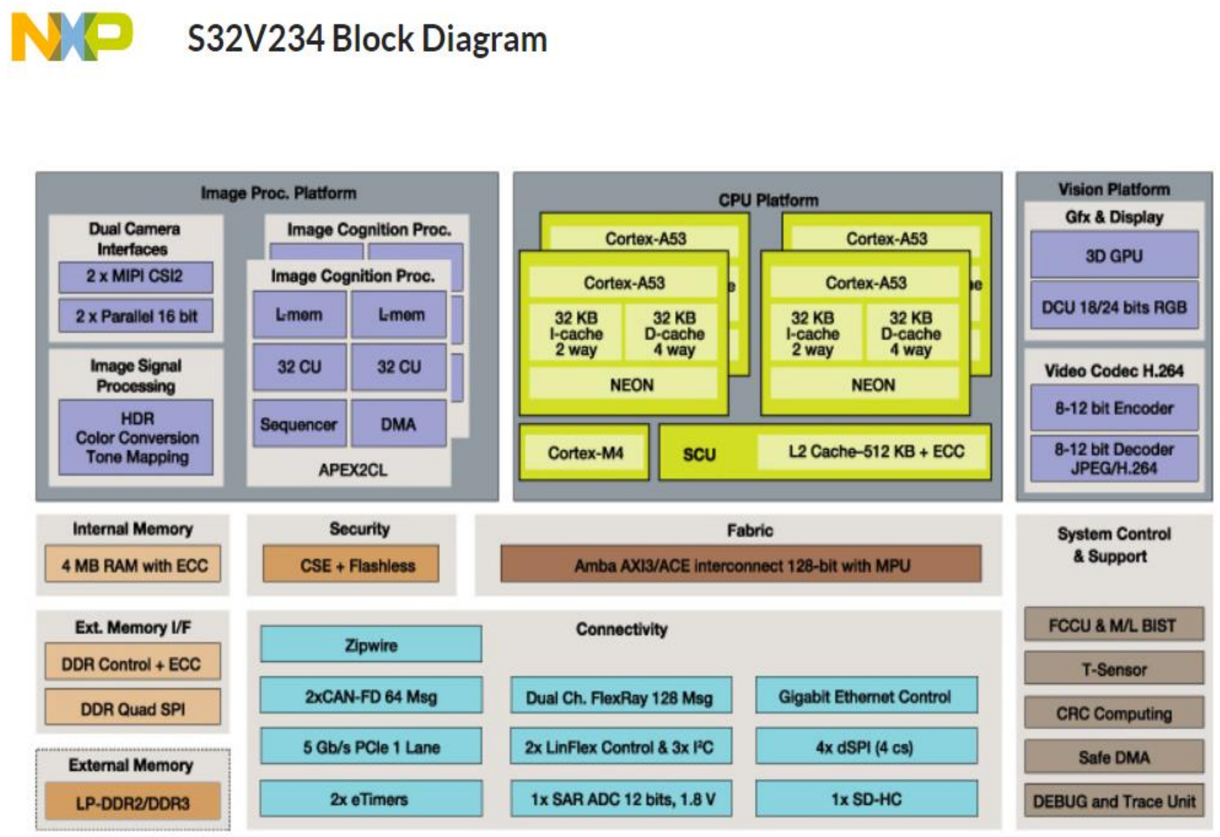


Figure 2: S32V234EVB Evaluation Board

Features:

- Quad Arm® Cortex®-A53 cores running up to 1GHz, Plus M4 core up to 133 MHz
- Dual APEX-2 vision accelerator cores enabled by OpenCL™, APEX-CV and APEX graph tool
- Supports ISO 26262 functional safety up to ASIL-C, IEC 61508 and DO 178 applications
- 3D GPU (GC3000) with OpenCL 1.2 EP 2.0, OpenGL ES 3.0, OpenVG 1.1
- Hardware security encryption on CSE2
- Embedded image sensor processing (ISP) for HDR, color conversion, tone mapping, etc. enabled by ISP graph tool
- MIPI CSI2 and parallel image sensor interfaces
- 4 MB on-chip system RAM with end to end ECC on all memories
- DRAM controller support for DDR3 / DDR3L / LPDDR2
- -40 to 125 °C (junction temperature) operation
- 621 FC-BGA 17 mm x 17 mm, 0.65 mm pitch

Face Detection technique:

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene. Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars. Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will invalidate the matching process.

There are several algorithms for face detection where we used ORB (Oriented FAST and Rotated BRIEF) for face detection.

1. Haar-Cascade Algorithm
2. ORB (Oriented FAST and Rotated BRIEF) Algorithm
3. Local Binary Pattern (LBP) Algorithm

ORB is basically a fusion of FAST key point detector and BRIEF descriptor with many modifications to enhance the performance. First it uses FAST to find key points, then apply Harris corner measure to find top N points among them. It also uses pyramid to produce multiscale-features. But one problem is that, FAST doesn't compute the orientation. So, what about rotation invariance? Authors came up with following modification. It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation. To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r, where r is the size of the patch.

Now for descriptors, ORB use BRIEF descriptors. But we have already seen that BRIEF performs poorly with rotation. So what ORB does is to "steer" BRIEF according to the orientation of key points. For any feature set of n binary tests at location (xi, yi), define a 2 \times n matrix, S which contains the coordinates of these pixels. Then using the orientation of patch, \theta, its rotation matrix is found and rotates the S to get steered(rotated) version S_\theta.

ORB discretize the angle to increments of 2 \pi /30 (12 degrees) and construct a lookup table of precomputed BRIEF patterns. If the key point orientation \theta is consistent across views, the correct set of points S_\theta will be used to compute its descriptor.

BRIEF has an important property that each bit feature has a large variance and a mean near 0.5. But once it is oriented along key point direction, it loses this property and become more distributed. High variance makes a feature more discriminative since it responds differentially to inputs. Another desirable property is to have the tests uncorrelated, since then each test will contribute to the result. To resolve all these, ORB runs a greedy search among all possible binary tests to find the ones that have both high variance and means close to 0.5, as well as being uncorrelated. The result is called rBRIEF.

For descriptor matching, multi-probe LSH which improves on the traditional LSH, is used. The paper says ORB is much faster than SURF and SIFT and ORB descriptor works better than SURF. ORB is a good choice in low-power devices for panorama stitching etc.

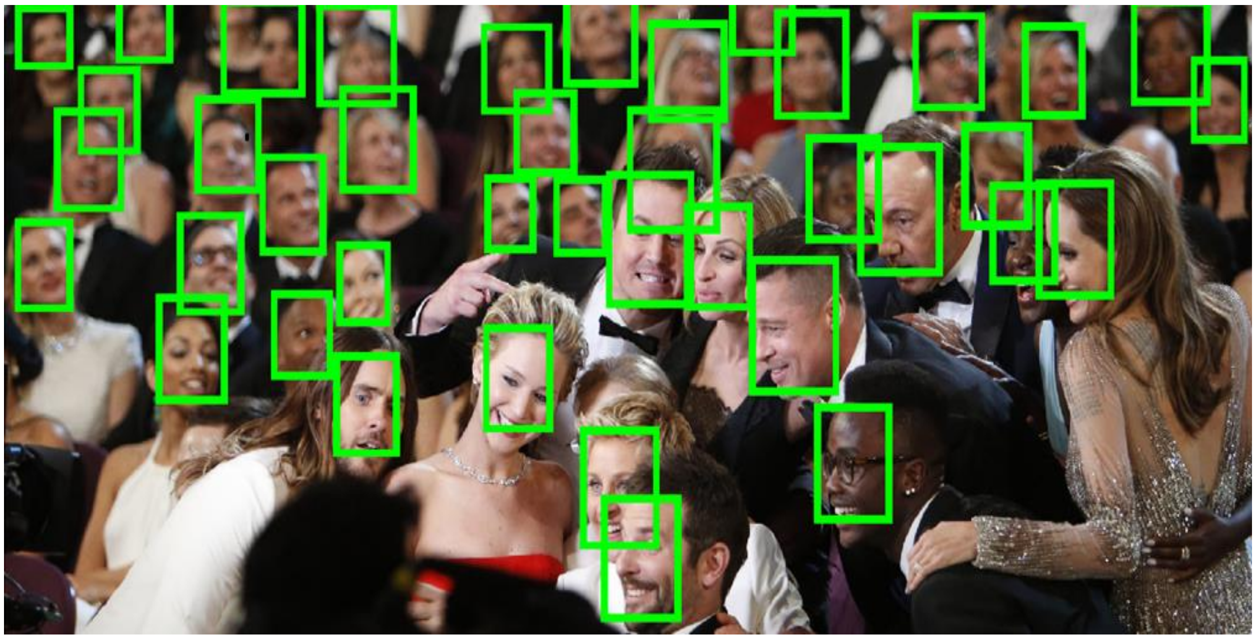


Figure 3: Face Detection

Corner Detection technique:

Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. Corner detection is frequently used in motion detection, image registration, video tracking, image mosaicking, panorama stitching, 3D modelling and object recognition. Corner detection overlaps with the topic of interest point detection.

A corner can be defined as the intersection of two edges. A corner can also be defined as a point for which there are two dominant and different edge directions in a local neighborhood of the point. An interest point is a point in an image which has a well-defined position and can be robustly detected. This means that an interest point can be a corner, but it can also be, for example, an isolated point of local intensity maximum or minimum, line endings, or a point on a curve where the curvature is locally maximal. In practice, most so-called corner detection methods detect interest points in general, and in fact, the term "corner" and "interest point" are used interchangeably through the literature. As a consequence, if only corners are to be detected it is necessary to do a local analysis of detected interest points to determine which of these are real corners. Examples of edge detection that can be used with post-processing to detect corners are the Kirsch operator and the Frei-Chen masking set.

"Corner", "interest point" and "feature" are used interchangeably in literature, confusing the issue. Specifically, there are several blob detectors that can be referred to as "interest point operators", but which are sometimes erroneously referred to as "corner detectors". Moreover, there exists a notion of ridge detection to capture the presence of elongated objects.

Corner detectors are not usually very robust and often require large redundancies introduced to prevent the effect of individual errors from dominating the recognition task. One determination of the quality of a corner detector is its ability to detect the same corner in multiple similar images, under conditions of different lighting, translation, rotation, and other transforms.

There are several algorithms for corner detection where we used Fast9 for corner detection.

1. Harris Algorithm
2. Canny Algorithm
3. Fast9 Algorithm

FAST9 is a key point detection algorithm proposed by Edward Rosten and Tom Drummond (Dept of Engineering, Cambridge University, UK). The primary purpose behind proposing this scheme was to make it computationally feasible to run on embedded processor.
FAST9 involves checking if a sequence of neighboring points (in this case 9) are bigger or smaller than the central pixel by a threshold. This involves a lot of if else conditions which are computationally very expensive on an embedded processor. Instead Rosten and Drummond proposed that one doesn’t have to really check for all the conditions to verify if a point is corner point or not. Instead you could make a few quick checks and then do an early exit, if it appears that the point under consideration is not a corner point. For arriving at the most optimal sequence of the quick checks, Rosten and Drummond proposed using machine learning.

Uncanny Vision was determined to make this key point detection scheme even faster, by utilizing the ARM Neon instruction set. We used a different technique to optimize FAST9. The performance of our implementation is not image or threshold dependent, whereas this is not the case with the original implementation proposed by the authors.
The original implementation had 2 steps to it. First was to detect potential corner points, based on a threshold. Step 2 was to rate these detected points using a score. This scoring can be used to select the best points amongst the corner points detected. The original paper proposes a few possible scoring functions, while OpenCV implements one particular option from amongst these. Uncanny Vision chose a different scoring function (one amongst the options suggested in the paper), which suits our implementation.

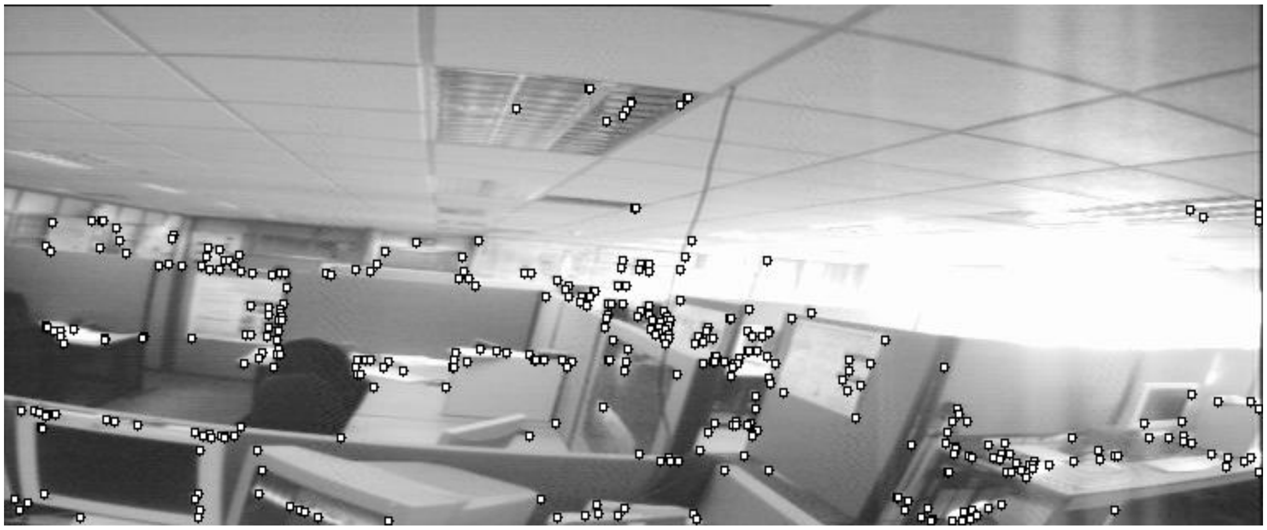


Figure 4: Corner Detection

System Implementation procedure:

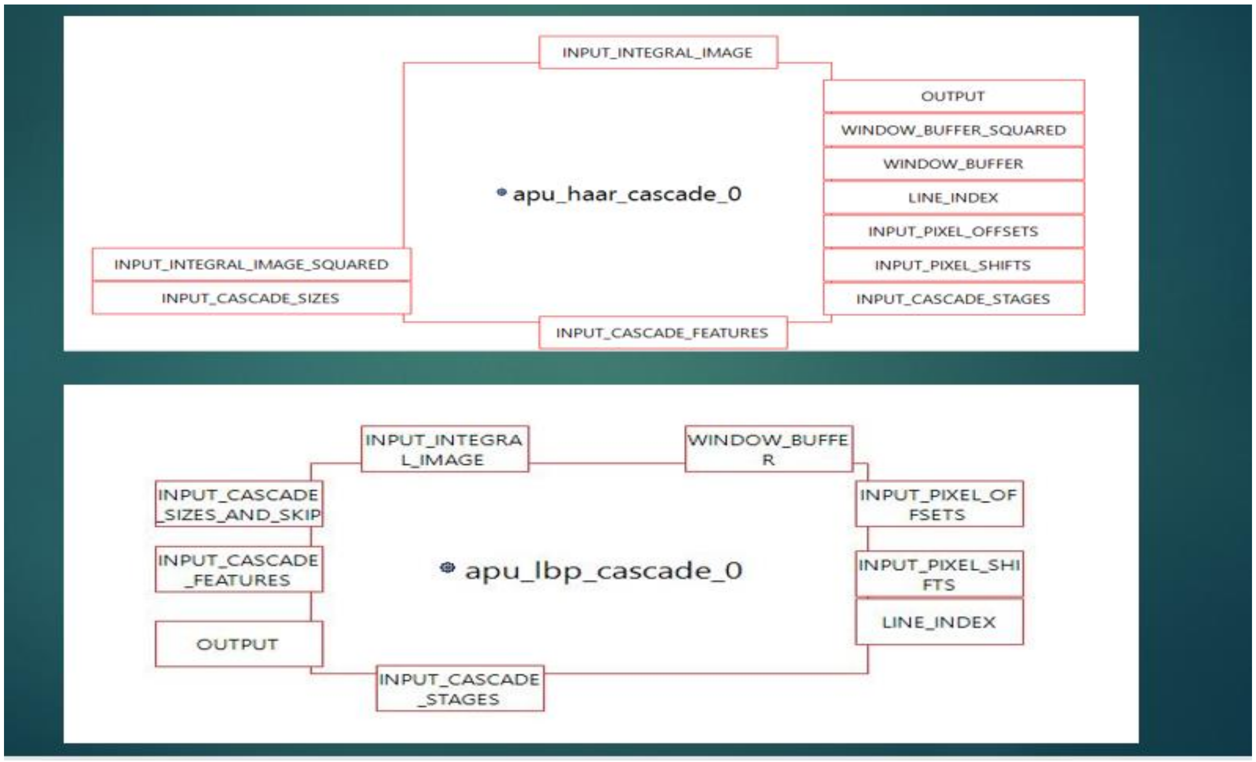
- ❖ An APEX graph is made using APEX2 Graph Project option using the kernels available in Vision SDK.
  1. Make a new APEX2 graph project named: “edge_pedestrian_detect_graph”. Select “File”, “New” and “Other”.
  2. From the Palette window (right side of the S32DS window) select “Kernel from Registry” to drop the block and select blocks according to your requirement of graph.
  3. You do not need to configure any properties for the graph. APEX Core Framework (ACF) will take care of this.
  4. Select a Block from the graph and look at the Properties window (on your left) then change the Name.
  5. After finishing graph, save and validate it.

- ❖ The graph built above is used to make an APEX2 program project.
  1. Make a new APEX2 Program project named: “edge_pedestrian_detect_program”. Select “File”, “New” and “Other”.
  2. From the Palette window (right side of the S32DS window) select Process from Graph block and use necessary graphs needed.

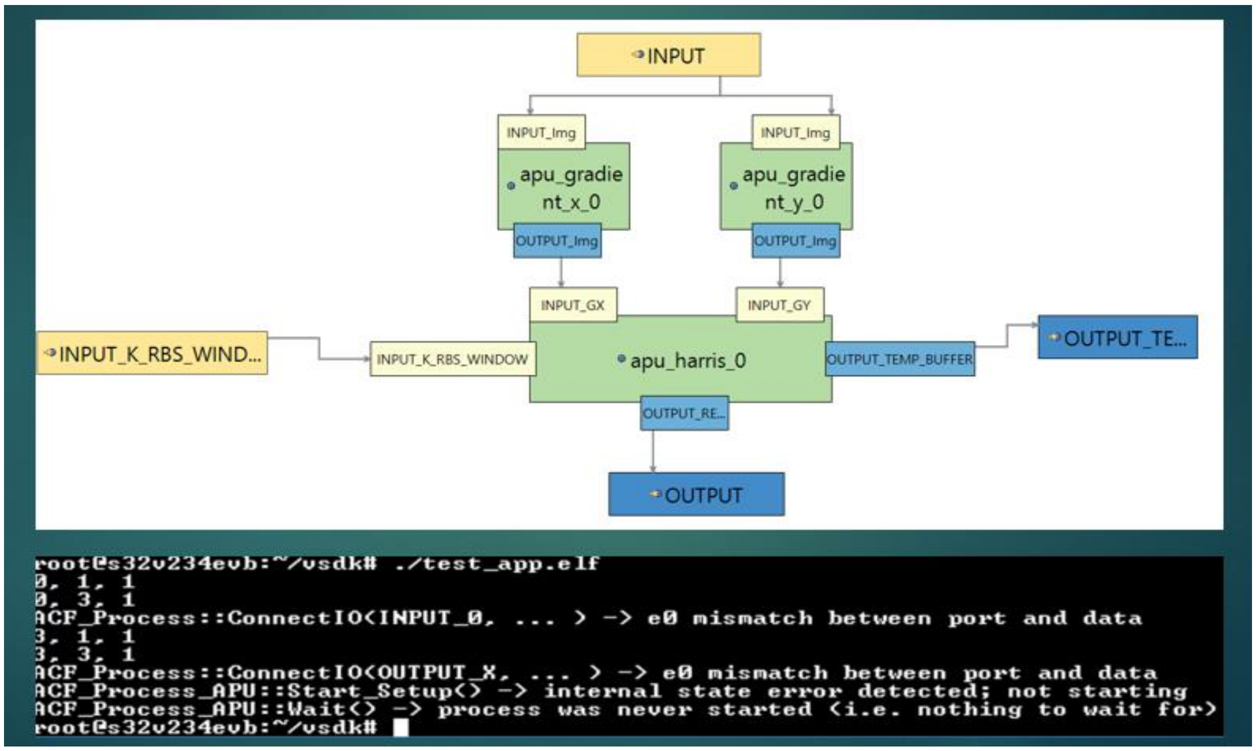
- ❖ Lastly, this source code is used into the Linux application program to accelerate the performance using APEX engines.
  1. Go to File – New – S32DS Application Project. Type the project name: “edge_pedestrian_detect_project”.
  2. Copy the image to the project folder.
  3. Now we have Linux application project ready. We will get back to configuring block properties of program project.
  4. By default, all source code will be generated inside the APEX program project itself. We can reconfigure the destination of source code to any other open projects. We will use this feature and generate the source code in Linux application project.
  5. Select the “Emit Configuration.” option.
  6. Define a new configuration and specify where you want to generate the source code.
  7. Hit “Emit” button to generate a source code at the designated location.
  8. Select “Window”, “Show View”, “Other”, “Remote Systems” then “Remote Systems”.
  9. Add new connection to remote system.
  10. Build the program and Copy application file:“.elf” and image file:“.png” to the “vsdk” folder.
  11. Execute your “edge_pedestrian_detect_program.elf” binary on the target!
  12. Use tera term to locate the image and you can find new image files generated in the same directory.

Attempts and Challenges Faced:

- ❖ Haar-cascade and LBP algorithm for face detection has too many I/O for kernel



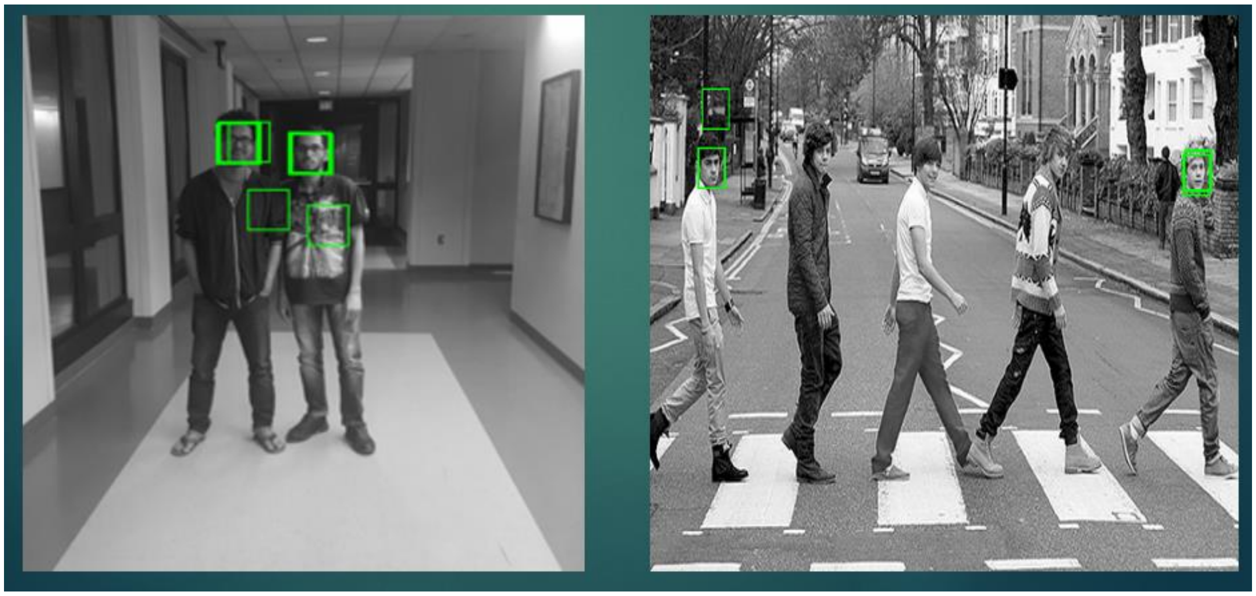
- ❖ Harris algorithm for edge detection got mismatching interfacing with different kernels



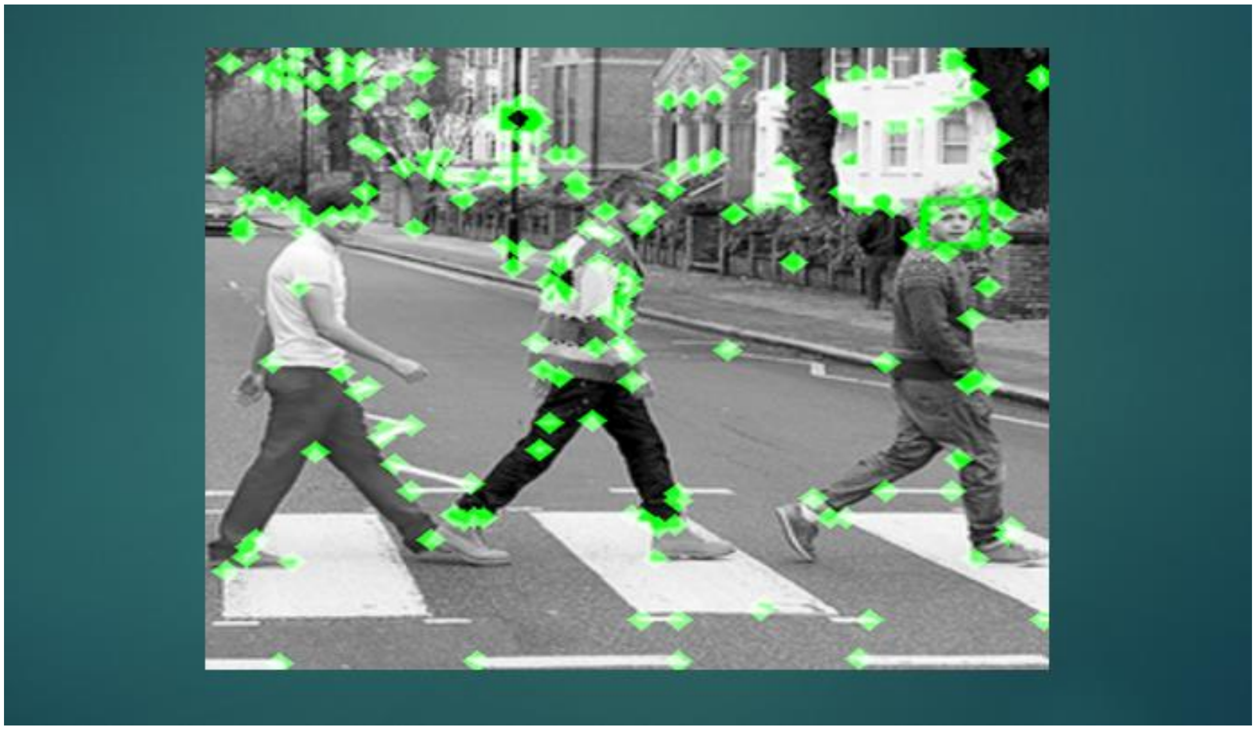
```
root@s32v234evb:~/vsdk# ./test_app.elf
0, 1, 1
0, 3, 1
ACF_Process::ConnectIO(INPUT_0, ... ) -> e0 mismatch between port and data
3, 1, 1
3, 3, 1
ACF_Process::ConnectIO(OUTPUT_X, ... ) -> e0 mismatch between port and data
ACF_Process_APU::Start_Setup() -> internal state error detected; not starting
ACF_Process_APU::Wait() -> process was never started (i.e. nothing to wait for)
root@s32v234evb:~/vsdk# █
```

- ❖ APEX and ISP Lane detection application using Hough Line Detector has Produced completely green output, no actual image visible

- ❖ Face detection project based on Local Binary Patterns (LBP) algorithm for detecting faces for different images



- Fast9 project based on fast9 corner detection algorithm for detecting corners for image



Result Evaluation:

Our target was to detect face and corner of an image, for that we used two images and track pedestrian and corner of that image.

Below picture shows the pedestrian tracking and corner detection.

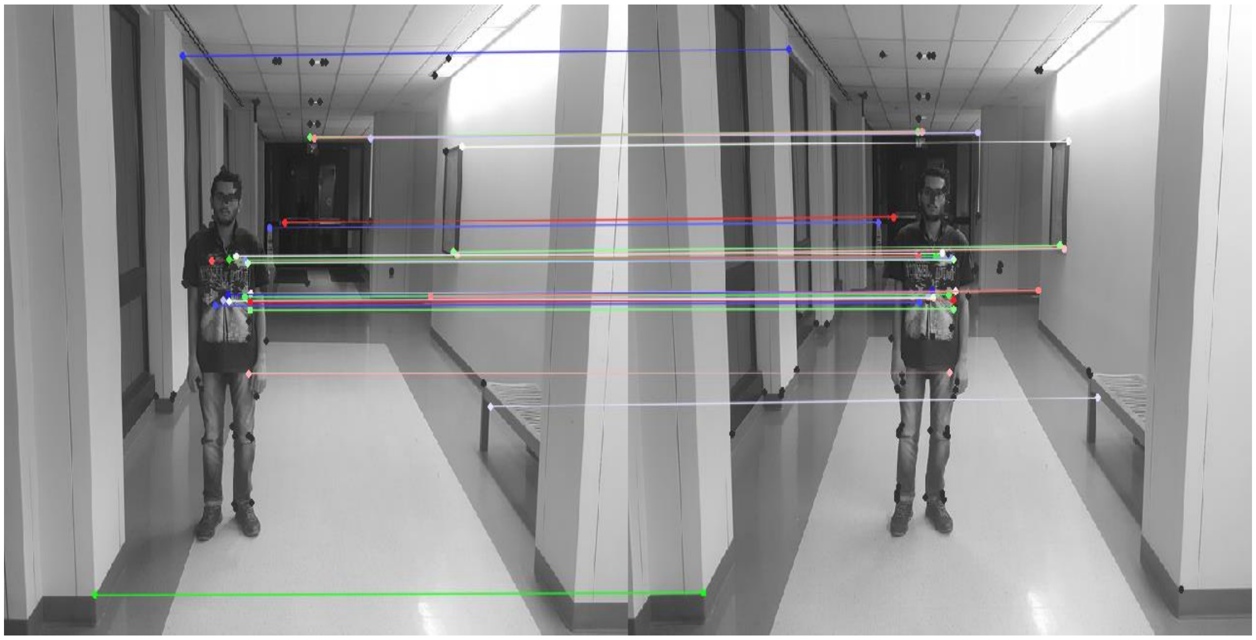


Figure 5: Corner and Face detection

Acknowledgement:

The authors would like to show their gratitude to **Dr. Mohamed El-Sharkawy**, Professor of Electrical and Computer Engineering for his enormous support for the project. Also, they are indebted to Department of Electrical and Computer Engineering, IUPUI and **Mr. Jeffrey Sears,** Laboratory Technician of Electrical and Computer Engineering. Finally, heartiest thanks and best wishes goes to **TA's** of the course, **Dewant Katare** and **Duvesh Pathak**.

References:

[1] https://community.nxp.com/

[2] Kernels Reference Manual

[3] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html

[4] https://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html

[5] https://www.nxp.com/video/:S32V234-Vision-Processor-4

[6] https://www.nxp.com/video/:S32V234-Vision-Processor

[7] https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

[8] 59500 Autonomous Embedded Systems Course Laboratory Manual