CAPTIVATING CONNECTED EXPERIENCE

Camera Architecture Options for Success

Sujal Shah, Director, Automotive Solutions

July 2nd, 2014

Symphony Teleca at a Glance

40 Offices Worldwide

7000+ Innovation Experts

400+ Customers

Delivery Centers in the
Americas, Europe and Asia

Introduction

- Tizen-IVI offers the whole world of HTML5/CSS/JS solutions from a UI point of view, however many of the features require developers to step out of the WRT for native support.

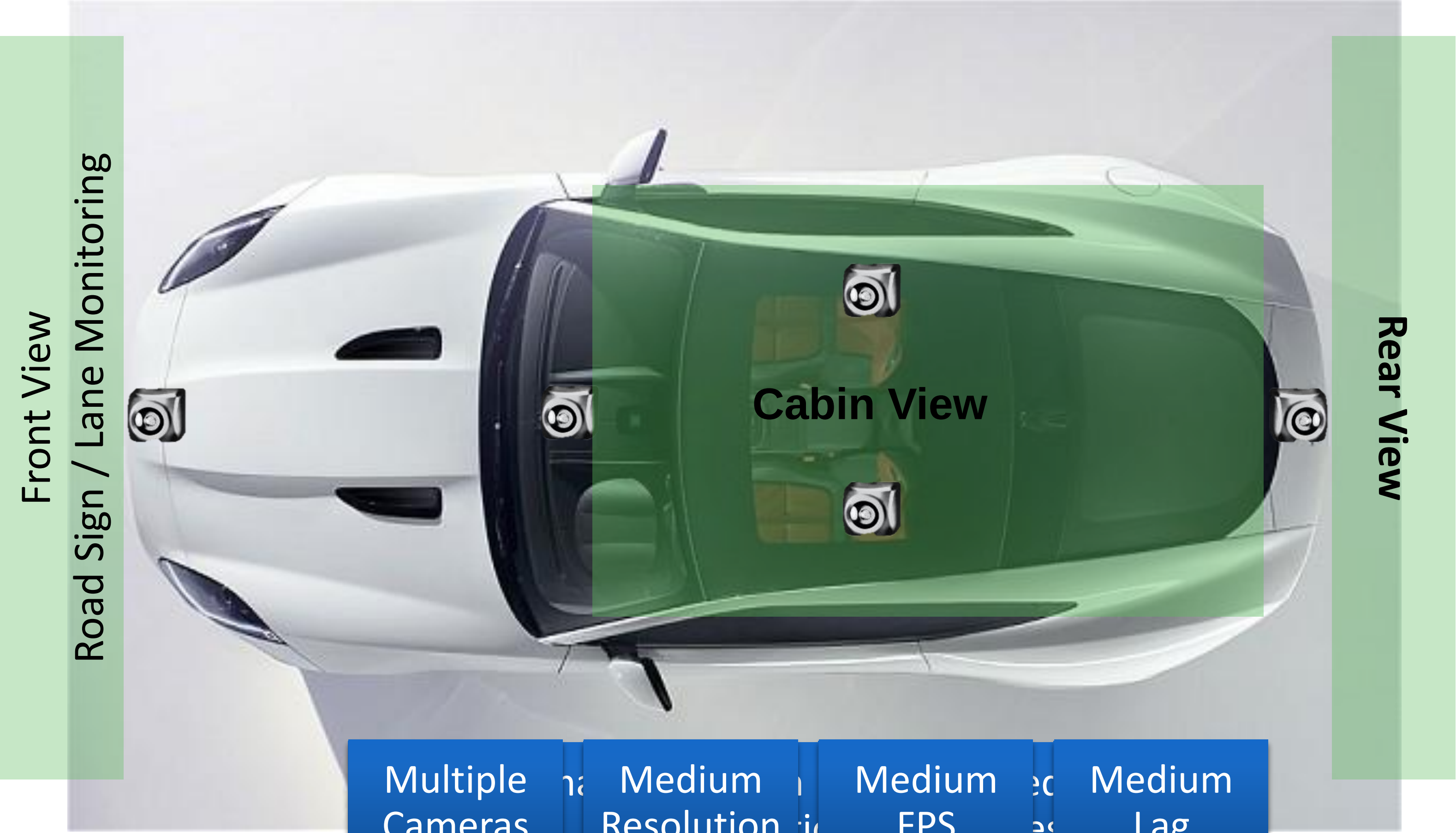
- Symphony Teleca has explored multiple options to address this issue

- We would like to present details of this investigation and hope this knowledge sharing will be helpful to other developers

Agenda

❑ Use Cases
❑ Hardware configurations
❑ General Software Architecture
❑ Solution Options

Camera Use Cases



Multiple Cameras

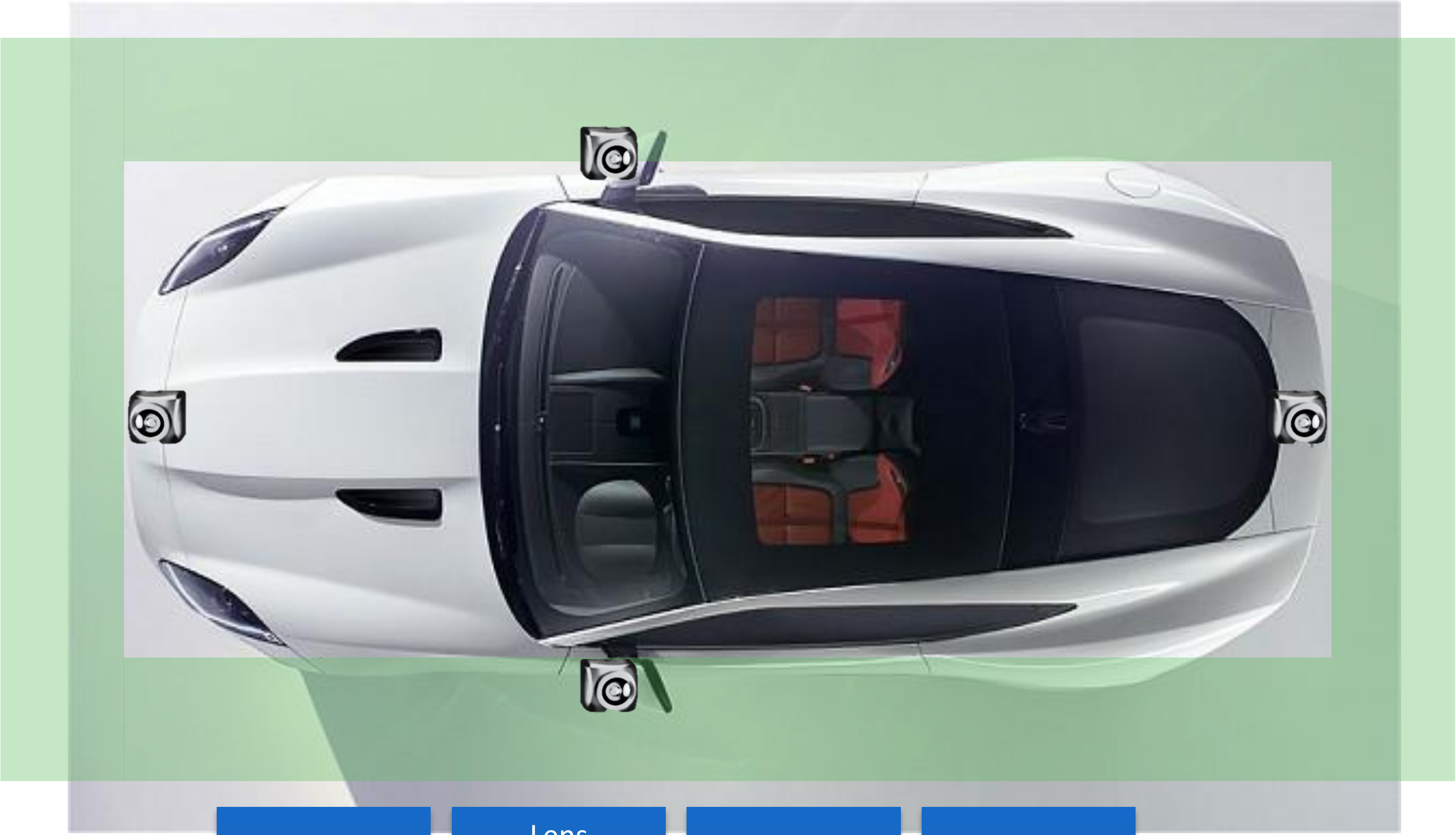
Medium Resolution

Medium FPS

Medium Lag

Camera Use Cases

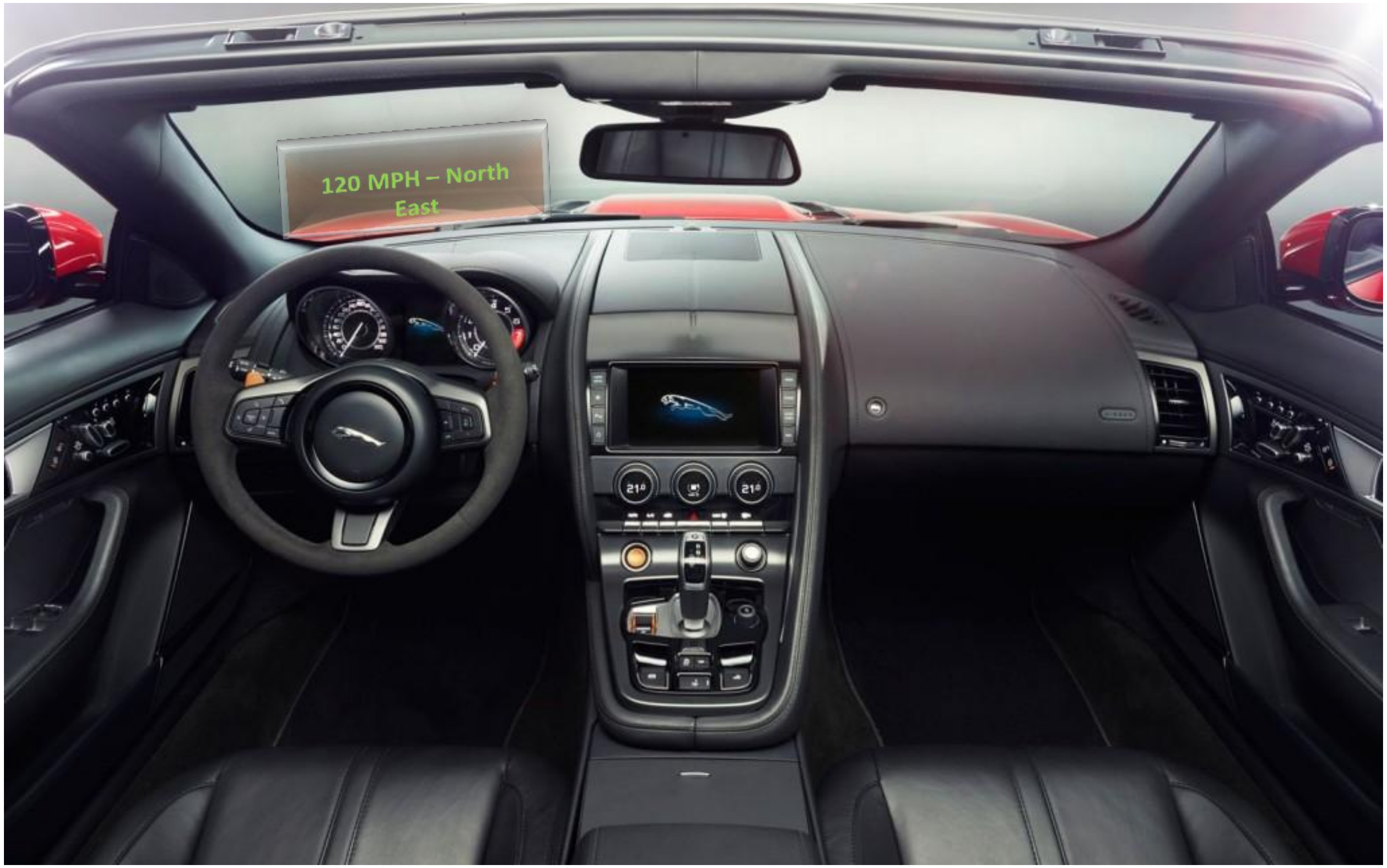
360° Top View



- Multiple Cameras
- Lens Distortion Correction
- Image Stitching
- Minimal Lag

Camera Use Cases

Camera View + Driving Data on a Heads-Up Display



Performance Requirements Considerations

Considering the level of abstraction of Tizen-IVI applications with respect to Native solutions, it is extremely important to fix these requirements before the HW selection step

- Desired FPS
- Video Resolution
- Video Lag
- Distance of Camera(s) from the Processing Units
- Number of Cameras
- Camera Display Concurrency
- Time to Switch Cameras
- Performance of CPU / Encoding Modules

Hardware Configuration Options

As with any embedded system development, it is important to consider hardware options first, as use cases may drive a particular HW combinations

To streamline the hardware selection process, we recommend focusing on the below three groups of hardware:

- ❑ USB, Analog and IP cameras
- ❑ Host system main board offering USB hub (one or many, USB2.0 or USB3.0), a means of connecting an Analog video capture board, and one or more network interfaces
- ❑ Analog video capture board offering single or multiple channel support, together with a video driver which may or may not perform channel multiplexing

Considerations for Hardware Setup

❑ IP Cameras

- IP cameras to support required video quality (frame size/fps) with MJPEG over TCP/IP and/or H.264 over RTSP/RTP depending on particular needs.

❑ Analog Video Cameras

- Non-Multiplexed MiniPCI Express analog video camera capture board
  - Multi channel MiniPCI Express cards expose only one Linux video device, and only one channel sends frames to the interface at one time
  - Such cards require rapid switching between channels.
  - Recommend using a capture board with driver that can support many Linux video devices
- Another option would be several video capture boards, if main board can support
- Cards with higher total FPS should be preferred for multi-camera support to minimize time lost during tuning phase after each channel switch
- It’s important to distinguish between per-channel FPS, and the total FPS supported by the video chip/card
- Card manufacturers are notorious for not keeping their drivers up to date. Check that you can build your card’s driver using the particular kernel used on the target system.

Considerations for Hardware Setup

❑ USB cameras to support required video quality with acceptable USB bandwidth while considering other devices connected to same USB Root Hub:

- Feasibility of multiple USB camera solutions is determined by single camera USB bandwidth allocation for acceptable video quality, the number of cameras needed, and the number of USB Root Hubs.
- The main issue here is the difficulty in knowing how much USB bandwidth will be allocated by a Hub for a given camera – it is not a simple function of video bandwidth, but depends on options presented to the USB driver by the device
- Typical USB Bandwidth allocations to expect for different camera profiles are from 20% to more than 60%.
  - 20% of bus bandwidth is reserved for non-isochronous devices. Hence this might be a limiting factor in multiple camera projects
- Another complication is that you would never find this USB bandwidth usage written on the box. Even more, it might depend to particular Firmware version. So, you would need to perform hands-on experiments with sample cameras to identity this parameter

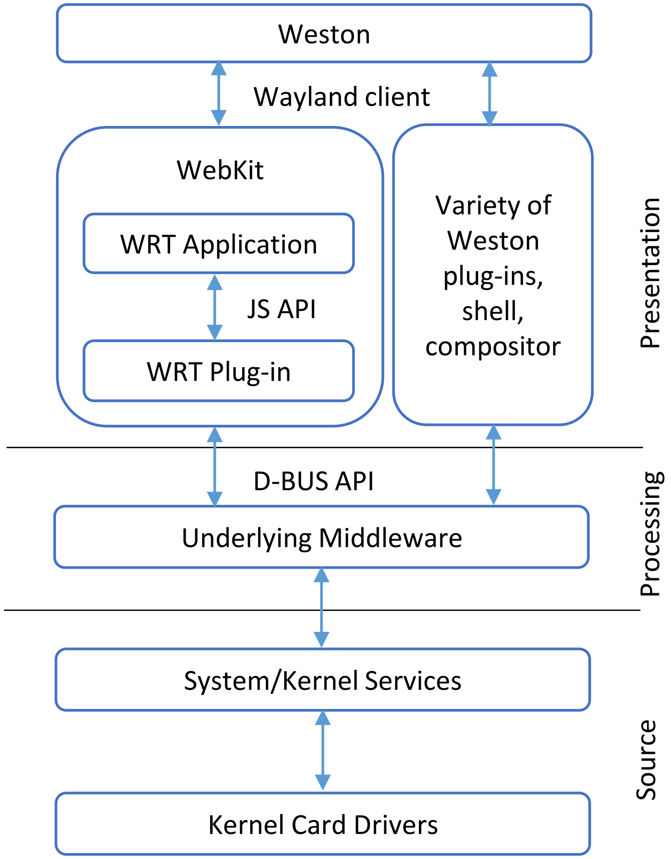
Considerations for Hardware Setup

❑ USB bandwidth issue mitigation:

- Consider requiring USB 3.0 support in root hub and camera as this might relax some of the limits around USB bandwidth
- Consider H.264 support in the camera and availability of the software options to deal with such interfaces

General Architecture

- ❑ WebKit is a sandbox where the app is running
  - WRT Application is a combination of HTML/CSS/JS technologies, hence it can deal with the video frames with help of JS or leave it to the functions available in WebKit core.
  - Video tag, Pure Canvas, WebGL or JS MPEG-4 decoder.
  - WRT Plug-in's primary usage is control (over D-BUS or similar interface) and secondary is bypass interface to the source of the video data
- ❑ With respect to webkit the options are:
  - Combination of webkitGetUserMedia and createObjectURL to get blob object referencing the stream taken from V4l2, hence applicable to USB and Video Capture Board options
  - Make your own Blob object with help of JS and WRT Plug-in and feed it to video tag like the one from the above option
- ❑ Weston plug-ins might be one of the options if you want to make overlay client where the WRT Application only controls the way and time video is presented. Video frame processing is then done outside WRT sandbox.



Processing Layer

- ❑ Underlying Middleware (Processing layer) is a combination of several components:
  - Tizen components such as GStreamer (or FFMPEG) to unify interface to the various video streaming devices.
    - This is distinct from GStreamer as used directly inside Webkit.
  - Custom SW components to control parameters of system components
  - Or Custom SW components to deal with the device interfaces directly, considering specific details of each video streaming device

- ❑ The processing layer group is essentially an abstraction layer that hides details of the various types of processing required by different video sources before that source is exposed to the WRT
  - GStreamer can perform most such processing tasks, even if some of the options required are really complicated.
  - Gstreamer processing overhead could be avoided where little or no processing is required by creating custom SW to deal with V4l2 or IP camera interface directly.
  - If GStreamer does not provide the processing you need, you should probably consider creating a GStreamer plug-in, which would then make all the GStreamer facilities available too

Source Layer

- Could be a Linux Video device in combination with V4l2 interfaces. Most applicable to USB and Video Capture board types of HW. Functionality and parameters available thru ioctl interface should be also considered as part of this layer group

- Could be custom interfaces made to support specific features such as video in h.264

- Network interface to IP cameras or non-standard interfaces to the devices where you would need to use some intermediate layers to get access to the camera even thru IP or other type of connection.

Software Specifics

❑ Webkit: Video Tag: Does video buffering if used with network stream set directly to SRC. Lag depends on video type and properties, but typical values to expect are *from 3 to 5 seconds.* Another complication is list of supported video formats. Webkit used in Tizen-IVI supports OGG and H.264/MPEG-4 Part 10. On the plus side, it's possible to run up to 6-8 streams at the same time with good FPS and quality of decoding.

❑ Webkit: Pure Canvas: As fast as can be expected for single camera set up and can offer single camera view with FPS close to 25-30, but strictly depends on implementation in JS and below, which provides byte array with frame. Typical lag is about 1 second or less.

❑ Webkit: WebGL could provide more features with additional video frame processing, but still might be comparable to the pure canvas solution

❑ Webkit: JS MPEG-4 decoder is fast enough to get smooth video play with minimal lag, but might not work out of the box on Tizen IVI.

❑ WRT plug-in: D-BUS control interface: Can be Corba or any other interface to the system or custom service running to serve the webkit with the video streams and control the cameras over ioctl and V4L2 interfaces

Software Specifics

❑ WRT plug-in: Combination with underlying MW to control and run Gstreamer, or to read the frames directly from the Linux video interface such as /dev/videoX. Good for intermediate processing of the frames, but relatively slow because can work only with Canvas or other direct draw method.

❑ Webkit: webkitGetUserMedia and createObjectURL. Doesn’t work out of the box with Tizen-IVI as Webkit is using GStreamer 1.0 library, but still sends the parameters in the GStreamer 0.10 format. Hence some modifications would be required to bypass this issue. Meanwhile, this way does work with Crosswalk and result is very smooth.

❑ Webkit: Blob object in combination with WRT plug-in to serve video tag source. Could be comparable to webkitGetUserMedia and createObjectURL method, but performance would depend entirely on the WRT Plug-in implementation.

❑ Weston: Shell or Compositor. Fast and flexible as you manage layer merging and surfaces directly. Can be helpful in complex layer merge scenarios such as Media Player with data shown on top of the video, but constraints are in the need for expertise and complexity of the system. Additionally it may impact whole system performance and stability, hence shouldn’t be used without proper assessment of the needs and risks.

Software Specifics

❑ GStreamer: Complex, but powerful solution which can read multiple sources (from files to IP cameras) and give multiple types of outputs with different conversions in between input and output. Main points to consider are:

- There will probably be more than one way to build a GStreamer pipe to get functionality you need
- Different versions of Tizen may include different versions of GStreamer and its plug-ins leading to different and sometimes failing behavior in between releases
- Vaapi (HW support) is present, but it's effectiveness depends on the Host system
- SW encoders may fail under heavy system load and produce corrupted streams
- Not everything is obvious. E.g. tcpserversink is good for streaming the data over the network, but you would need to create your own wrapper (proxy) to add HTTP headers if you try to use it with browser Video tag.
- Many useful plug-ins are still in "bad" and "ugly" packages meaning you have very minimal support for most of functionality outside mainstream usage

Software Specifics

❑ FFMPEG: Simpler than GStreamer, but also limited. Upstream versions are not expected to be available in the Tizen IVI repo and please be aware of modules which are not even built for Tizen.

❑ Custom SW: Needed to deal with devices directly, or to control device status while streaming is ongoing thru other components. Highly dependent on circumstances, but might be required for:

- Setting up Video Capture board parameters
- Controlling status and availability of the devices
- Matching devices versus some ID to identify exact cameras in multi camera setup

Solution – Rear View Camera

- Rear view Camera- Minimal lag, high resolution, flipped frames
- For the USB and Analog cameras the simplest way would be a combination of webkitGetUserMedia and createObjectURL methods.
- A custom solution could be implemented, such as own blob object or Weston plug-in for overlay in combination with GStreamer.
- Latter options are helpful especially if there is a need for camera multiplexing or IP camera usage.
- Flipping could be done by browser as layer property, hence no additional processing required.

Solution - 360° Top View

- ❑ Multiple cameras setup, correction of lens distortion, stitching of camera outputs into single panoramic video, minimal lag
- ❑ Architecturally correct way would be custom GStreamer plug-in implementation and usage of OpenCV for stitching images from different sources, blending and lens corrections.
- ❑ Standard videomixer plug-in is good example for quick start with Gstreamer
- ❑ Custom SW could implement the same functionality by reading the frames from Linux Video interfaces directly or using GStreamer as a library.
- ❑ For minimal lag Weston overlay or own blob object could be used to pass video

Solutions - Heads-Up Display

- ❑ Camera view with driving details overlaid for a heads-up display:

- ❑ Merge of the dynamic/static data with actual camera view, minimal lag, medium FPS

- ❑ If data is primarily textual information, then browser layer capabilities could be used to merge it with video stream. Otherwise variety of GStreamer plug-ins could do it at the cost of some MW complication.

- ❑ For medium FPS, browser Canvas is good option where the frames are read directly from device (or from GStreamer for IP cameras), processed in custom SW and then passed to Canvas thru WRT plug-in.

- ❑ WebkitGetUserMedia and createObjectURL methods are still good if there is no need to deal with IP cameras.

Solutions – Cabin View & Front View

❑ Cabin View

- Multiple cameras setup, medium or low FPS, medium lag
- In general similar to “360° Top View” case, but lower requirements for presentation part, hence Canvas or own blob object can be implemented to handle display

❑ Front View – Lane and Sign Recognition

- One by one frame processing before display in various recognition cases such as faces, road signs or other objects recognition –
- Minimal lag, medium FPS as the recognition may take longer than single frame release time, no en(de)coding artifacts to achieve best recognition results
- Custom SW to read frames directly from the device or GStreamer library and to process it further can be implemented.
- Depending on the processing required, it could be a Gstreamer plug-in doing analysis of the frames and sending the signals to upper layers.
- Canvas or blob object for medium FPS and minimal lag.
- Weston overlay may over complicate the system, but technically possible to achieve minimal lag and best performance of the display

Need More Details?

Need more details?

Specifics of particular selections of SW and HW, for example regarding, WebKit, WRT-plugin, GStreamer or Weston can be requested at sujal.shah@symphonyteleca.com.

Go beyond ordinary.
Wherever you want to go,
we’ll get you there. Faster.