DRIVEWORKS

August 2016

NVIDIA

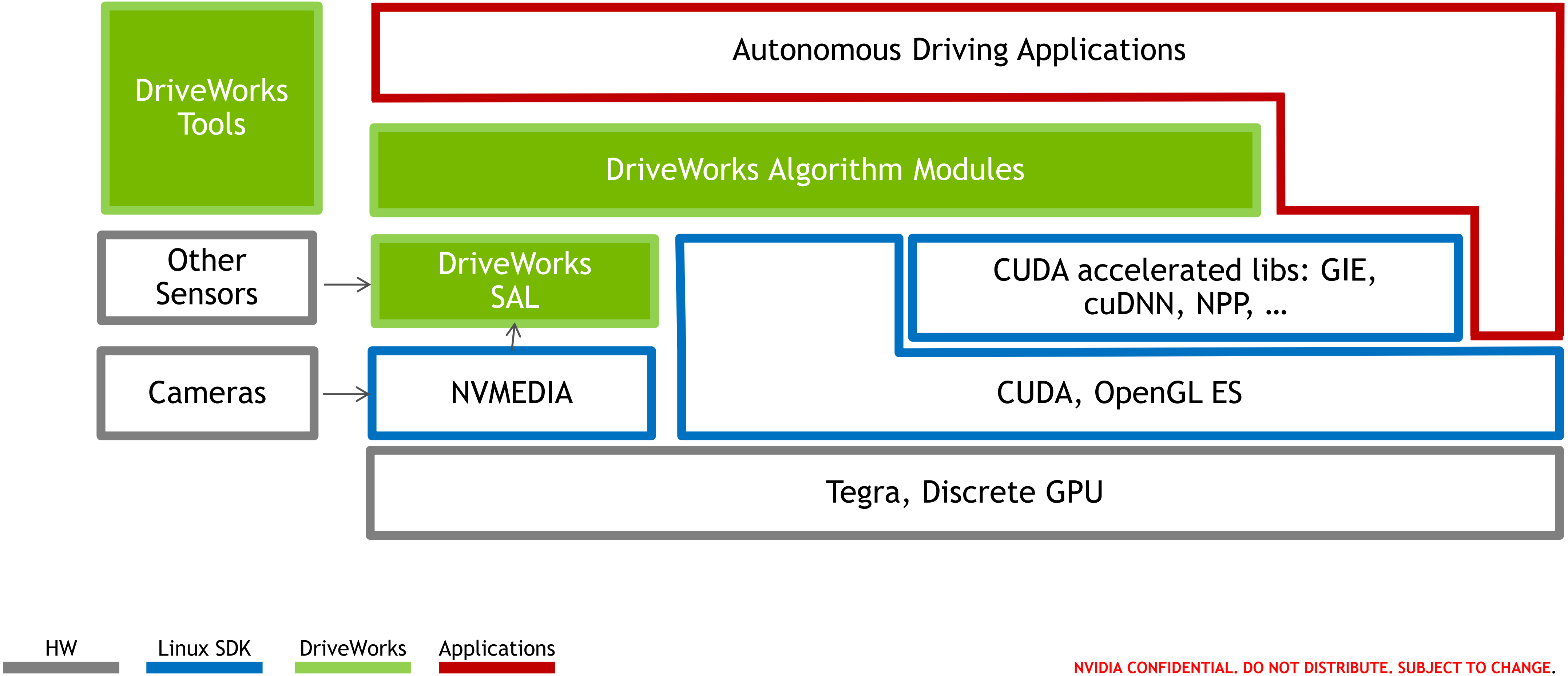
WHY DRIVEWORKS SDK?

- Smart car's complexity is high and will increase
  - More sensors, more functionality, more autonomy
- Distributed ECUs -> Tegra Soc central processor
  - More computation power, whole system awareness
  - Different programming models, parallel computing, scheduling
- DriveWorks as an API for Autonomous Driving
  - SDK, Runtime , Tools, Reference Applications, Library Modules
- Design Philosophy
  - Modular, Educational, Optimized, Open

DRIVE PX: AUTONOMOUS DRIVING PLATFORM

DRIVEWORKS SDK	**DETECTION**	**LOCALIZATION**	**PLANNING**	**VISUALIZATION**
	Detection/Classification	Map Localization	Vehicle Control	Streaming to cluster
	Sensor Fusion	HD-Map Interfacing	Scene understanding	ADAS rendering
	Segmentation	Egomotion (SFM, Visual Odometry)	Path Planning solvers	Debug Rendering
System SW	V4L/V4Q, CUDA , cuDNN, NPP, OpenGL, …			
Hardware	Tegra , dGPU			
Sensors	Camera, LIDAR, Radar, GPS, Ultrasound, Odometry, Maps			

DRIVEWORKS SW STACK



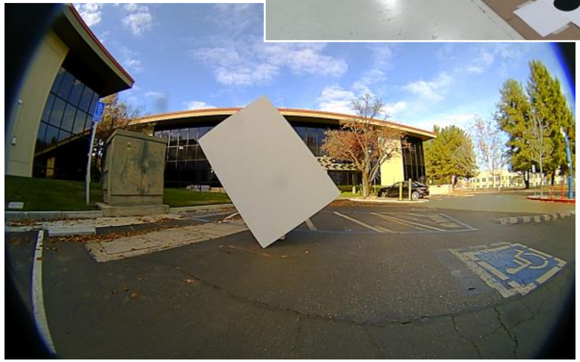
SOFTWARE BEST PRACTICES

- Coding standards
  - MISRA C/C++ - with documented exceptions
  - ISO26262 – (Feb'17)
- SW development process
  - Common CMAKE infrastructure
  - Continuous Integration
  - Mandatory Code reviews
  - QA infrastructure

- Light Agile methodology
  - Bi-Weekly sprint reviews
  - Jira for issue tracking
- Validation of design (**Eat your own dog food!**)
  - Develop -> Test -> productize
- 3rd Party libs/dependencies
  - To a minimum
  - Binary static linkage

DRIVEWORKS TOOLS

CALIBRATION AND SENSOR REGISTRATION



Set of tools to calibrate sensors, and runtime module to perform online calibration

Features

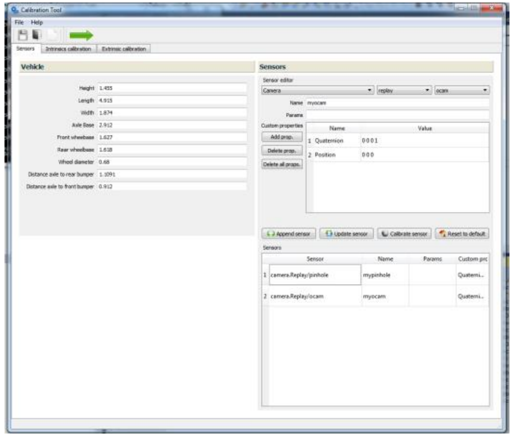
- Factory calibration tool (goal: zero stop calibration)
- Camera Intrinsic calibration – OCAM/Pinhole model. Pattern
- Camera Extrinsic calibration
  - Two cameras
  - 4-camera setup (surroundview config)
- Lidar to camera extrinsic calibration

- Online calibration (post V0.1)
  - Recalibration of extrinsics only, with possible extension recalibrate intrinsics as well
  - Optimized bundle adjustment for automotive configurations

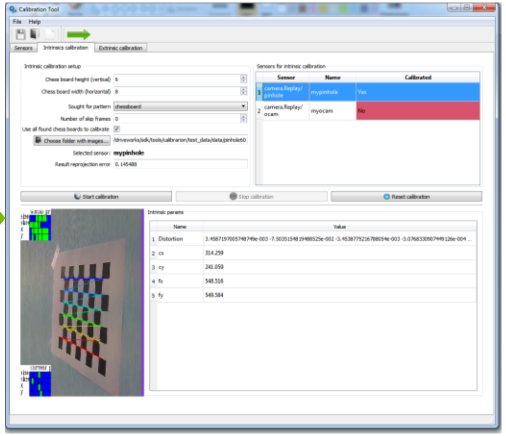
Modules
- Productized tools
- Patterns and tool for intrinsic calibration
- Patterns and library for extrinsic calibration
- Libraries for on rig calibration (post V0.1)

CALIBRATION AND SENSOR REGISTRATION

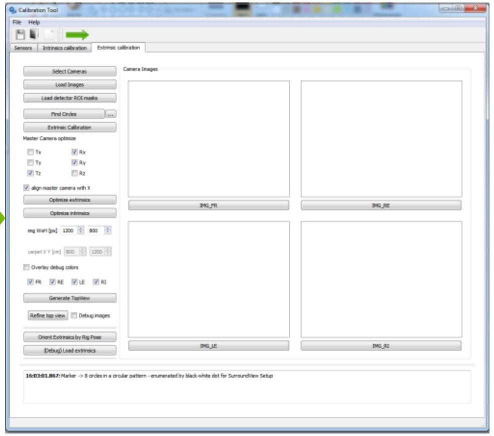
- Rig defines sensors and also rough location estimates
- Camera Intrinsics: OCAM and OpenCV Pinhole parameters
- Camera Extrinsics: 4 SurroundView or relative between 2 cameras
- Lidar Extrinsic: relative to a camera that sees the pattern



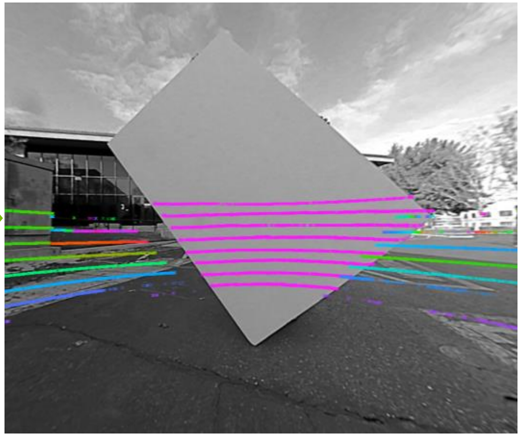
Rig Configuration



Camera Intrinsics



Camera Extrinsics



Lidar Extrinsics

TRACE CAPTURING AND REPLAY

Same platform and SW as both development and deployment

- Tuned performance to avoid glitches during capturing and recording
- Optimized for Load balaning threads and cores, memory and IO

Unique time synchronization protocol (PTP Aurix)

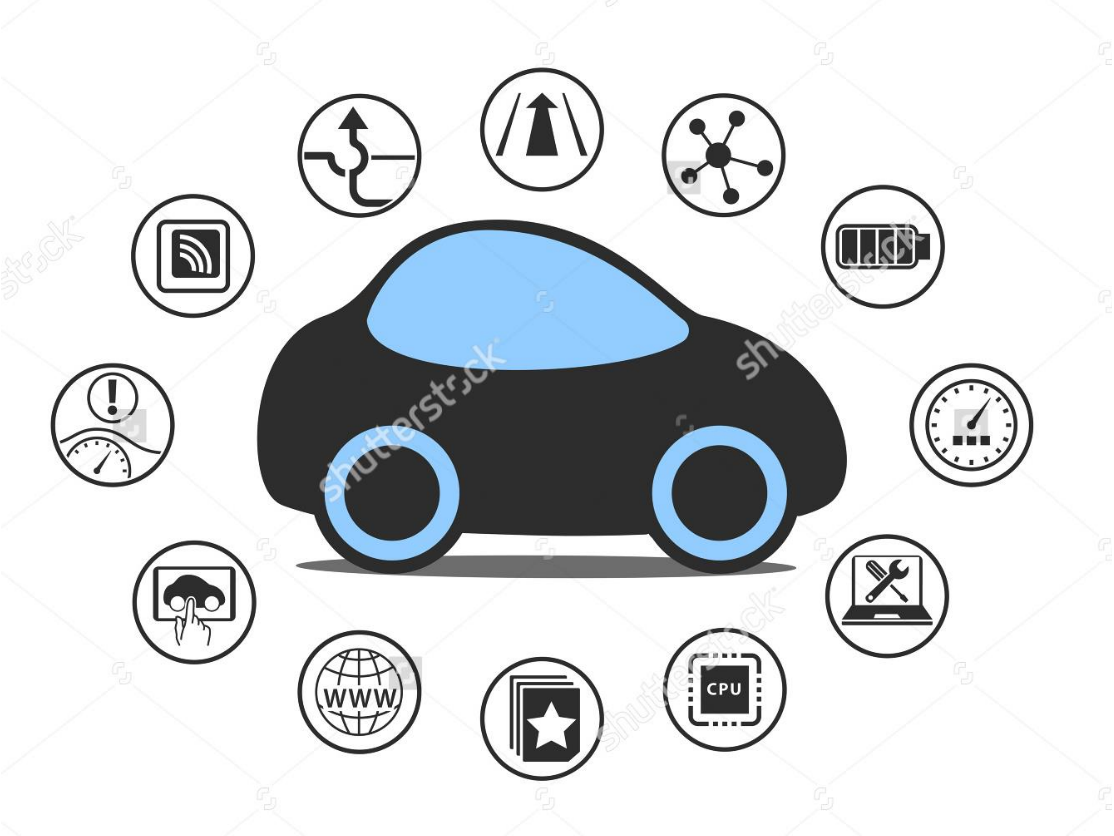
Single man operation:

- Support to launch multi-sensor recording at one key press
- Coordinated play/pause/stop for all sensors

For future versions include (post V0.1)

- Synchronization between multiple processes (different Tegras)
- Built-in calibration capabilities

DRIVEWORKS SAL



SENSOR ABSTRACTION LAYER (SAL)

Goals

- Provide a common and simple unified interface to the sensors
- Provide both HW sensor abstraction as well as virtual sensors (for replay)
- Provide raw sensor serialization (for recording)
- Deal with platform and SW particularities
  - API/Processor Conversion/transfer: CUDA, GL, NvMedia, CPU
  - Exploit additional SoC engines: H264/H265 codec, VIC

COMMON SENSOR API

dwSAL_createSensor

Prepare sensor for data delivery: power up, establish connection, open socket, allocation FIFOs, etc...

dwSAL_startSensor

Start recording into sensor FIFO

Minimum Latency

Data consumption

dwSAL_stopSensor

Stop recording into sensor FIFO, drain FIFO.

dwSAL_releaseSensor

Shutdown sensor, release resources

SENSOR DATA CONSUMPTION

Avoid _unnecesary_ memory copies

- Pre-allocated pools. no-copy, no runtime alloc policy
- Direct access to sensor data pointer via transfer ownership paradigm
- Non-blocking accessors and blocking with timeout

Generic accessors

- dwSAL_readRawData()
- dwSAL_returnRawData()

Mainly used for serialization

Custom (per sensor) accessor

- dwSAL_read[Camera,Lidar,…]Data()
- dwSAL_return[Camera,Lidar,…]Data()

Provide per sensor specific data

SCHEDULING

- Current paradigm is non-blocking functions and blocking with timeout
- Defined by EGL, CUDA and NvMedia paradigms and capabilities
- Goal is event-driven and non-blocking data-flow model to be light-weight and efficient
  - Be able to schedule work ahead to hide latencies on triggering work for all our HW engines
  - Use as little threads as necessary to increase runtime determinism of the system

QUERY SENSORS

- Query sensors supported by HW platform
- DW SAL built using factory pattern

$\underbrace{\text{sensor\_type.mode}}_{\text{Protocol}}?\underbrace{\text{param1=value1,param2=value2,}}_{\text{Parameter String}}\ldots$

```
// get information about available sensors
    uint32_t numSensors = 0;
    dwSAL_getNumSensors(&numSensors, hal);
    for (uint32_t i=0; i < numSensors; i++)
    {
        const char* protocol  = "";
        const char* params = "";
        dwSAL_getSensorProtocol(&protocol, i, hal);
        dwSAL_getSensorParameterString(&params, i, hal);

        std::cout << "Sensor [" << i << "] : "
                  << protocol << " ? " << params << std::endl;
    }
```

```
Initialize Driveworks SDK v0.1.0 (541a3e72cf25fb4f1ab4919639cd42785f3469ac)
Platform: OS_VIBRANTE_V4L:
  Sensor [0] : can.socket ? device=can0[,baud=500000]
  Sensor [1] : can.aurix ? aurix=10.0.0.1,bus={a,b,c,d,e,f}[,aport=50000,bport=60395]
  Sensor [2] : can.virtual ? file=/path/to/file.can
  Sensor [3] : camera.gmsl ? csi_port={ab,cd,ef},camera-count={1,2,3,4},camera-type={ov10635,c-ov10640-b1}
  Sensor [4] : camera.virtual ? video=filepath.h264[,timestamp=file.txt]
  Sensor [5] : gps.uart ? device=/dev/ttyXXX[,baud={1200,2400,4800,9600,19200,38400,57600,115200}[,format=nmea0183]]
  Sensor [6] : gps.virtual ? file=filepath.gps
  Sensor [7] : lidar.virtual ? video=filepath.bin
  Sensor [8] : lidar.socket ? ip=X.X.X.X,port=XXXX,device={QUAN_M81A,IBEO_LUX}
Driveworks SDK released
```

SENSOR CREATION

- Sensor created via protocol and parameter string

```
dwInitialize(&sdk, DW_VERSION, &sdkParams);

// create HAL module of the SDK
dwSAL_initialize(&hal, sdk);

// create GMSL Camera interface
dwSensor cameraSensor = DW_NULL_HANDLE;
{
    dwSensorParams params;
    std::string parameterString = arguments.parameterString();
    params.parameters           = parameterString.c_str();
    params.protocol             = "camera.replay";
    dwStatus result = dwSAL_createSensor(&cameraSensor, params, hal);
    if (result != DW_SUCCESS) {
        std::cerr << "Cannot create driver: camera.replay with params: " << params.parameters << std::endl;
        exit(1);
    }
}
```

$\underbrace{\text{sensor\_type.mode}}_{\text{Protocol}}?\underbrace{\text{param1=value1,param2=value2,}}_{\text{Parameter String}}\ldots$

SAL SUPPORTED SENSORS

- Currently Supported sensors
  - CSI cameras: OV10635, OV10640
  - CAN: SocketCAN, Aurix EasyCAN
  - LIDAR: Quanergy M81A, IBEO Lux
  - GPS: uart

- Sensors in bring-up
  - CSI Cameras: AR0231
  - USB Cameras: PointGrey
  - LIDAR: Velodyne
  - Radar: Delphi ESR2.5, SSR2

DW IMAGE TYPE

- Image handling
  - Formats YUVxxx, RGB, Raw
  - Memory layout: Pitch linar / Block Linear
  - APIs
    - CUDA (pitch-linear & block-linear)
    - NVMEDIA (pitch-linear & block-linear)
    - GL (block-linear)
    - CPU (pitch-linear)

```
typedef enum {
    DW_IMAGE_CPU = 0,
    DW_IMAGE_GL,
    DW_IMAGE_CUDA,
#ifdef VIBRANTE
    DW_IMAGE_NVMEDIA,
#endif
    DW_IMAGE_UNKNOWN
} dwImageType;
```

DW IMAGE TYPE

Shared Properties & Specific Types

```
typedef struct {
    /// The type of image.
    dwImageType type;

    /// The width of the image in pixels.
    uint32_t width;
    /// The height of the image in pixels.
    uint32_t height;
    /// The pixel format of the image.
    dwImagePixelFormat pxlFormat;
    /// The pixel type of the image.
    dwTrivialDataType pxlType;

    /// The time in microseconds when
    /// the image was acquired.
    dwTimestamp_t timestamp_us;
} dwImageProperties;
```

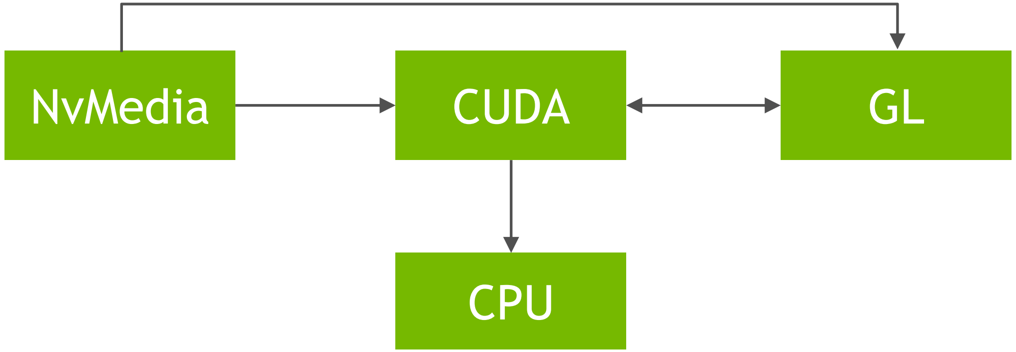
```
/// @brief A CUDA image.
typedef struct {
    /// The properties of the image.
    dwImageProperties prop;
    /// The memory layout of the image.
    dwImageCUDAMemoryType layout;
    /// The plane count of the image.
    uint32_t planeCount;
    /// The pitch of each plane in bytes.
    size_t pitch[DW_MAX_IMAGE_PLANES]; // pitch in bytes
    /// The pointer to the image planes.
    void *dptr[DW_MAX_IMAGE_PLANES];
    /// The CUDA image plane data.
    cudaArray_t array[DW_MAX_IMAGE_PLANES];
} dwImageCUDA;
```

```
#ifdef VIBRANTE
/// @brief An NvMedia image.
typedef struct {
    /// The properties of the image.
    dwImageProperties prop;
    /// The pointer to the NvMedia image.
    NvMediaImage *img;
} dwImageNvMedia;
#endif
```

DW IMAGE STREAMER

Transfers images between APIs (CUDA <-> GL <-> CPU <-> NvMedia)

- _zero_ copies whenever possible
- Copy is involved otherwise
- Currently supported paths:



- Example

```
dwImageStreamer_postCUDA(&frameCUDArgba, cuda2gl);
if (dwImageStreamer_receiveGL(&frameGL, 30000, cuda2gl) == DW_SUCCESS) {
    // Do my thing
    dwImageStreamer_returnReceivedGL(frameGL, cuda2gl);
}
```

DW FORMAT CONVERTER

Handles image format conversion within the same API

YUV 2 RGB

- Using GPU (CUDA API)

```
DW_API_PUBLIC
dwStatus dwImageFormatConverter_copyConvertCUDA(dwImageCUDA *output, const dwImageCUDA *input,
    dwImageFormatConverter formatConverter, cudaStream_t stream);
}
```

- Using VIC engine (NvMedia API)

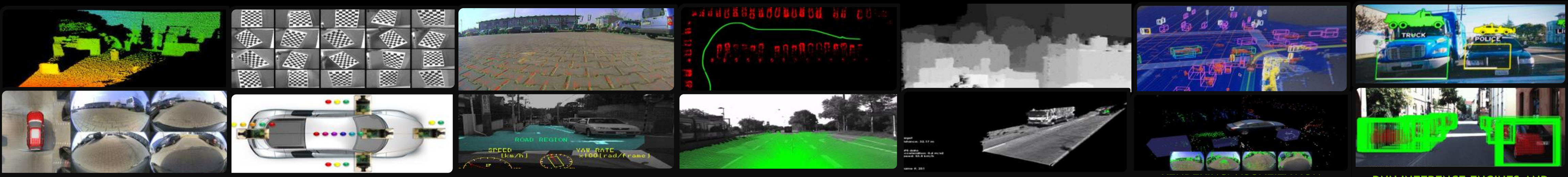
```
DW_API_PUBLIC
dwStatus dwImageFormatConverter_copyConvertNvMedia(dwImageNvMedia *output, const dwImageNvMedia *input,
    dwImageFormatConverter formatConverter);
}
```

DRIVEWORKS SDK MODULES

SDK LIBRARY MODULES

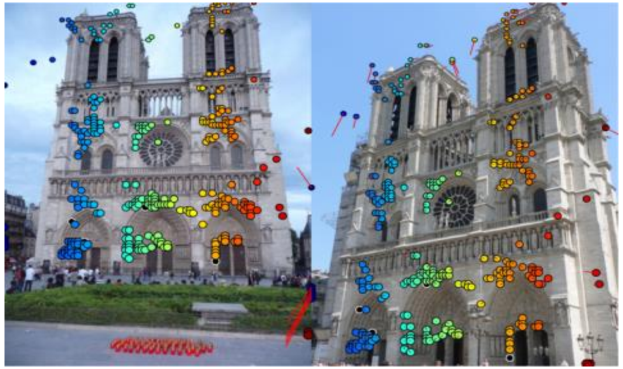
- 2D Feature Tracker
- Grid based fusion / occupancy grid module
- Structure From Motion and Egomotion
- Rendering/Visualization pipeline for debug and display
- 2D Object Tracking
- DNN inference engines

(+ documentation and samples)



MULTISENSOR SYNC'ED RECORDING | CALIBRATION/REGISTRATIONS MULTI-SENSOR TOOLS | FEATURE/OBJECT TRACKING AND EGOMOTION | GRID BASED FUSION ENGINE OCCUPANCY GRID MODULE | STEREO RECONSTRUCTION | PIPELINE FOR DEBUG AND DISPLAY | DNN INFERENCE ENGINES AND DIGITS TEMPLATES

2D FEATURE TRACKER



Highly optimized high quality 2D feature detector and tracker

FEATURES

- Pyramid Sparse Optical Flow
- Harris Corner Detector
- Feature Managment
    - *Multi-Frame Tracking*
    - *Uniform Sampling w/ Minimal Distance*
    - *Max Feature Count*
- *Motion Prediction Models*

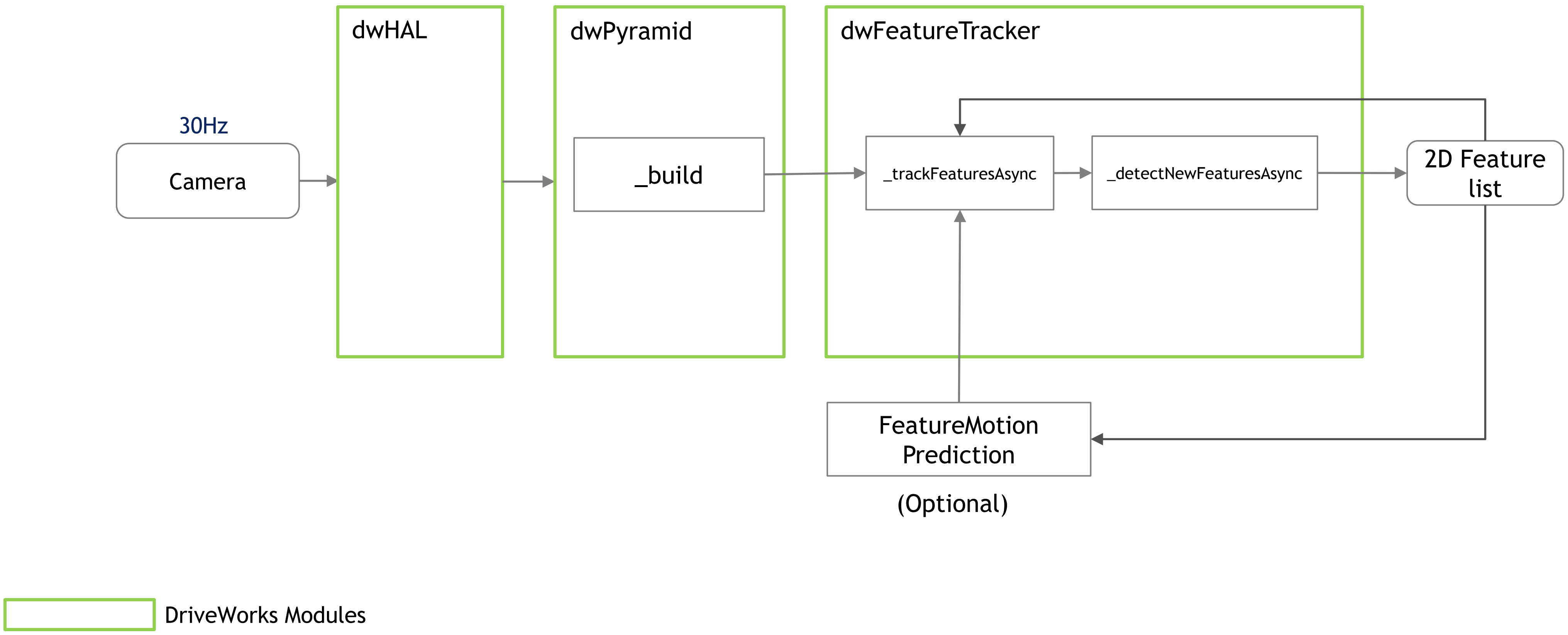
Future Enhancements/derivatives

- Local fisheye undistorsion
- Improved outlier filtering (bayesian)
- Feature descriptors (for multi view matching)

MODULES

- Pyramid generator
- 2D tracker with feature management

2D FEATURE TRACKER PIPELINE

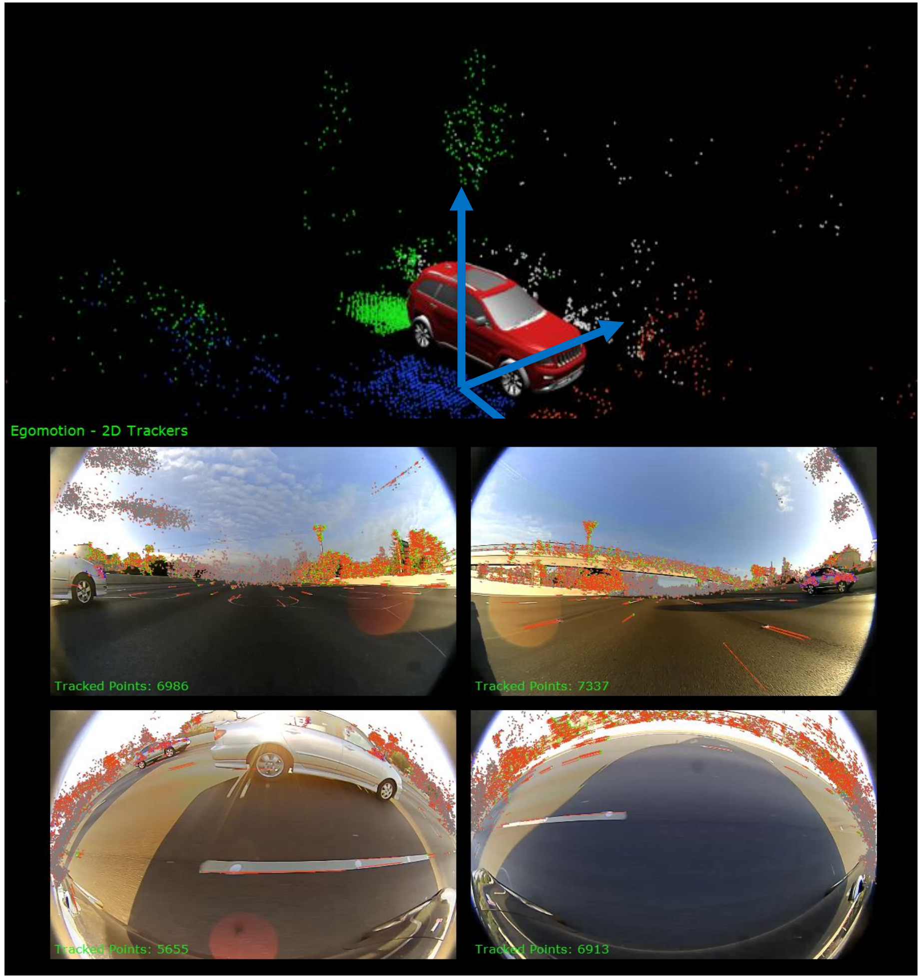


2D TRACKER KPIS

TX1	Runtime	Runtime for 4 Cameras	Memory Traffic	Memory Traffic for 4 Cameras	Number of Kernels
Pyramid Creation, 3 levels	291 us	1164 us	700 KB	2800 KB	2 per Camera and Frame
Feature Motion Prediction	7.5 us	30 us	50 KB	150 KB	1 per Camera and Frame
Frame to Frame Tracking	510 us	2040 us	750 KB	3000 KB	3 per Camera and Frame
Feature Detection	253 us	1012 us	600 KB	2400 KB	4 per Camera and Frame
Feature History Update	65 us	260 us	1100 KB	4400 KB	1 per Camera and Frame
Total	1126.5 us	4505 us	3200 KB	12750 KB	

Image size= 1280x800, max features=2000, window size = 10, iterations = 10, threshold = 0.01

STRUCTURE FROM MOTION (SFM)



Highly optimized 3D triangulation code from 2D tracker features

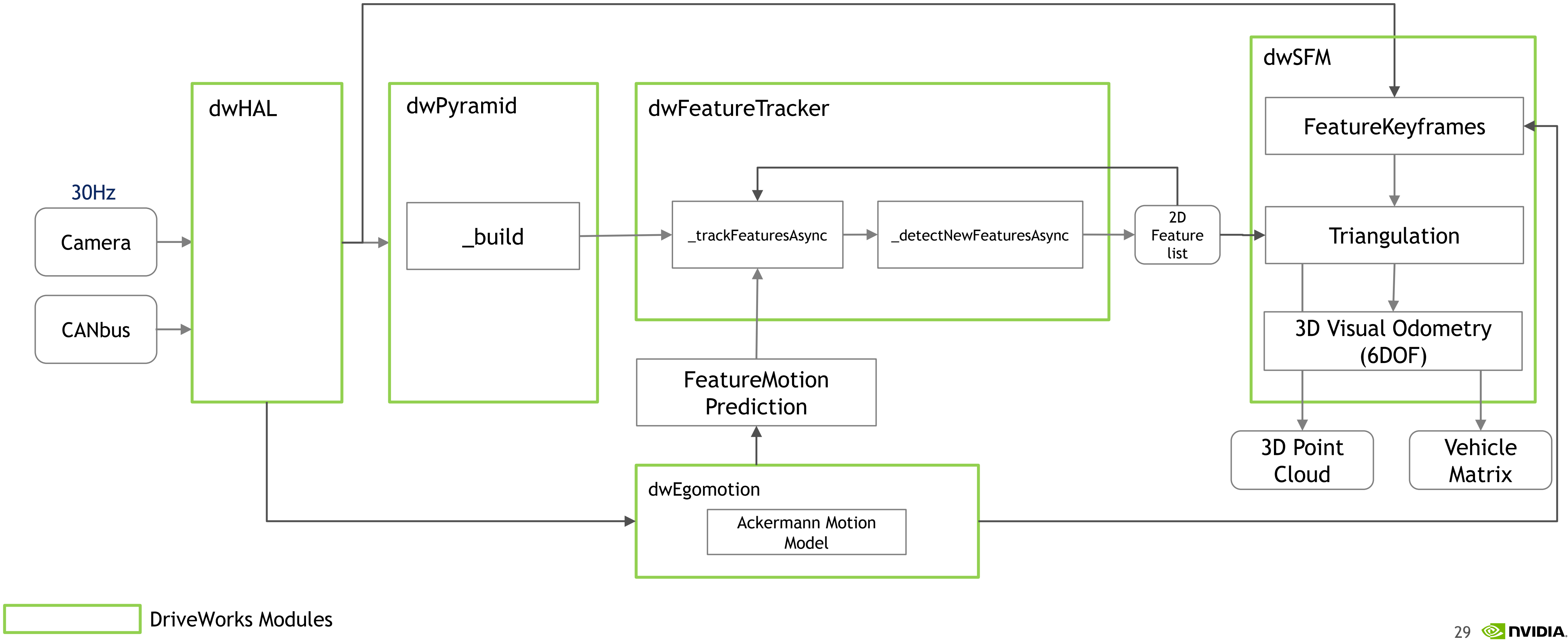
FEATURES

- Pyramid Sparse Optical Flow
- Harris Corner Detector
- Feature Managment
  - Multi-Frame Tracking
  - Uniform Feature Sampling
- Ackerman Motion Model
  - Feature Motion Prediction
  - 3D triangulation bootstrapping
- 6DOF Visual Odometry
- Multi-Baseline Triangulation

Future Enhancements/derivatives(post V0.1)

- Local fisheye undistorsion
- Improved outlier filtering (bayesian)
- Feature descriptors (for multi view matching)

SFM PIPELINE

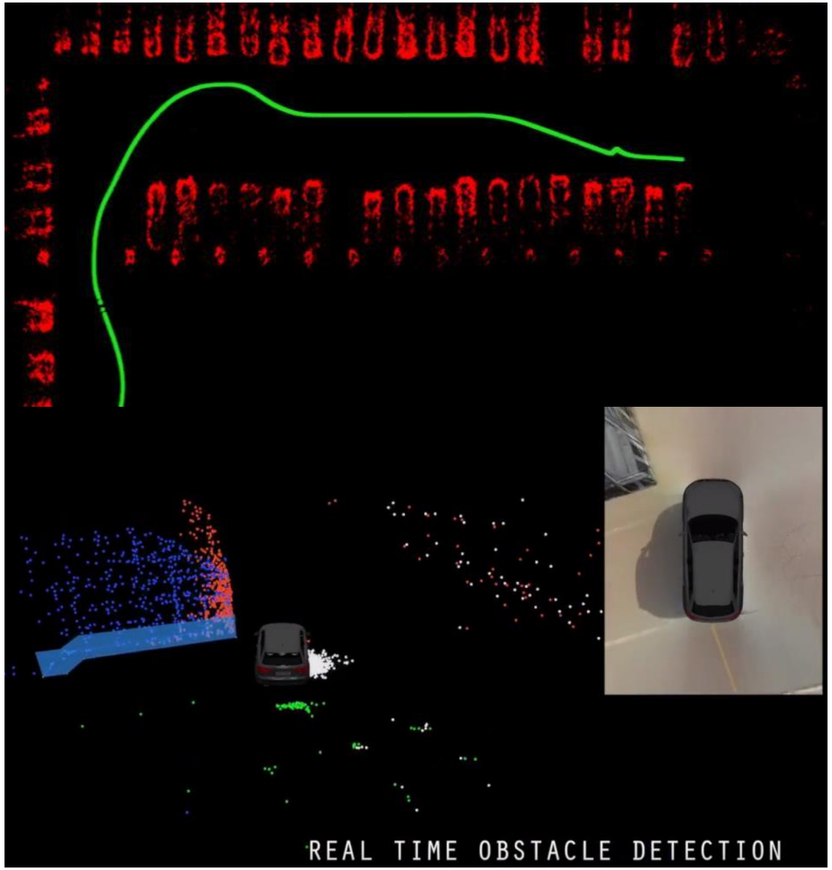


SFM KPI

TX1	Runtime	Runtime for 4 Cameras	Memory Traffic	Memory Traffic for 4 Cameras	Number of Kernels
2D Feature tracking	1126.5 us	4505 us	3200 KB	12750 KB	
Visual Odometry, Position on Ground Plane	20 us	20 us	20 KB	20 KB	1 per Bundle of Frames
Visual Odometry, 6 DOF Optimization	2250 us	2250 us	2200 KB	2200 KB	28 per Bundle of Frames
Triangulation	40 us	160 us	110 KB	440 KB	1 per Camera and Frame
Total	3436.5 us	6936 us	5530 KB	15410 KB	

Image size= 1280x800, max features=2000, window_size = 10, iterations = 10, threshold = 0.01

OCCUPANCY GRID FUSION



Generate accurate occupancy maps of surroundings of the vehicle for obstacles and free space detection

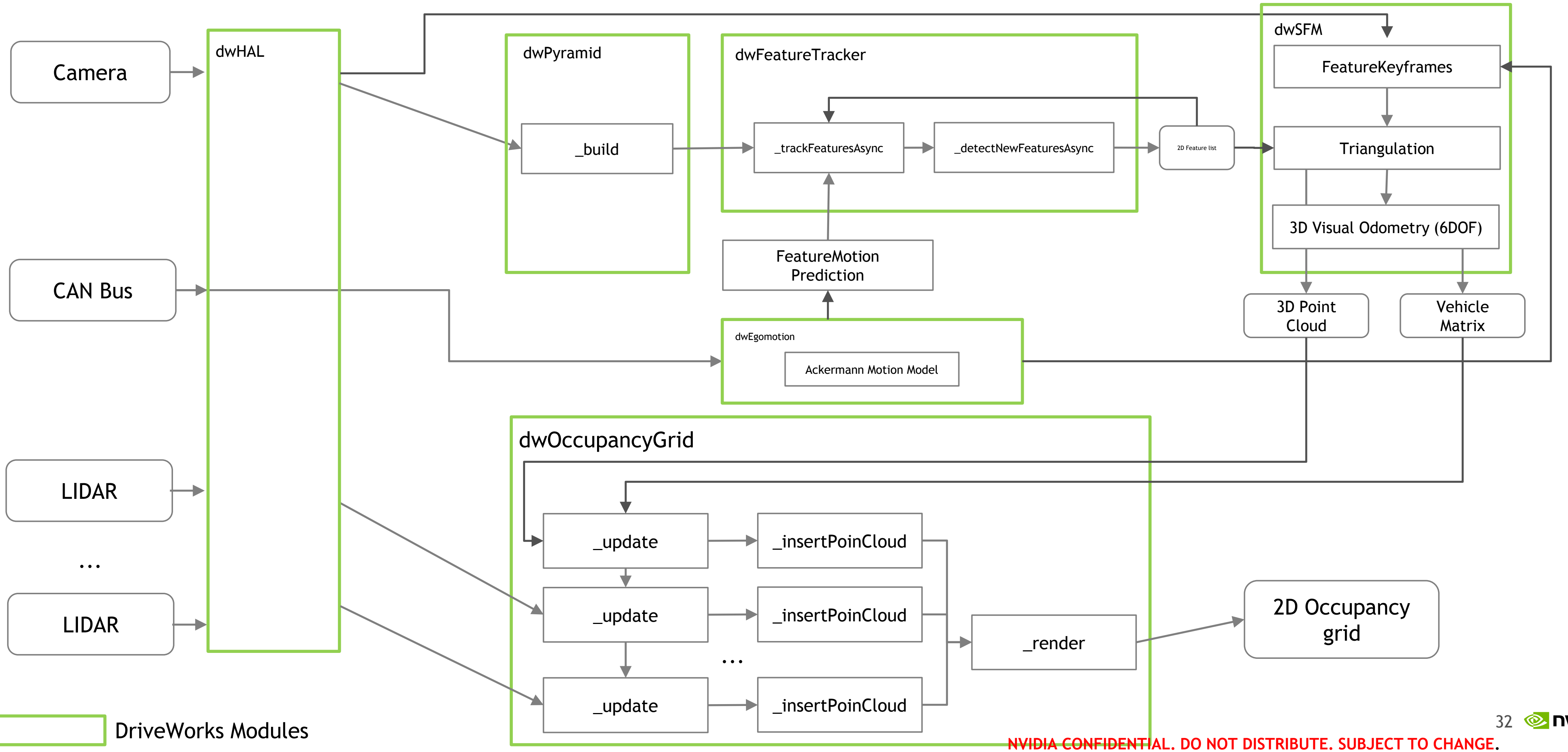
FEATURES

- 2D map generation from point clouds
- Aging and pseudo-probabilistic updates
- Basic blurring for distance field computation
- Basic point cloud clustering
  - 3D clustering
  - 2D connected components clustering

Future Enhancements/derivatives (post V0.1)

- Bayesian grids
- Radar, ultrasound and stereo
- Improved and optimal object clustering and detection
- Freespace detector

OCCUPANCY GRID FUSION

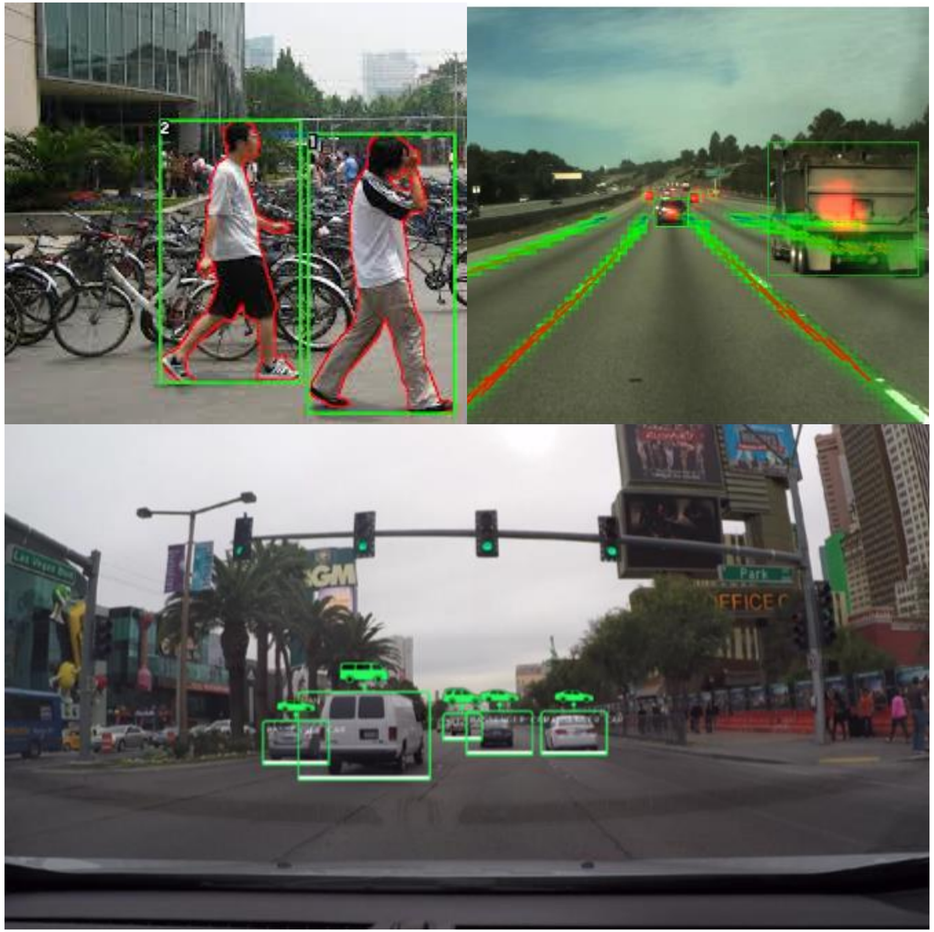


2D OBJECT TRACKER

2D feature based bounding box tracker. This module is to be used in conjunction with a costly detector (e.g. DNN)

Supports for
- Bounding box tracking
- Bounding box merging from external source

DNN INFERENCE ENGINE



Inference engine abstraction supporting Caffe and GIE optimized networks

FEATURES

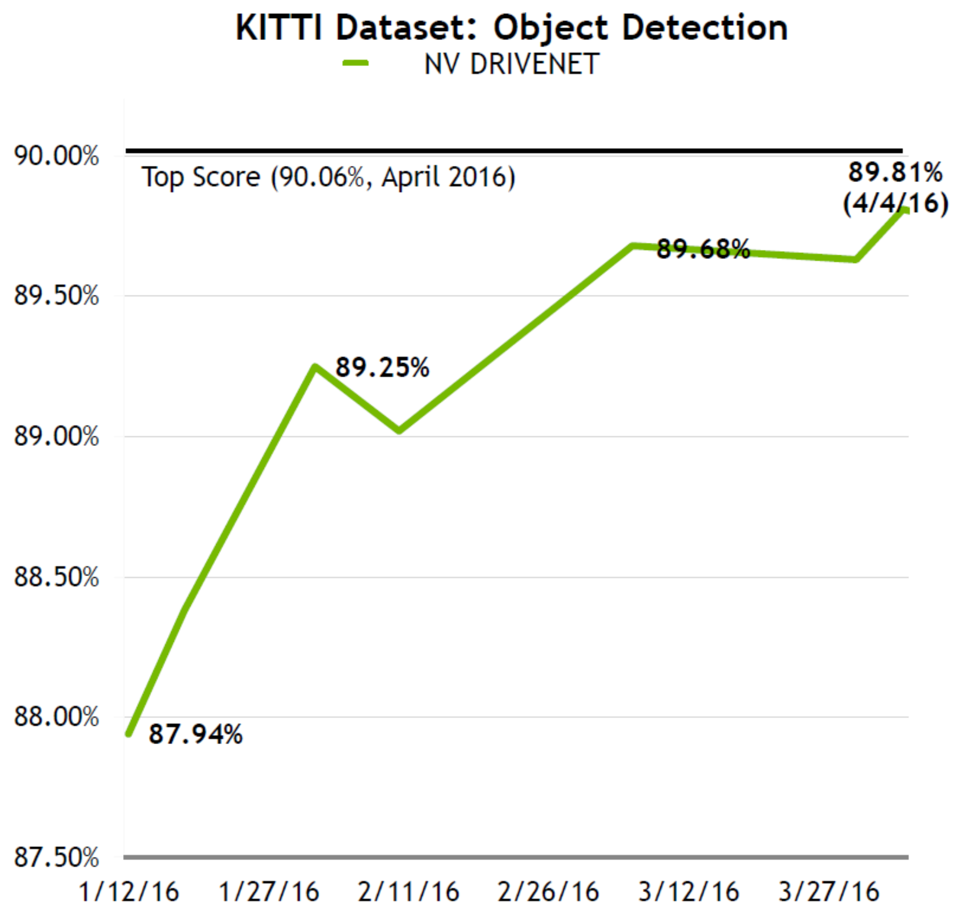
Various detectors for

- Unique interface
- No dependencies
- Data blob conditioning module for common cases
- GIE optimizer command line tool

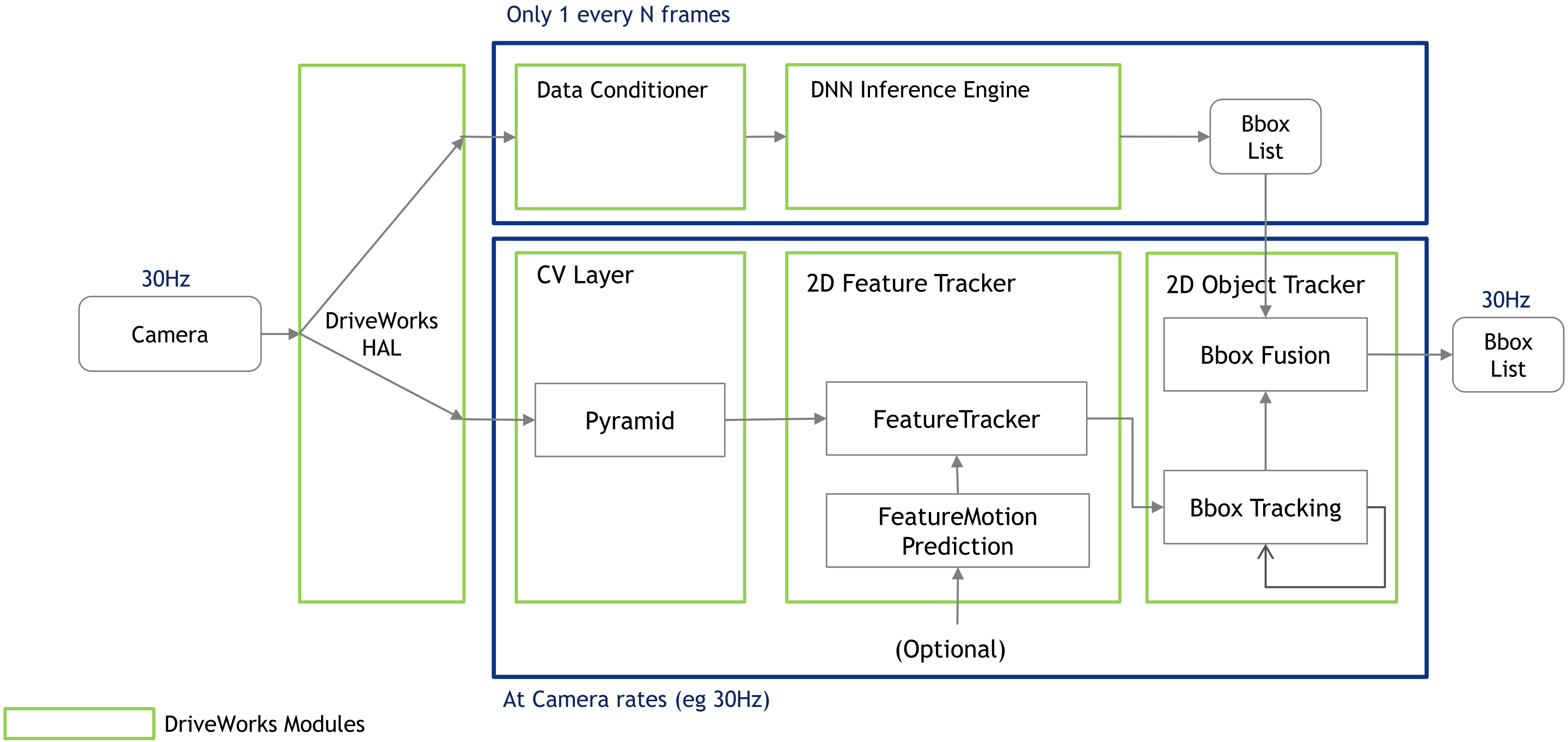
DNN KPI

Parker performance:

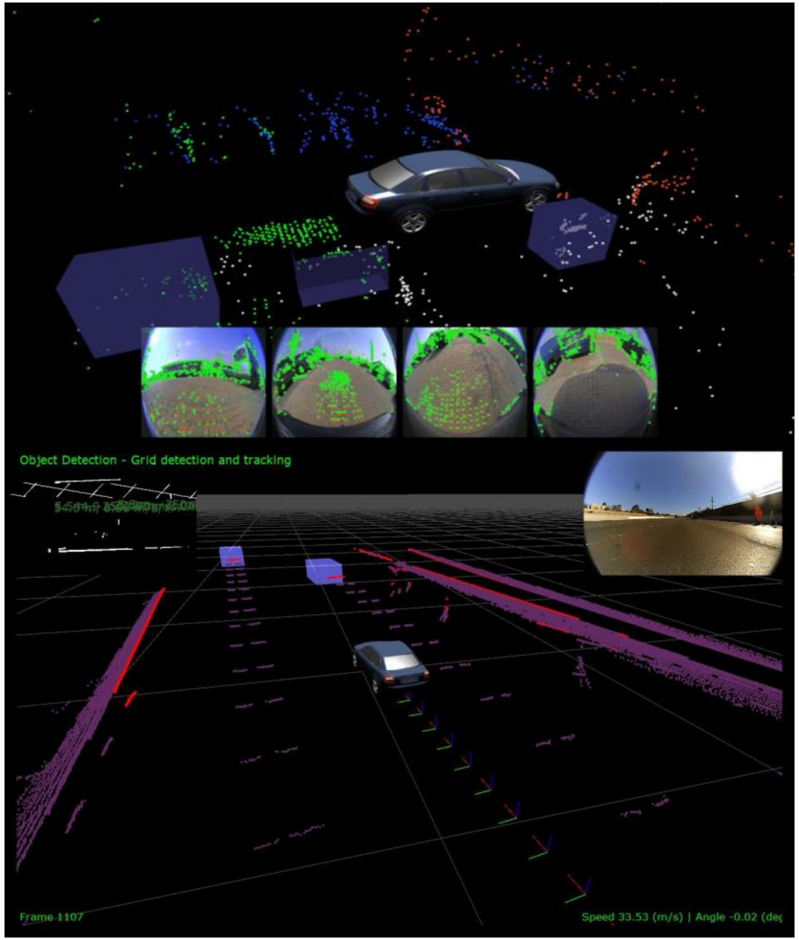
- DNN inference 11ms for 2 x 640x480
- Tracking performance 1.5ms



DNN WITH TRACKING



RENDERING HELPERS



GL/GLES based routines to enable rendering basic primitives in an easy non-GL manner. This is not a render engine.

Various helpers for

- Image thumbnails
- Text rendering
- 2D/3D Pointcloud rendering (color, textured)
- 2D/3D line rendering (color, textured)
- 2D/3D bounding boxes

NVIDIA