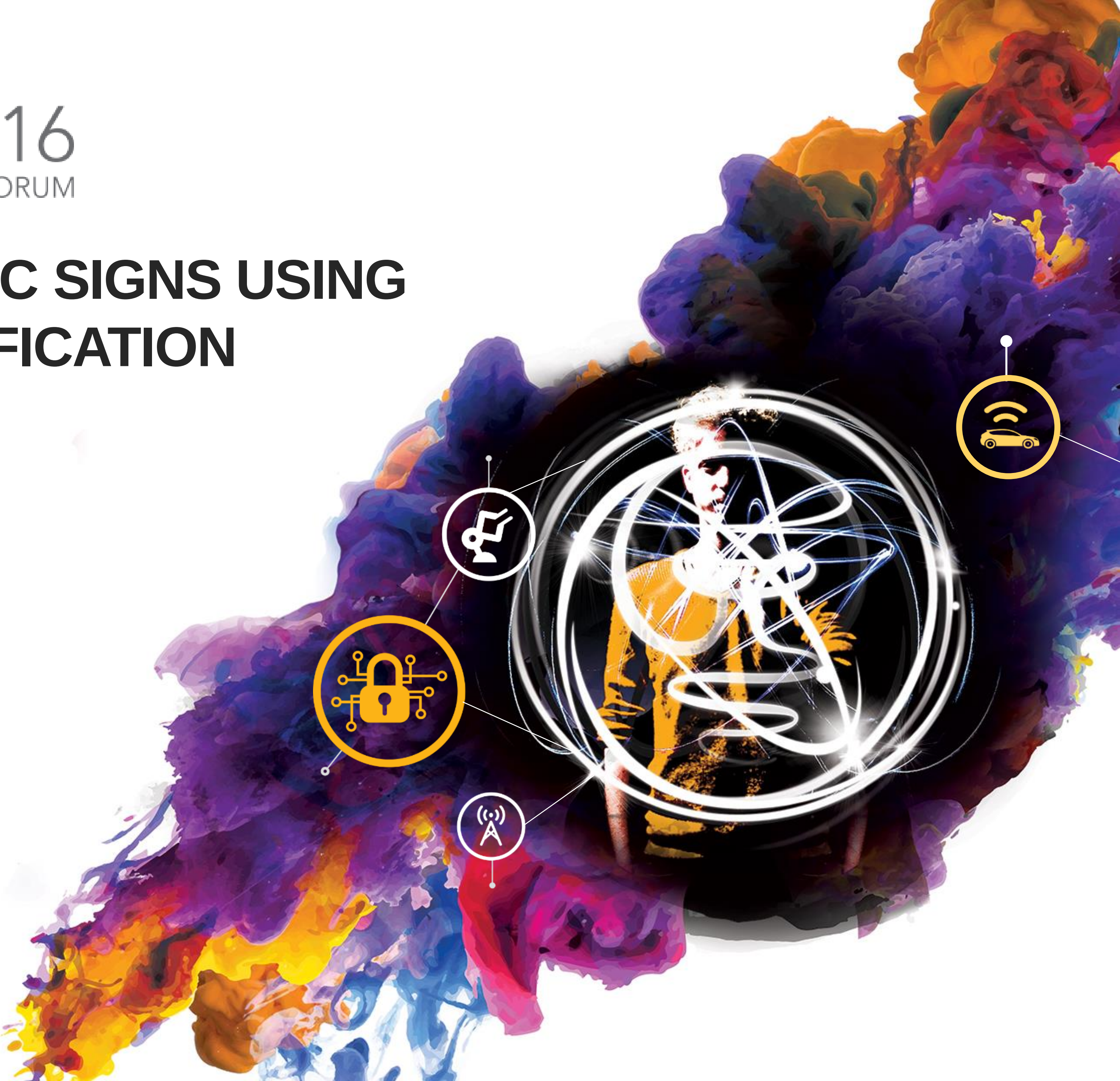
RECOGNITION OF TRAFFIC SIGNS USING CNN AND OTHER CLASSIFICATION ALGORITHMS

FTF-AUT-N1791



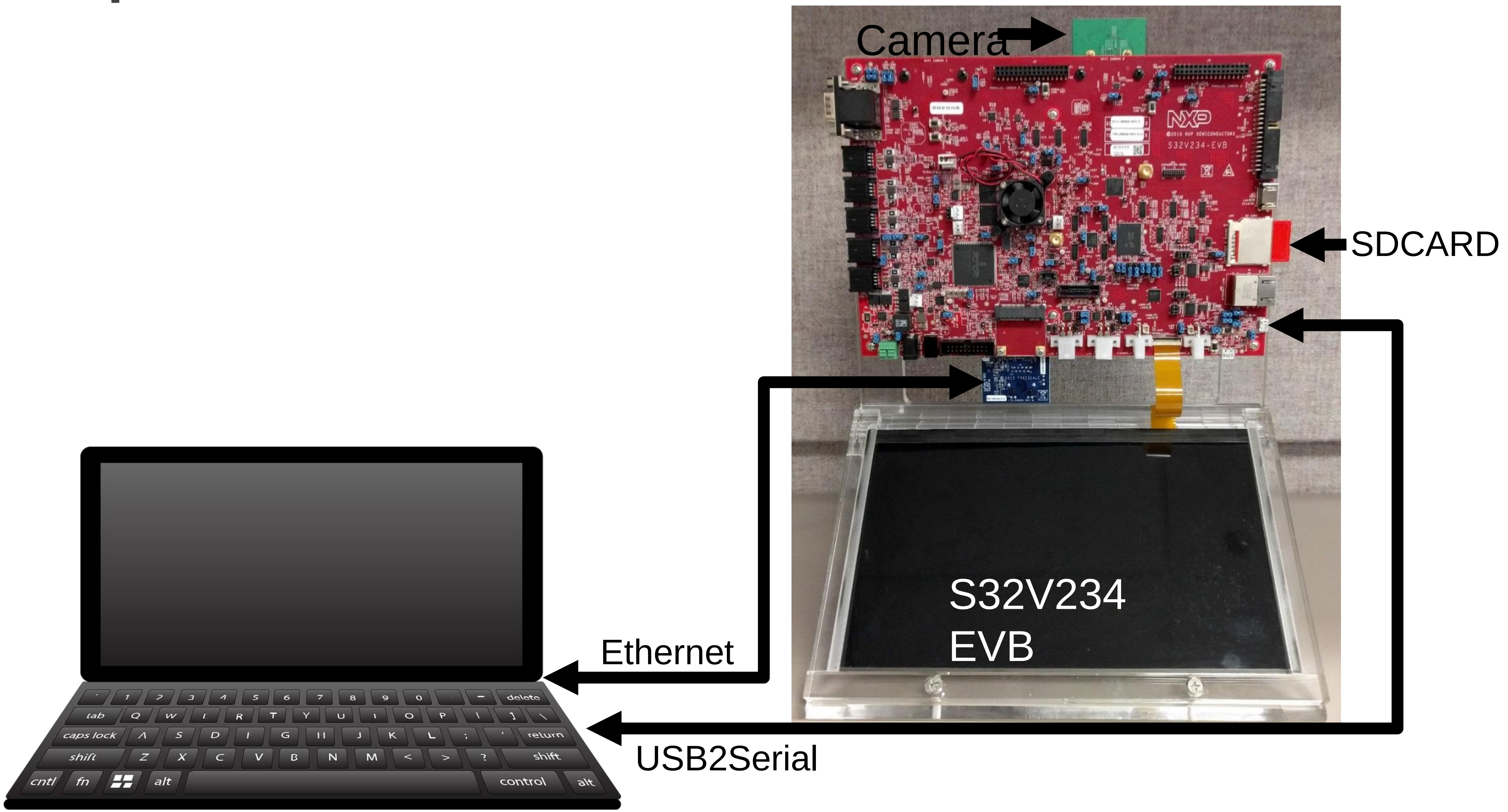
IOSEPH MARTINEZ
APPLICATIONS ENGINEER
FTF-AUT-N1791
17 MAY 2016

AGENDA

- Introduction
- Neural Networks
- Hands-On: FNN Hand written digits detection
- Convolutional Neural Networks
- Hands-On: CNN Hand written digits detection
- Practical Case: Lane Detection Algorithm
- Hands-On: Lane Detection

INTRODUCTION

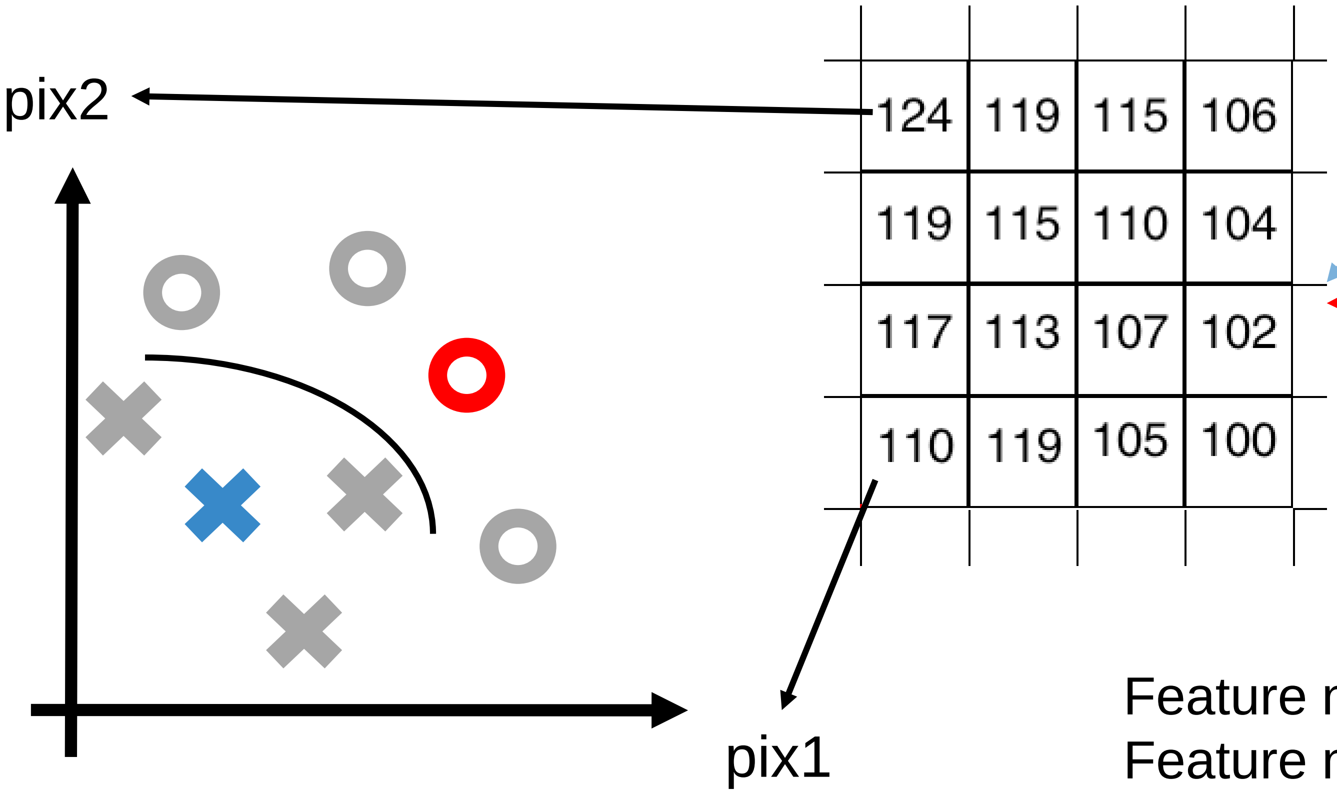
HW Setup



NEURAL NETWORKS

Object Classification: Car Detection

Machine Learning

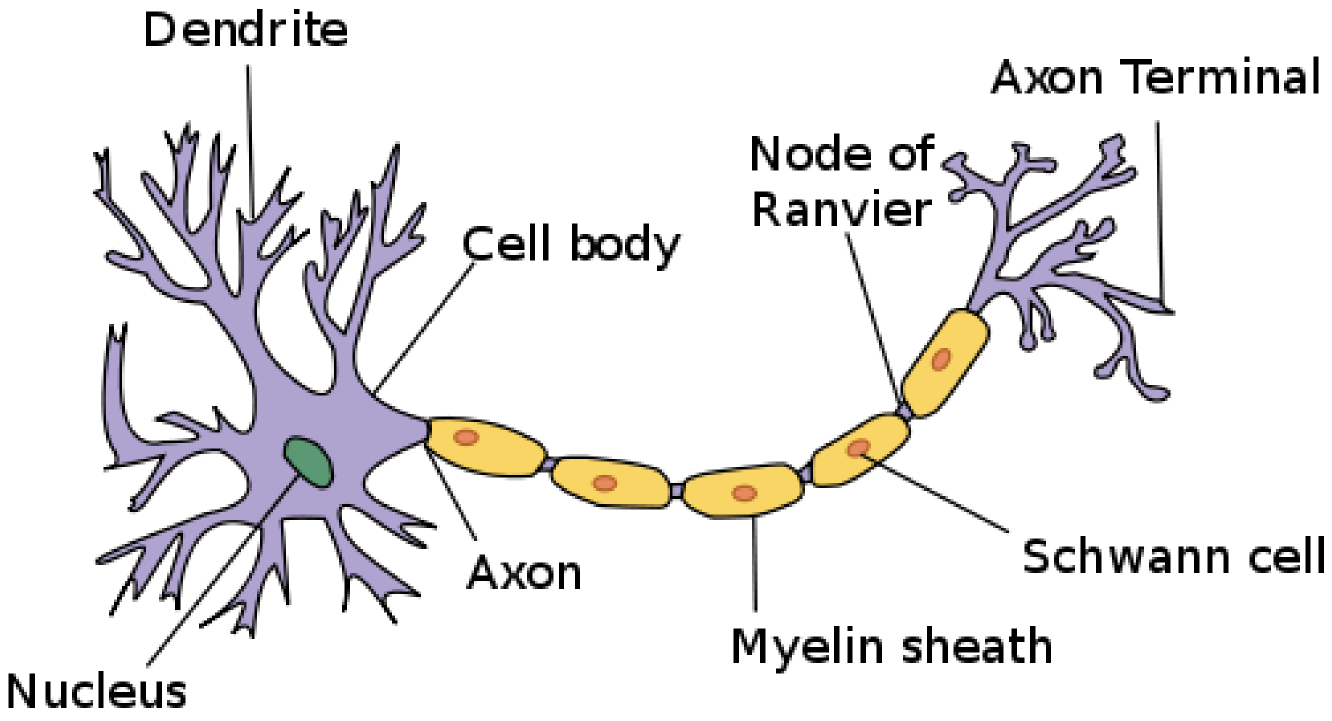


Feature number = Width x Height
Feature number = 30 * 30 = 900

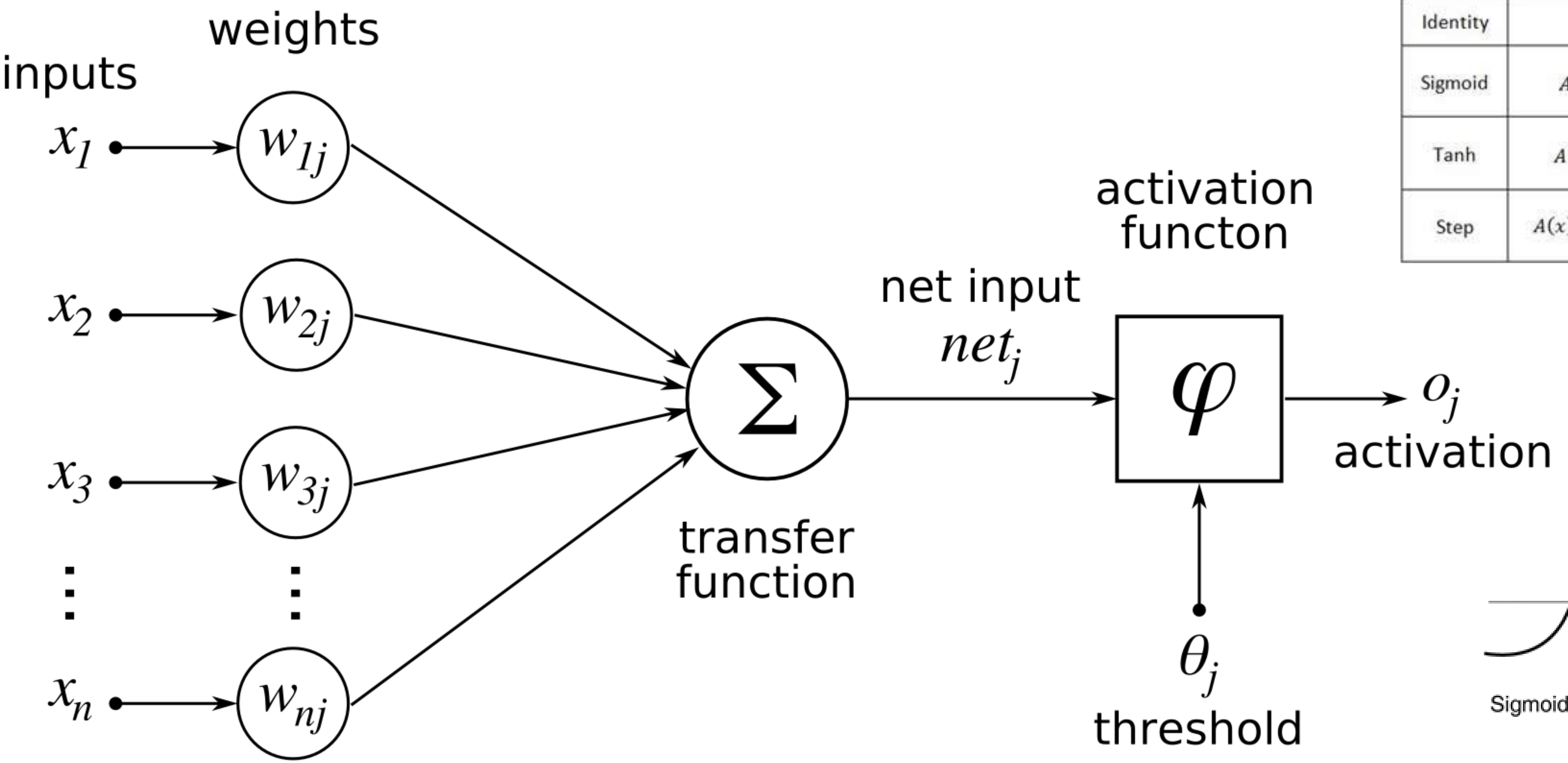
Neural Networks - Background

- Central nervous systems inspired the concept of artificial neural networks
- First models come back since 1943. Research stagnated around 70s
- Neural Networks resurged with the backpropagation algorithm that solved the X-OR problem.
- The approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing
- Support vector machines and other linear classifiers had overtook over some neural network applications

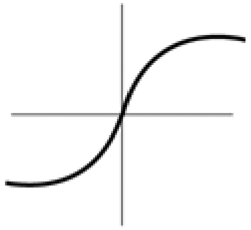
Neuron in the brain



Neural Network: Model

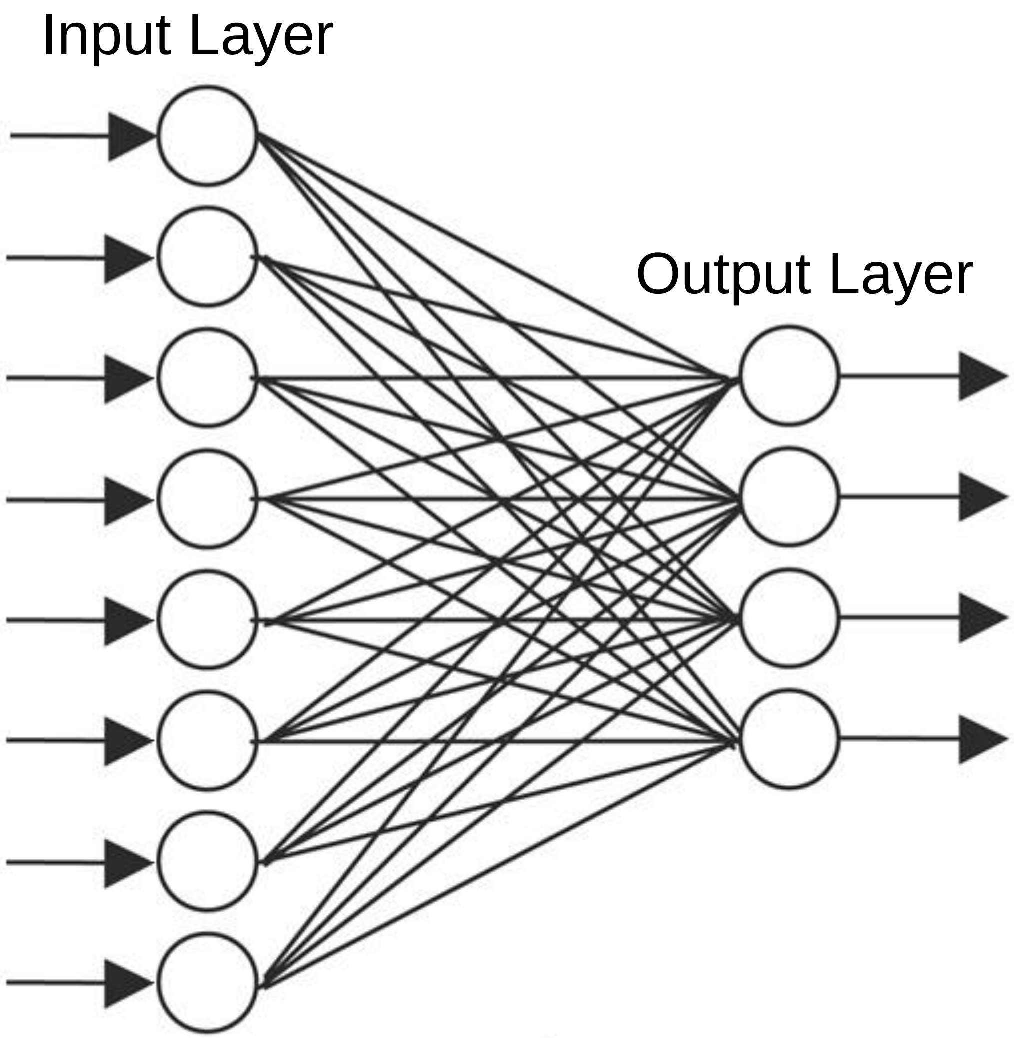


Activation Functions	
Name	**Formula**
Identity	$A(x) = x$
Sigmoid	$A(x) = \frac{1}{1+e^{-x}}$
Tanh	$A(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Step	$A(x) = \begin{matrix} -1 & if\ x < 0 \\ 1 & if\ x \geq 0 \end{matrix}$



Sigmoid Function

The single-layer Neural Network



Neural Networks: NAND

x1	x2	y
0	0	1
0	1	1
1	0	1
1	1	0

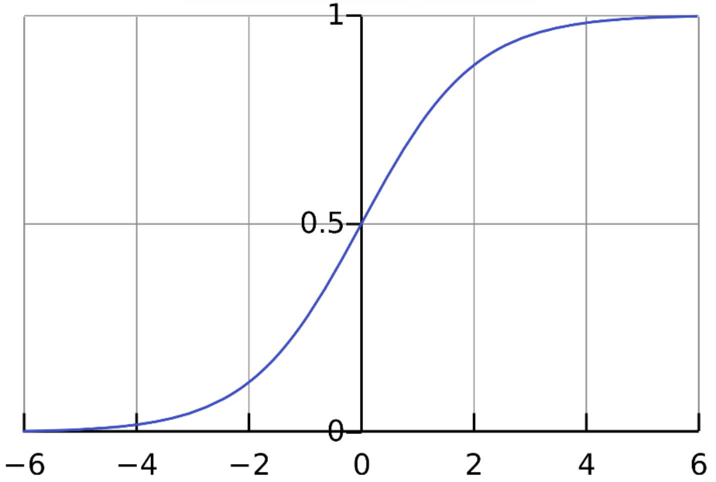
$$a_1^{(2)} = g(\ 20\ \cancel{+\Theta_{11}^{(1)}x_1}\ \cancel{+\Theta_{12}^{(1)}x_2}\ )$$

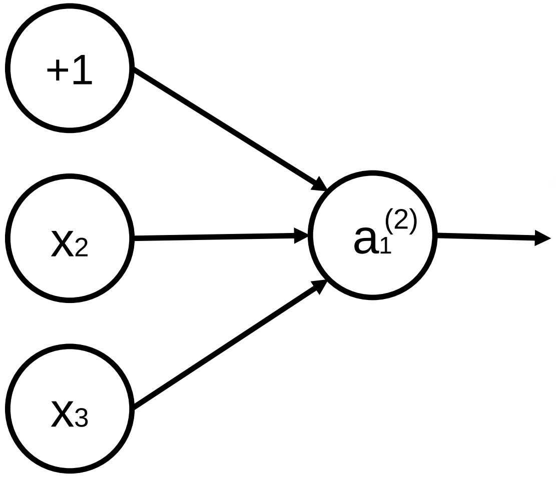
$$a_1^{(2)} = g(\ 20\ \cancel{+\Theta_{11}^{(1)}x_1}\ -\ 15\ )$$

$$a_1^{(2)} = g(\ 20\ -\ 15\ \cancel{+\Theta_{12}^{(1)}x_2}\ )$$

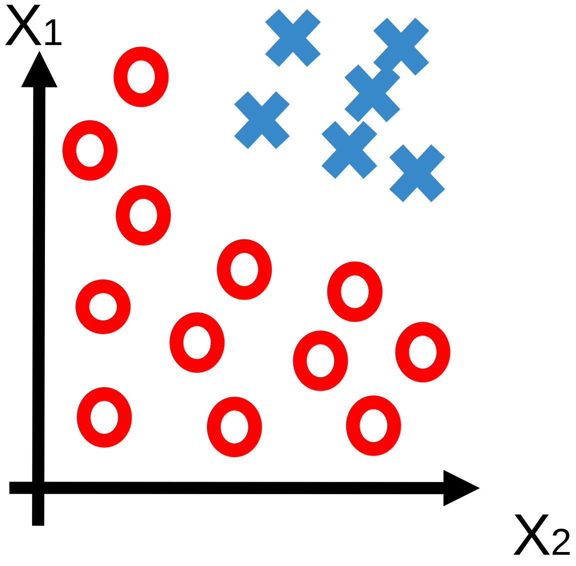
$$a_1^{(2)} = g(\ 20\ -\ 15\ -\ 15\ )$$

$$a^{(2)} = g(z^{(2)})$$



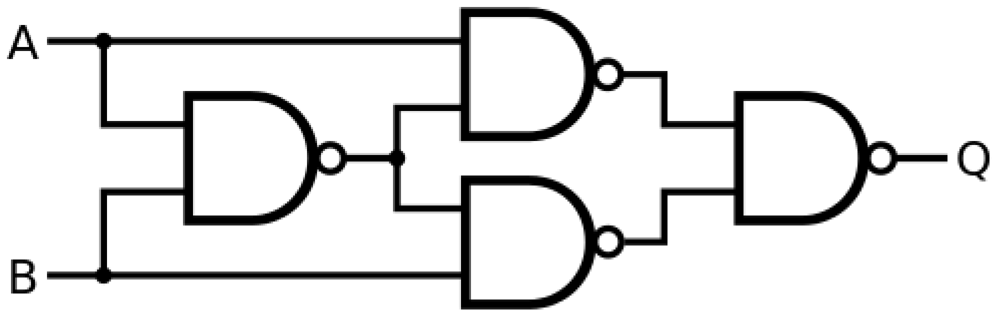


$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2\ )$$

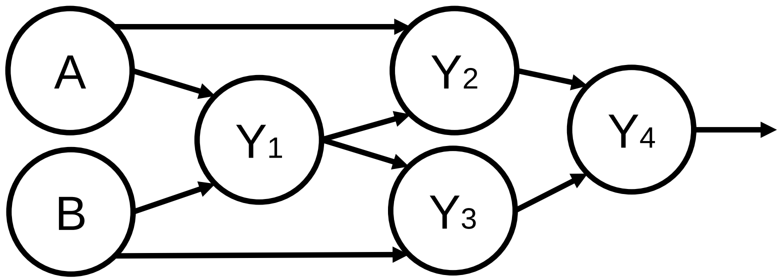


The XOR Problem

- Single layer networks or perceptrons were not very popular due the limitations to solve nonlinear problems.
- Multilayered networks can solve nonlinear problems but need to be trained and that was a new problem.
- It was not until the backpropagation algorithm was discovered when NN became particularly useful

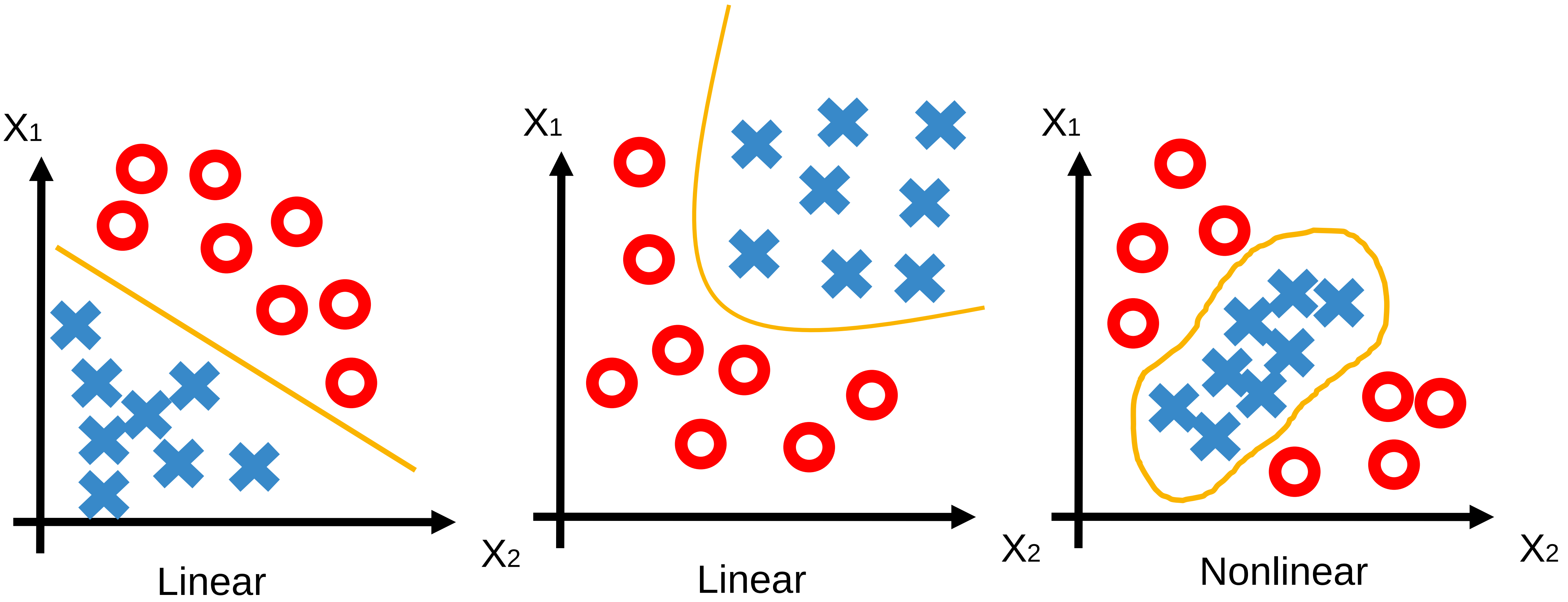


Logic XOR with NAND gates



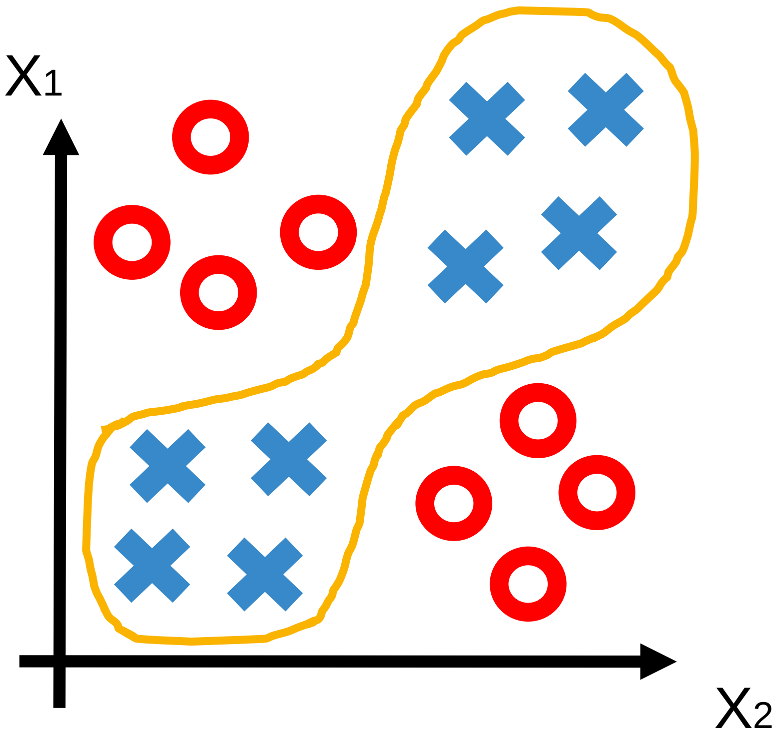
XOR with multilayered NN

Linear vs. Nonlinear Statistical Models

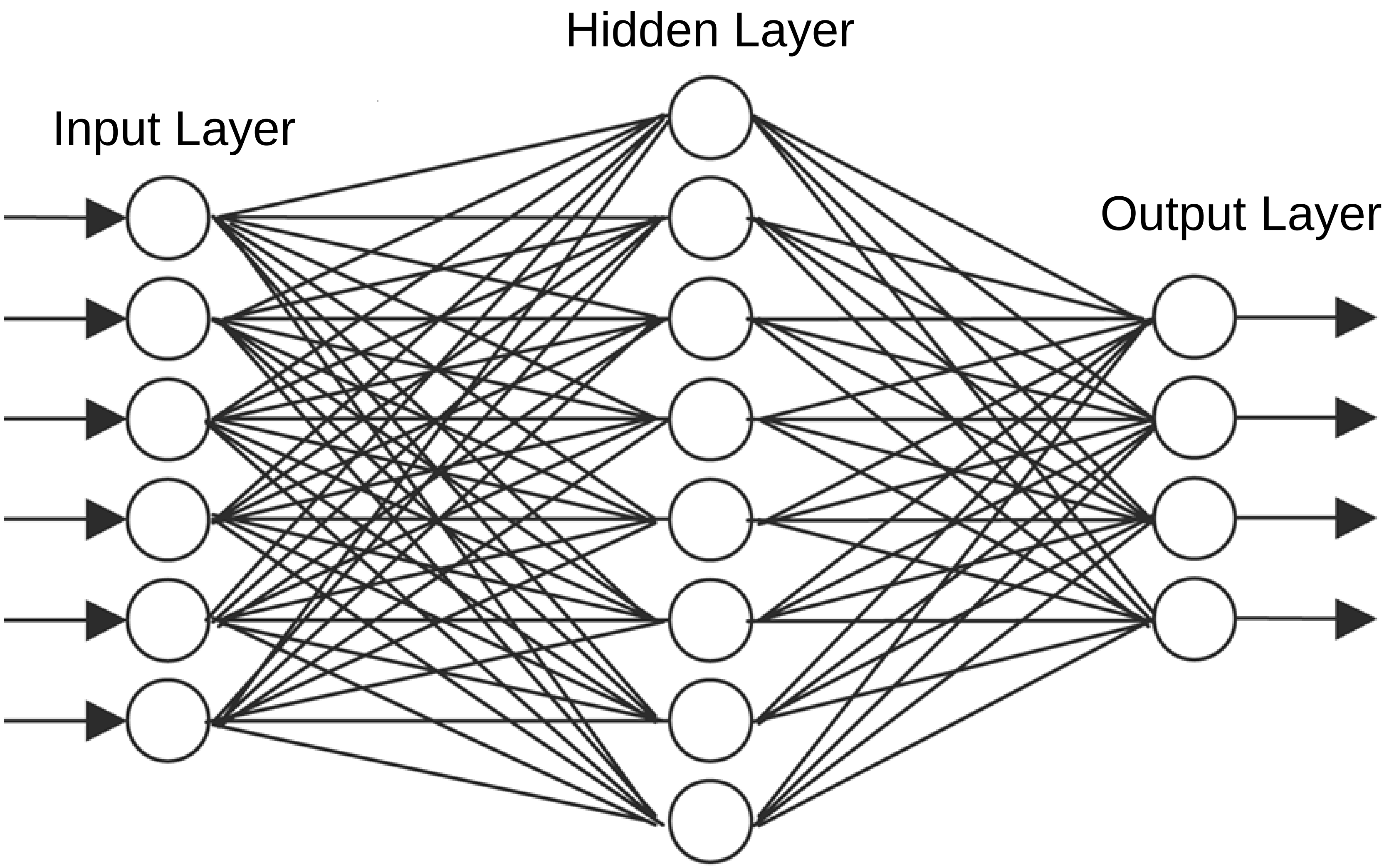


Neural Networks: XOR/XNOR

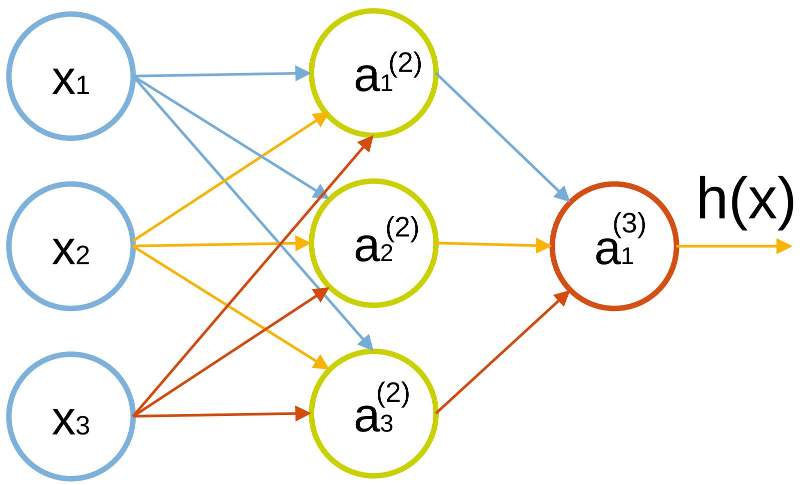
- XOR is a non linear function



Fully Connected Multi-layer Neural Networks



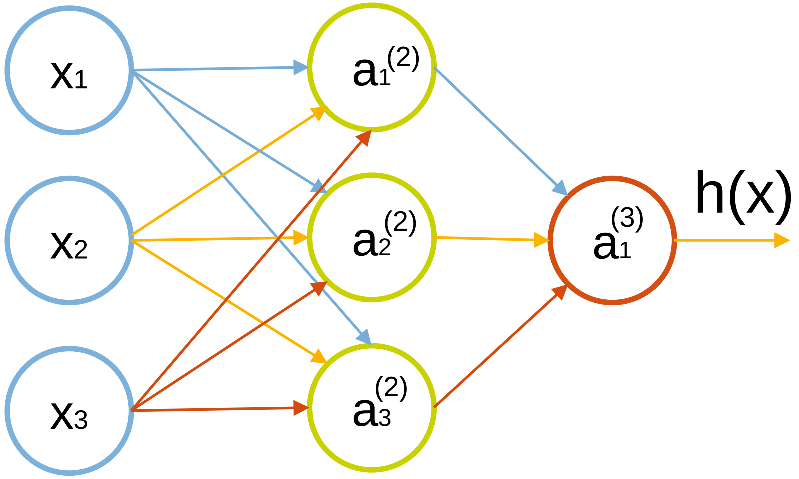
Neural Network: Mathematical Definition



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Neural Network: Mathematical Definition

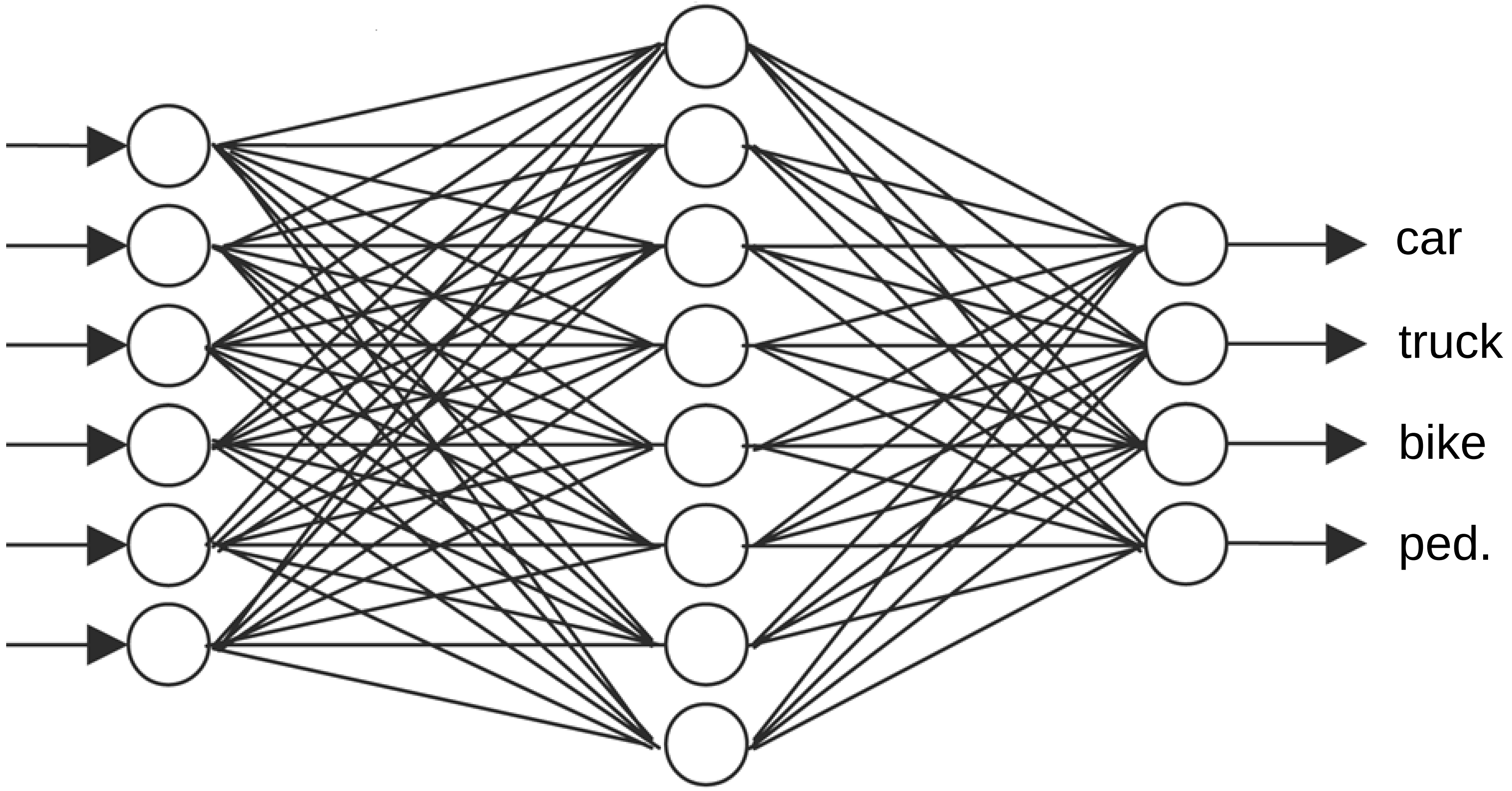


$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

Multiclass Classification



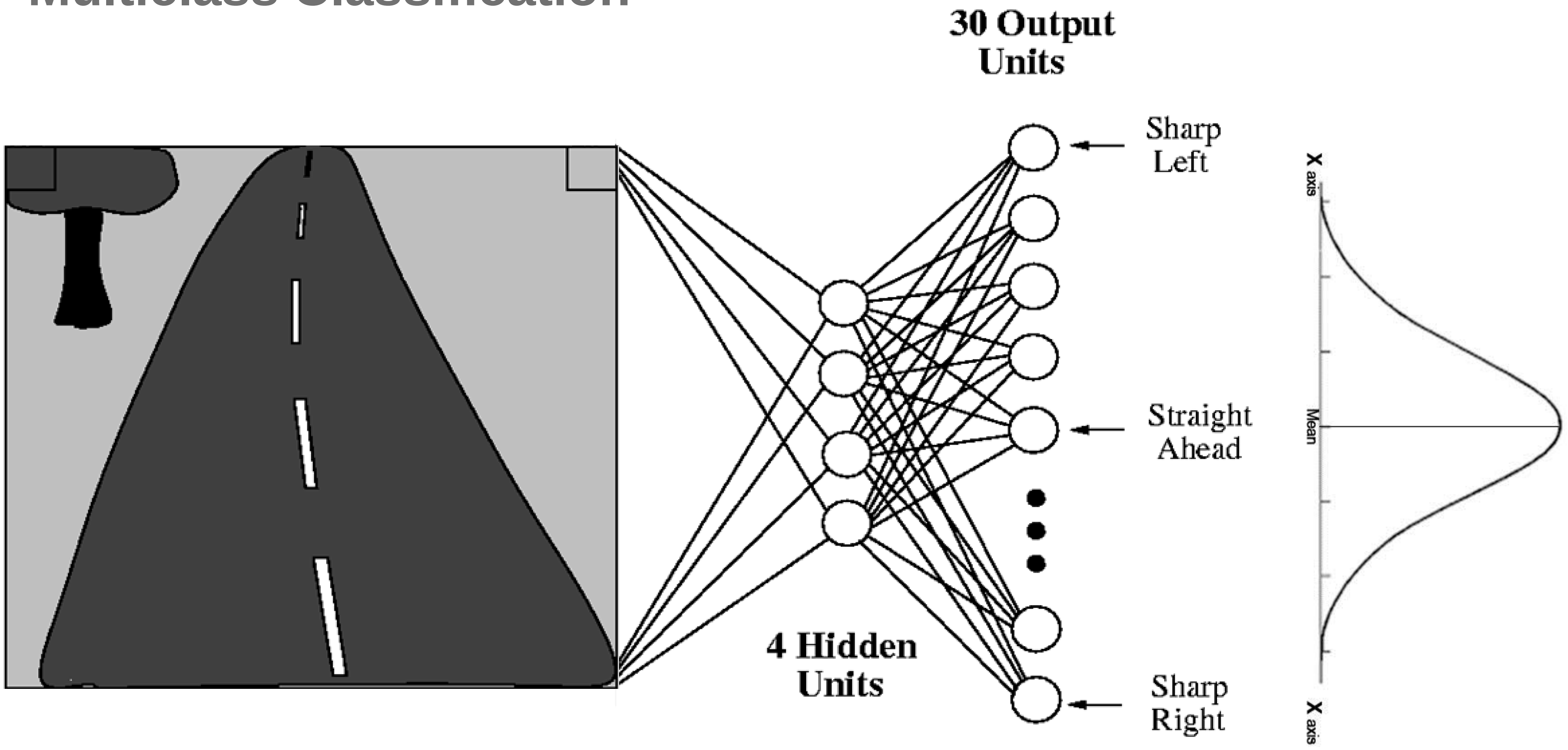
$$h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

$$h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

$$h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

$$h_\Theta(x) \approx \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Multiclass Classification



TRAINING

Cost Function

- Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

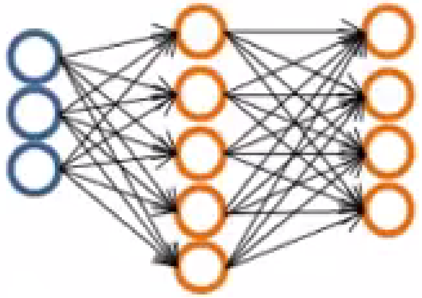
- Neural network:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

- Optimization:

$$\min_{\Theta} J(\Theta) \qquad\qquad \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

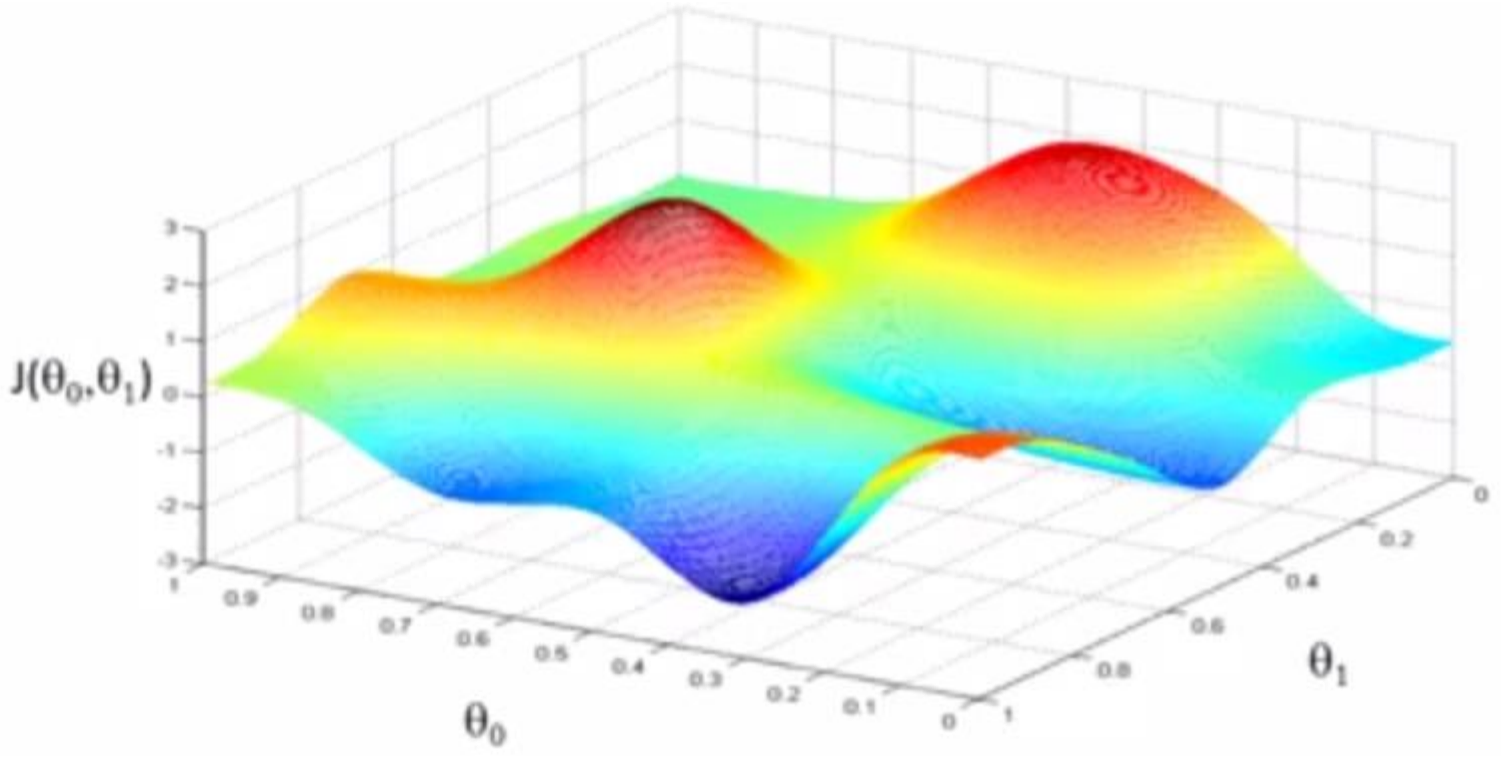
Backpropagation Algorithm



- The backpropagation algorithm is used to calculate the gradient
- The following steps are required:
  1. Perform forward propagation (weights with random data initialization)
  2. Calculate delta errors for the last layer: $\delta^{(L)} = a^{(L)} - y$
  3. Calculate delta errors of the previous layer: $\delta^{(l-1)} = (\Theta^{(l-1)})^T \delta^{(l)} .* g'(z^{(l)})$
  4. Accumulate the partial derivatives for each training example: $\triangle^{(l)} := \triangle^{(l)} + \delta^{(l+1)} a^{(l)T}$
  5. Finally the gradient can be calculated by:

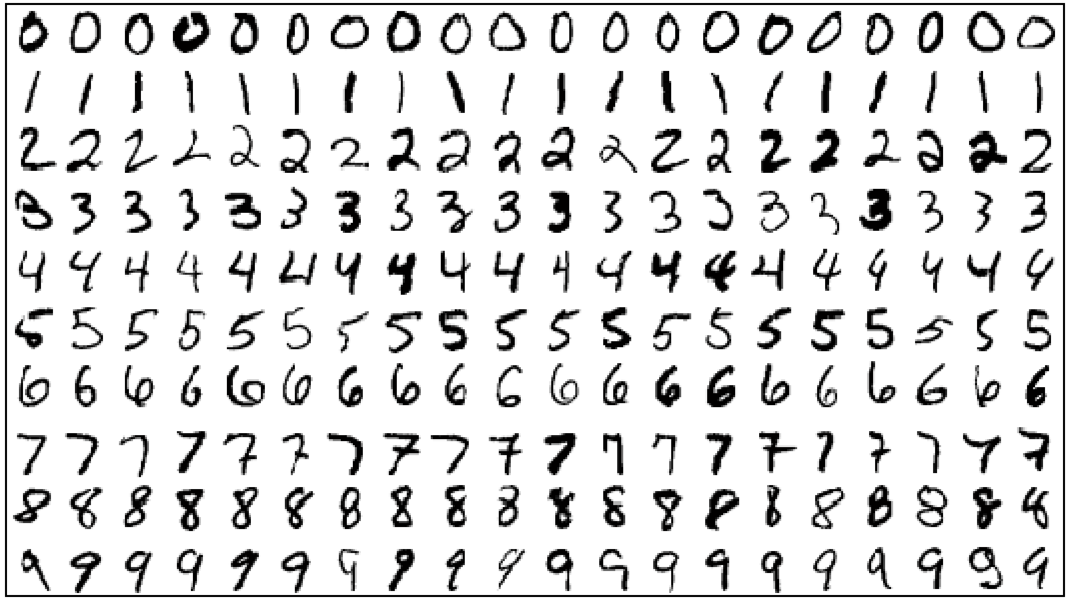
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) \begin{array}{ll} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ := \frac{1}{m} \triangle_{ij}^{(l)} & \text{if } j = 0 \end{array}$$

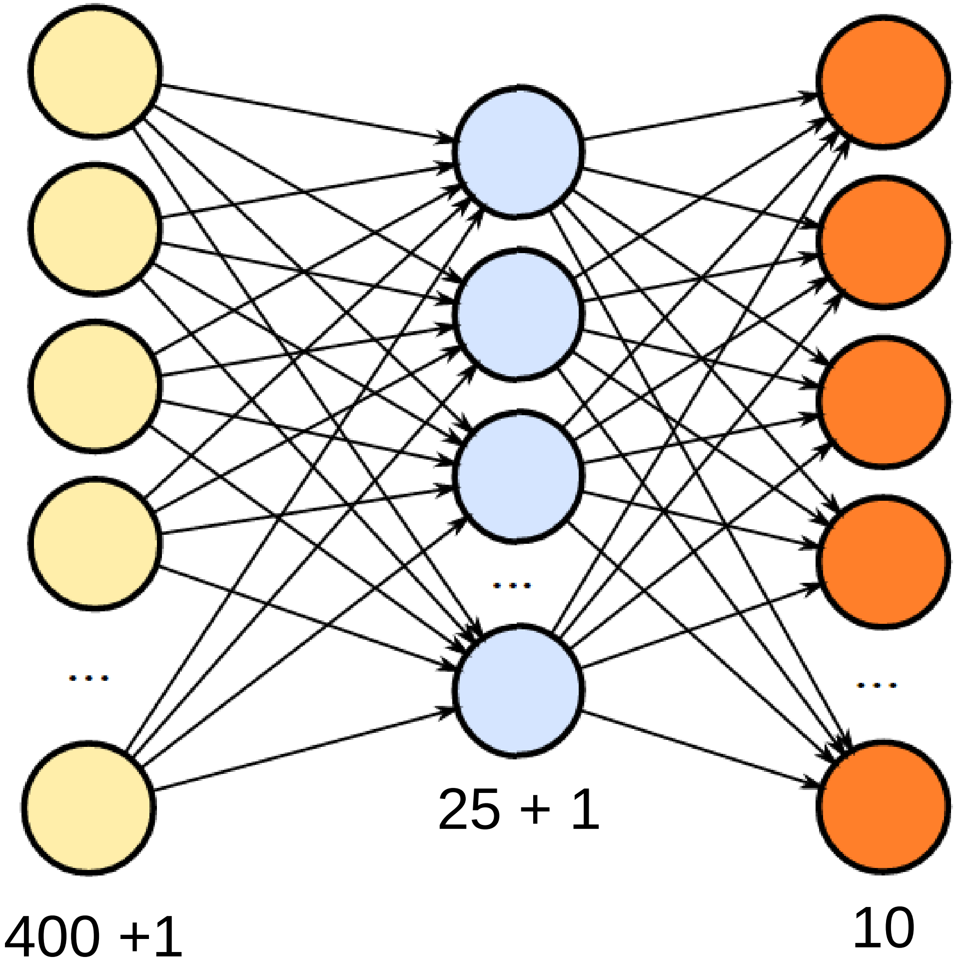
Cost Function



HANDS-ON: FNN HAND WRITTEN DIGITS DETECTION

Neural Networks – Hand-written digits





Neural Networks – Hand-written digits

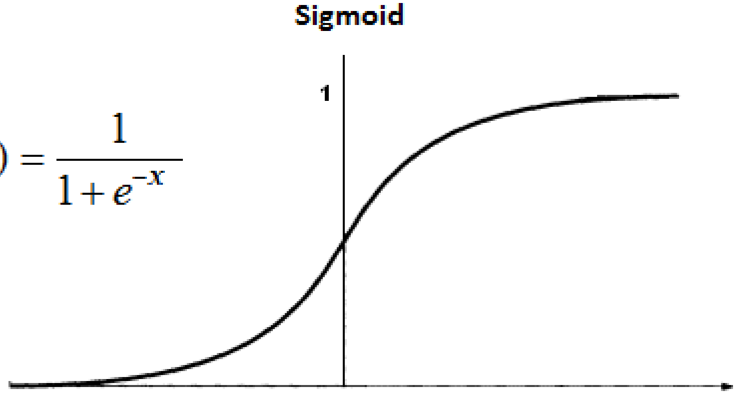
- On you Host:
  - Open and edit main.cpp to perform Neural Network on your camera input
    - gedit ~/s32v234/demos/nn1/src/main.cpp
  - You need to perform the following edits:
    1. Write the activation function code (sigmoid)
    2. Manipulate the input so the numbers are more distinguishable
    3. Based on the result of the last layer, determine the detected number

  - Some Hints:
    - OpenCV Mat objects allow matrix algebra be applied to them. You can do scalar summation and multiplication with + and * operators, cv::exp(src,dst) gives you the element wise exponent value
    - Contrast or threshold can improve the visualization of the numbers
    - You can obtain data from a cv::Mat object by doing: mat.at<float>(row,col). The result layer, h22, has 1 row and 10 columns

Neural Networks – Hand-written digits

Training Example Sample

$$f(x) = \frac{1}{1+e^{-x}}$$



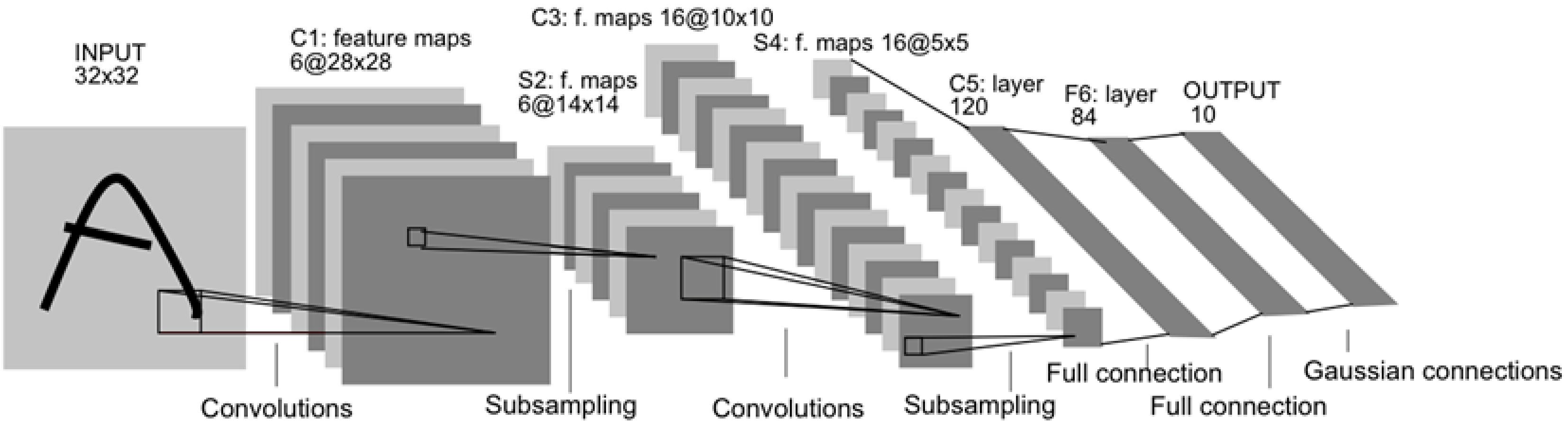
Neural Networks: Step 2

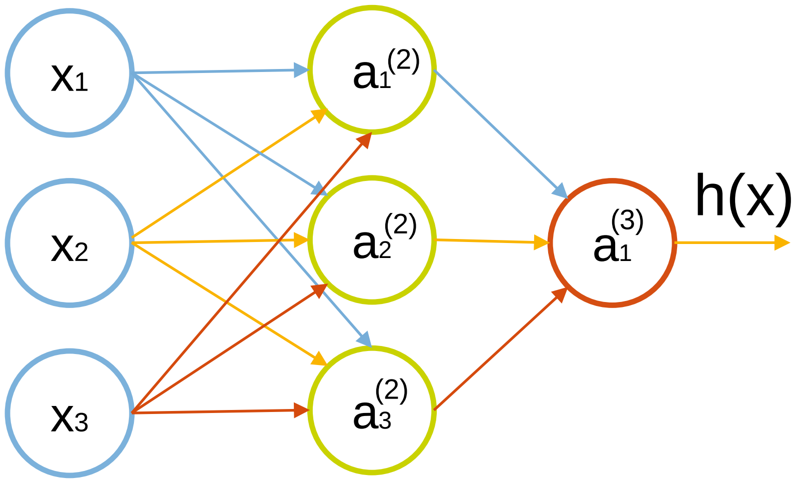
- On Your Host:
  - Build your application:
    - cd ~/s32v234_sdk/demos/N1791_NN/build-v234ce-gnu-linux-d/
    - ./build.sh
  - Copy the generated binary to your Network File System:
    - cp isp_csi_dcu.elf ~/rootfs/s32v234/demos/

- On Your Target (Serial Console):
  - Stop the previous demo and run the generated binary:
    - ../s32v234/demos/isp_csi_dcu.elf
  - Observe the results on the screen

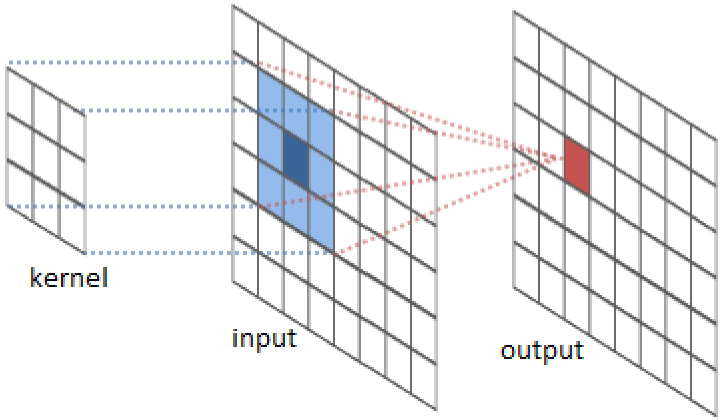
CONVOLUTIONAL NEURAL NETWORKS

Fully Connected NNs vs Convolutional NNs

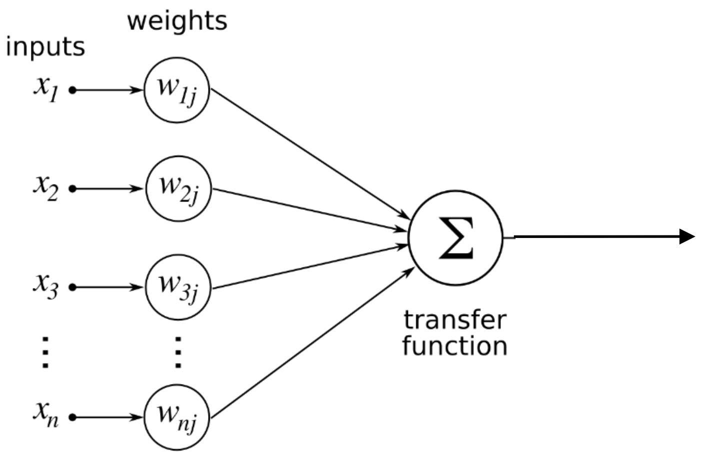




Convolution in a ConvNet



=

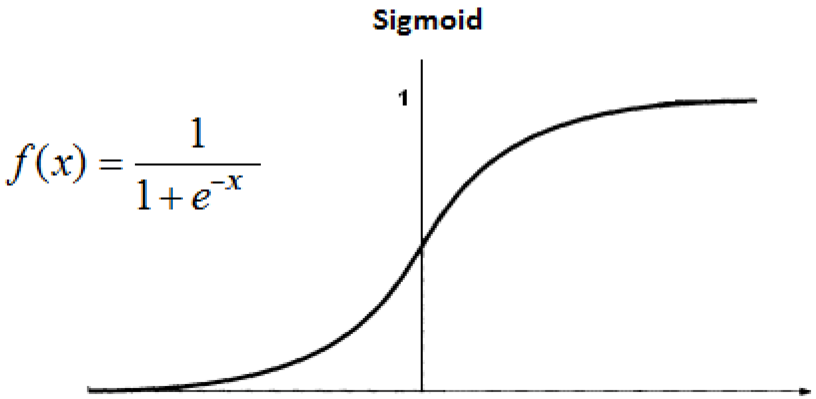


$$Y_n = K_0{*}X_{n+0} + K_1{*}X_{n+1} + K_2{*}X_{n+2} + K_3{*}X_{n+3} + \ldots + K_8{*}X_{n+8}$$

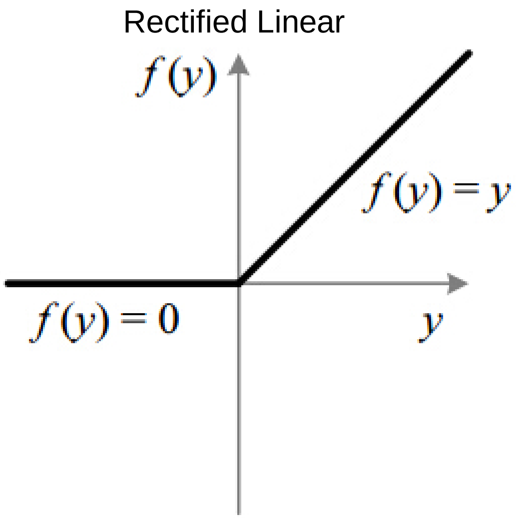
$$Y = K_1{*}X_1 + K_2{*}X_2 + K_3{*}X_3 \ldots + K_n{*}X_n$$

Rectified Linear Function (ReL)

- Instead of the Sigmoid Unit, ConvNets use a ReLu as activation function.

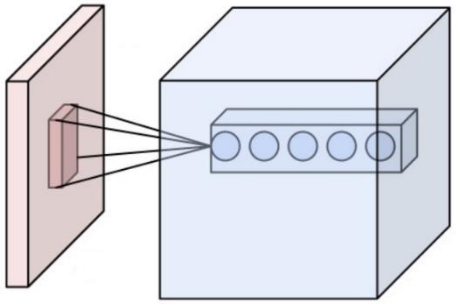


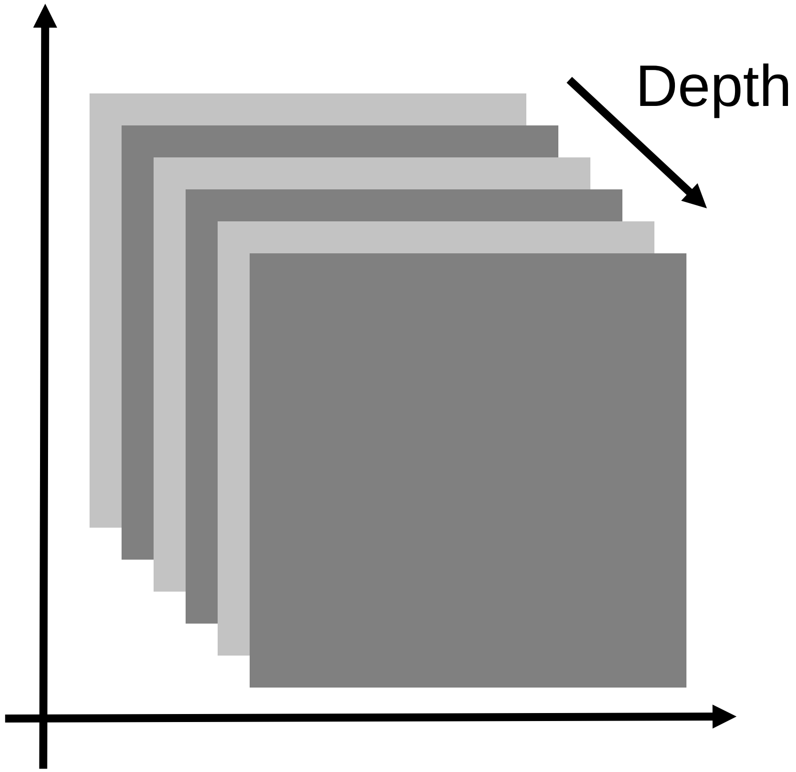
- Range is [0,1]
- Gradient vanishes
- Useful to model probability

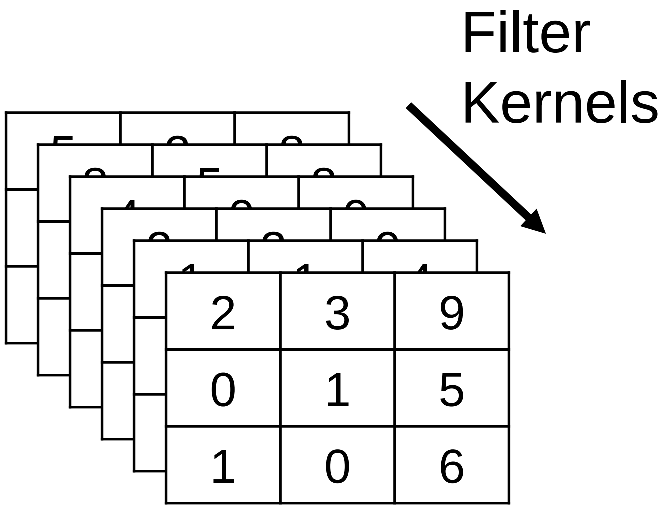


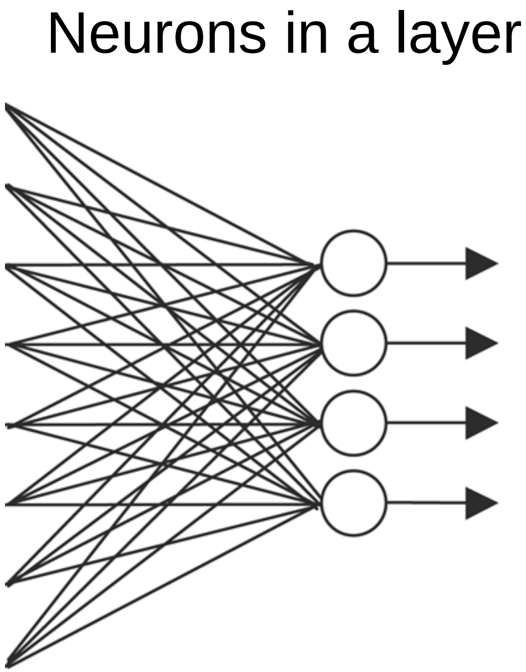
- Range is positive real numbers
- Gradient does not vanishes
- Easy to calculate

ConvNets have 3 Dimensions



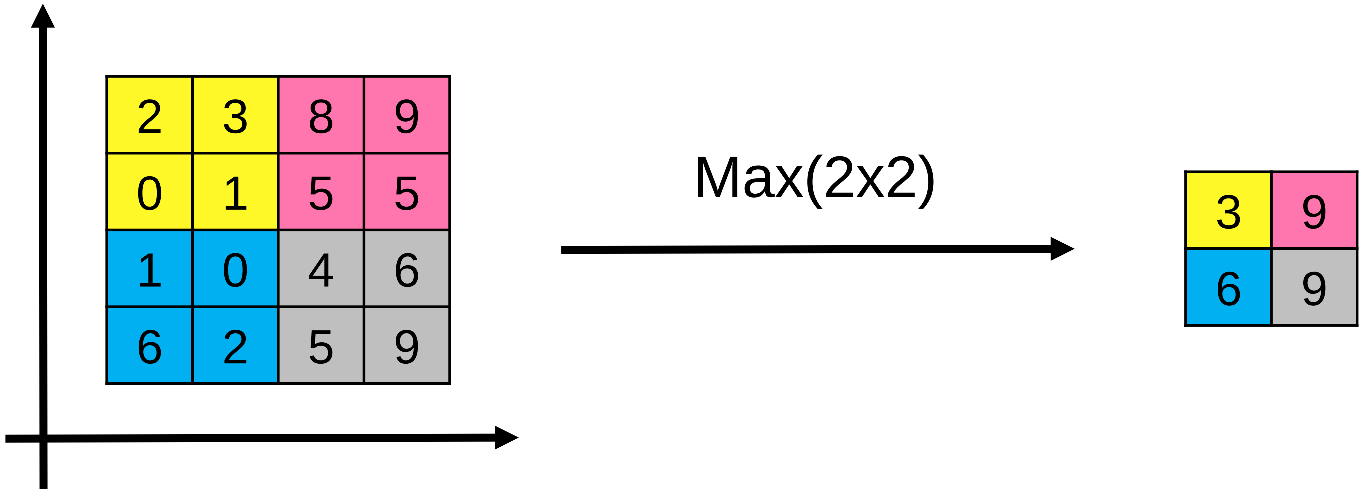






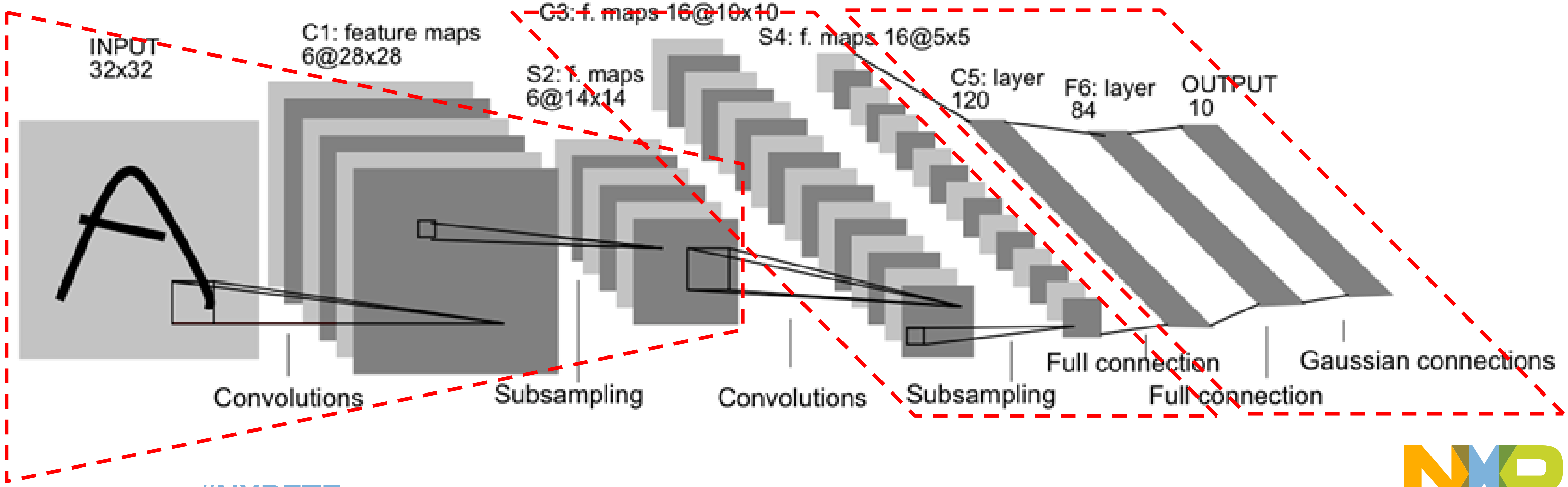
Pooling

- Reduces spatial size of the data and parameters to reduce the computational effort
- Commonly: Down sample by 2, taking the max value from a total of 4 elements
- Other pooling filters can be used but the max pooling is the one providing the best performance results



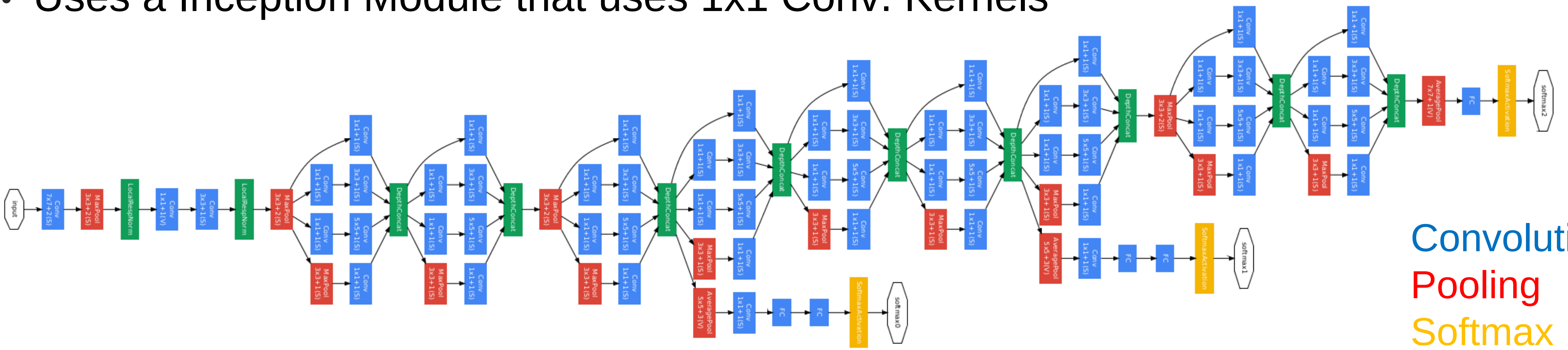
LeNet

- Developed in the 90's
- First successful convolutional neural network
- It was used to detect digits for zip codes on letters



GoogLeNet

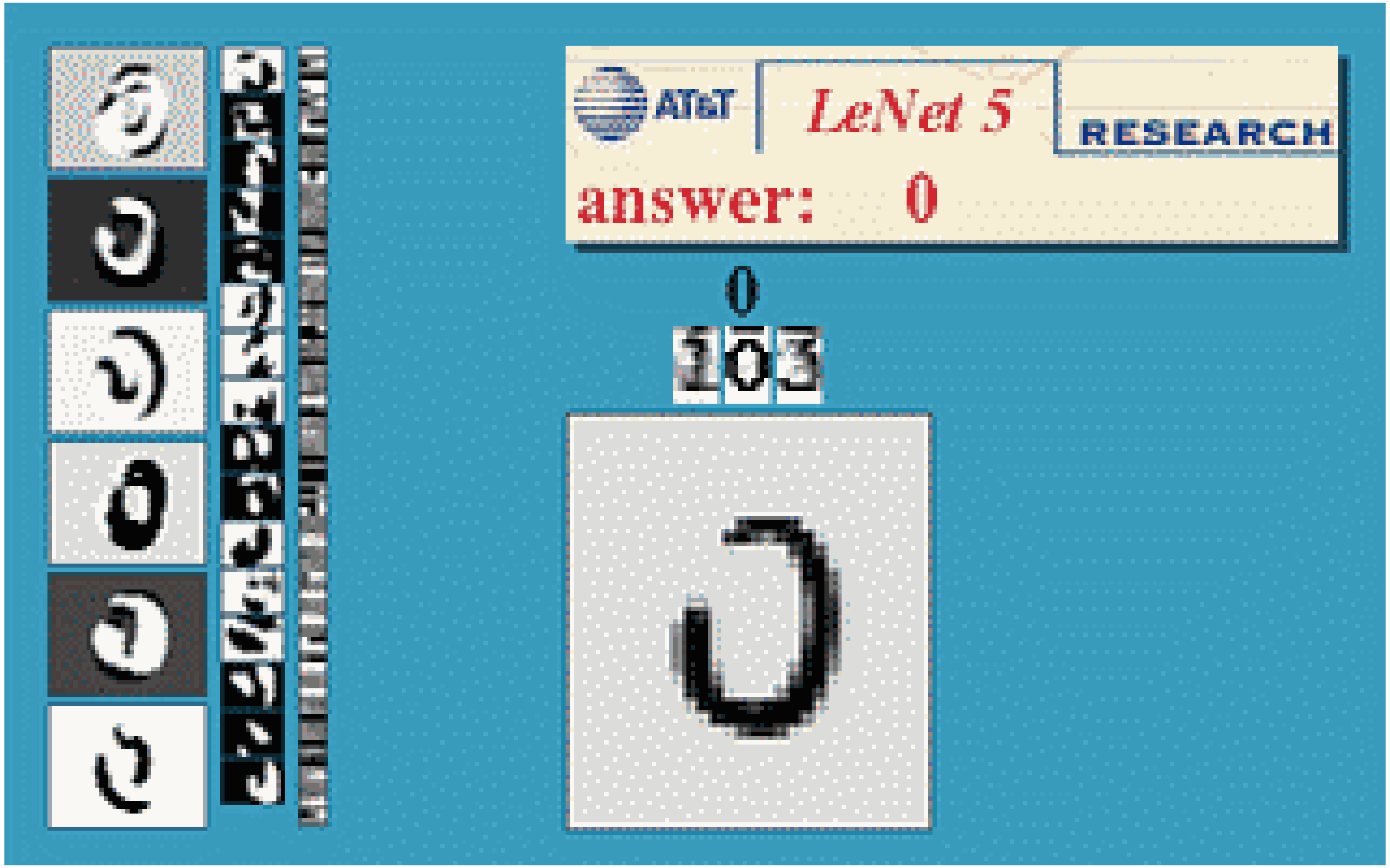
- It was the winner from the ILSVRC 2004
- The number of parameters is considerable smaller than other contemporary networks: 4M (AlexNet 60M, VGGNet has 140M)
- Uses average pooling instead of Fully Connected layers
- Uses a Inception Module that uses 1x1 Conv. Kernels



Convolution
Pooling
Softmax
Other

HANDS-ON: CNN HAND WRITTEN DIGITS DETECTION

LeNet – Hand-written digits



Convolutional Neural Networks – Hand-written digits

- On you Host:
  - Open and edit main.cpp to perform Neural Network on your camera input
    - gedit ~/s32v234/demos/N1791_CNN/src/main.cpp
  - You need to perform the following edits:
    1. Manipulate the input so the numbers are more distinguishable
    2. Compare results to previous hands-on (fully connected NN)
    3. Compare processing speed with previous hands-on (fully connected NN)

  - Some Hints:
    - Compile your project and rename your file to a different name. That way you will be able to easily run one program or the other.

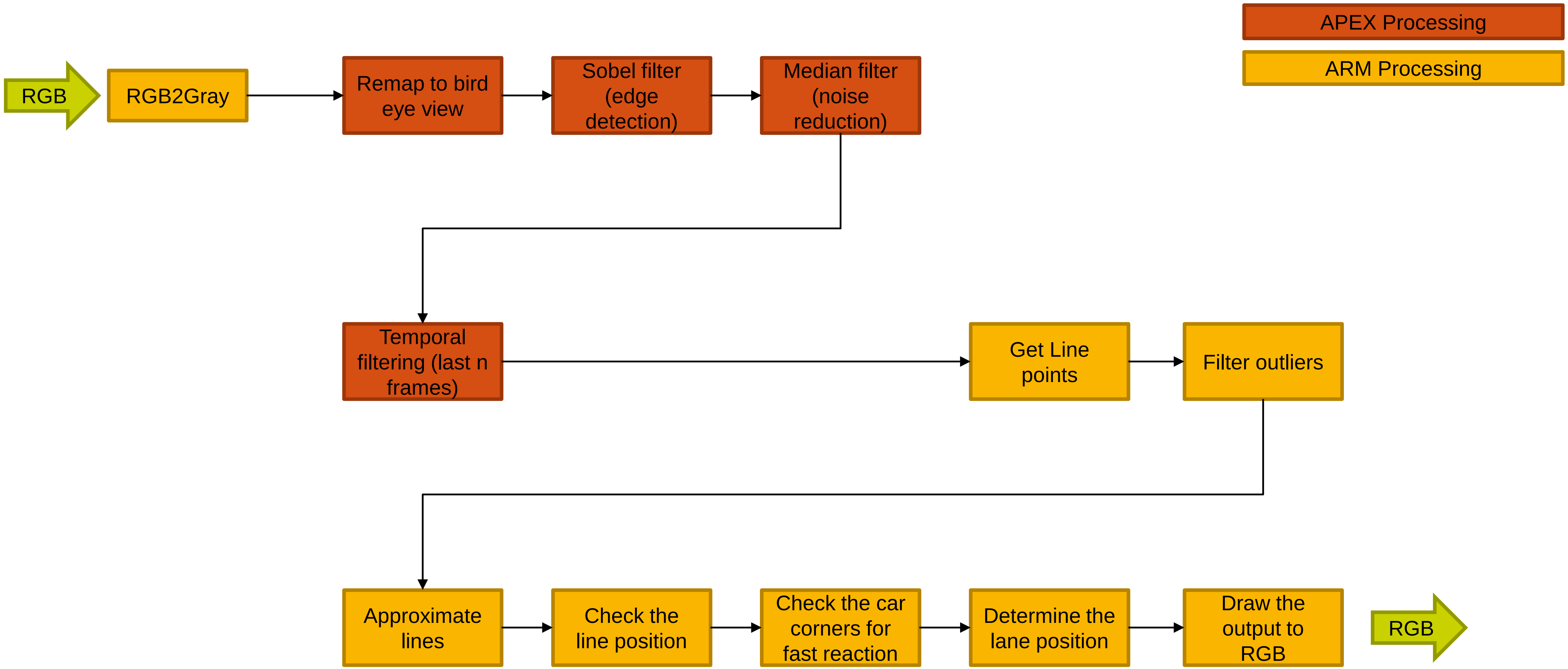
Convolutional Neural Networks: Step 2

- On Your Host:
  - Build your application:
    - cd ~/s32v234_sdk/demos/N1791_CNN/build-v234ce-gnu-linux-d/
    - ./build.sh
  - Copy the generated binary to your Network File System:
    - cp isp_csi_dcu.elf ~/rootfs/s32v234/demos/

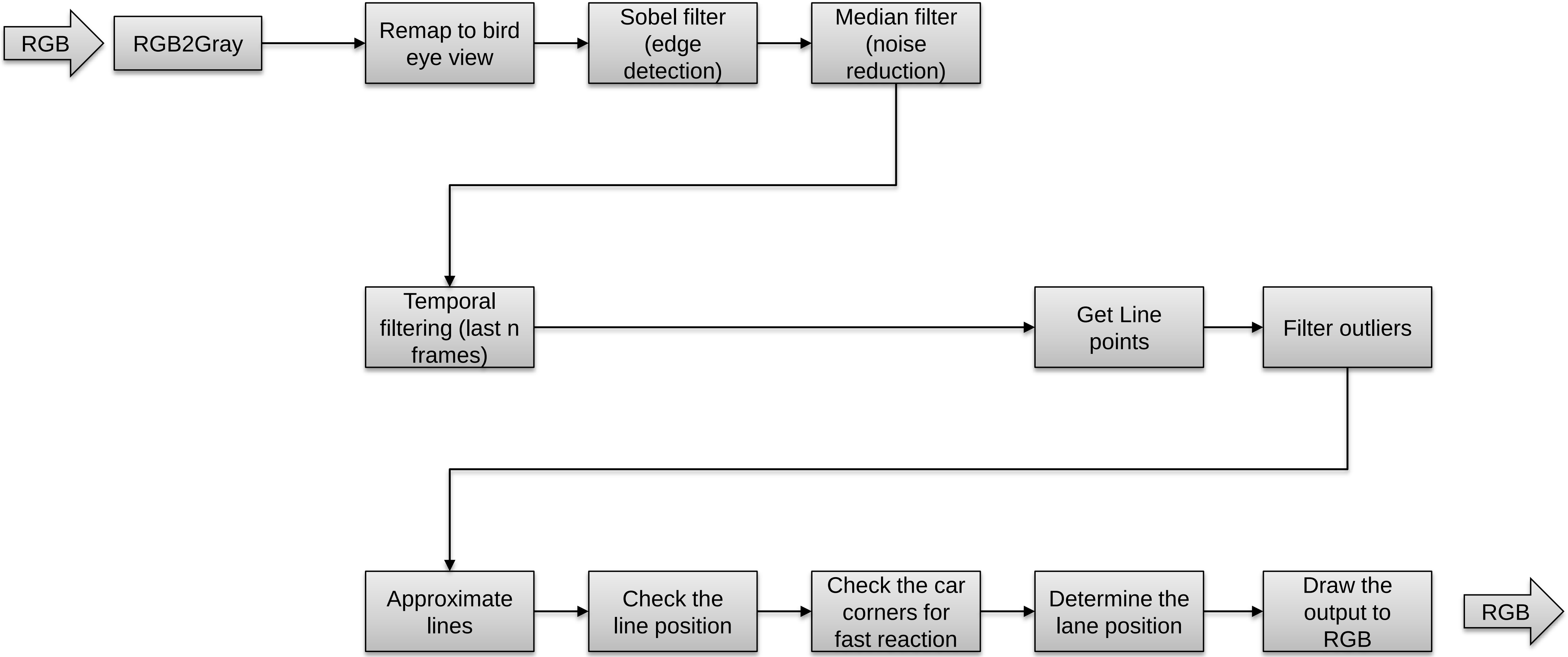
- On Your Target (Serial Console):
  - Stop the previous demo and run the generated binary:
    - ../s32v234/demos/isp_csi_dcu.elf
  - Observe the results on the screen

PRACTICAL CASE: LANE DETECTION ALGORITHM

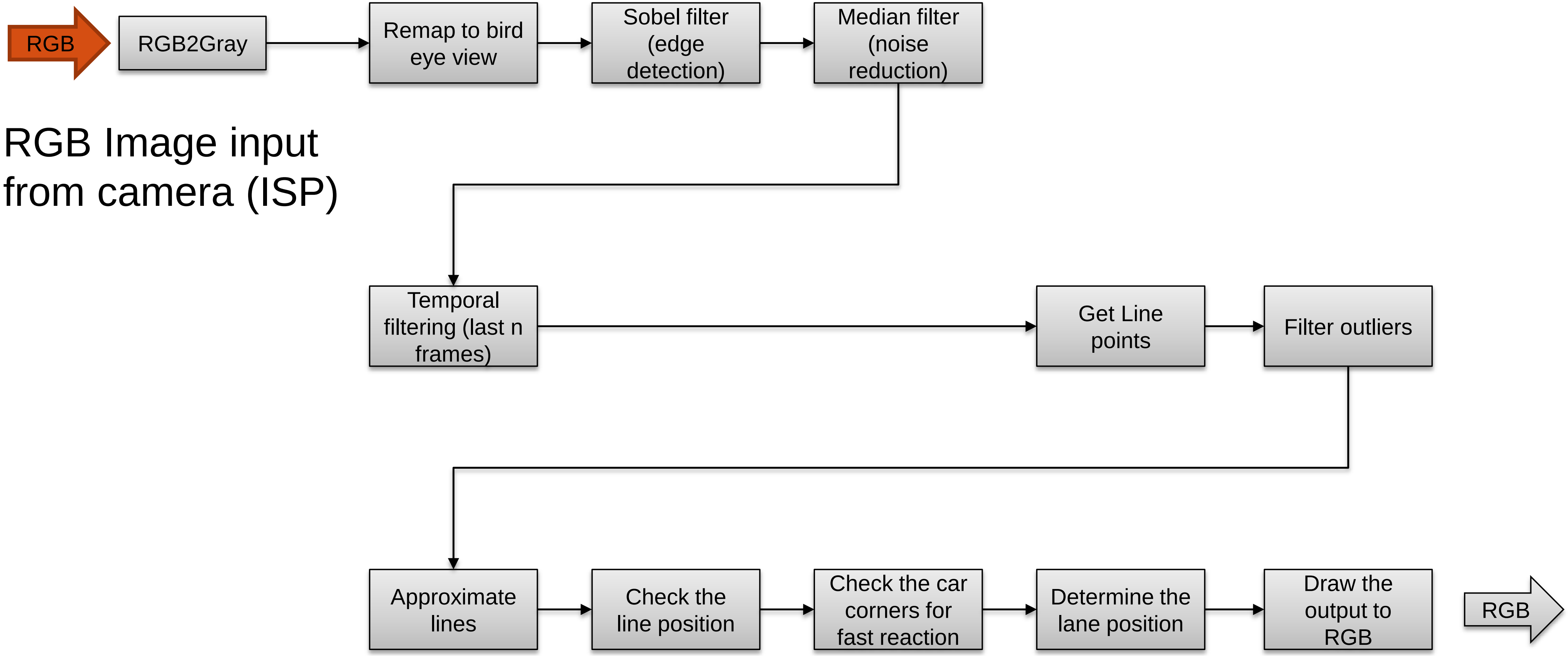
1.2 LDW Algorithm overview



1.2 LDW Algorithm overview



1.2 LDW Algorithm overview



1.2 LDW Algorithm overview

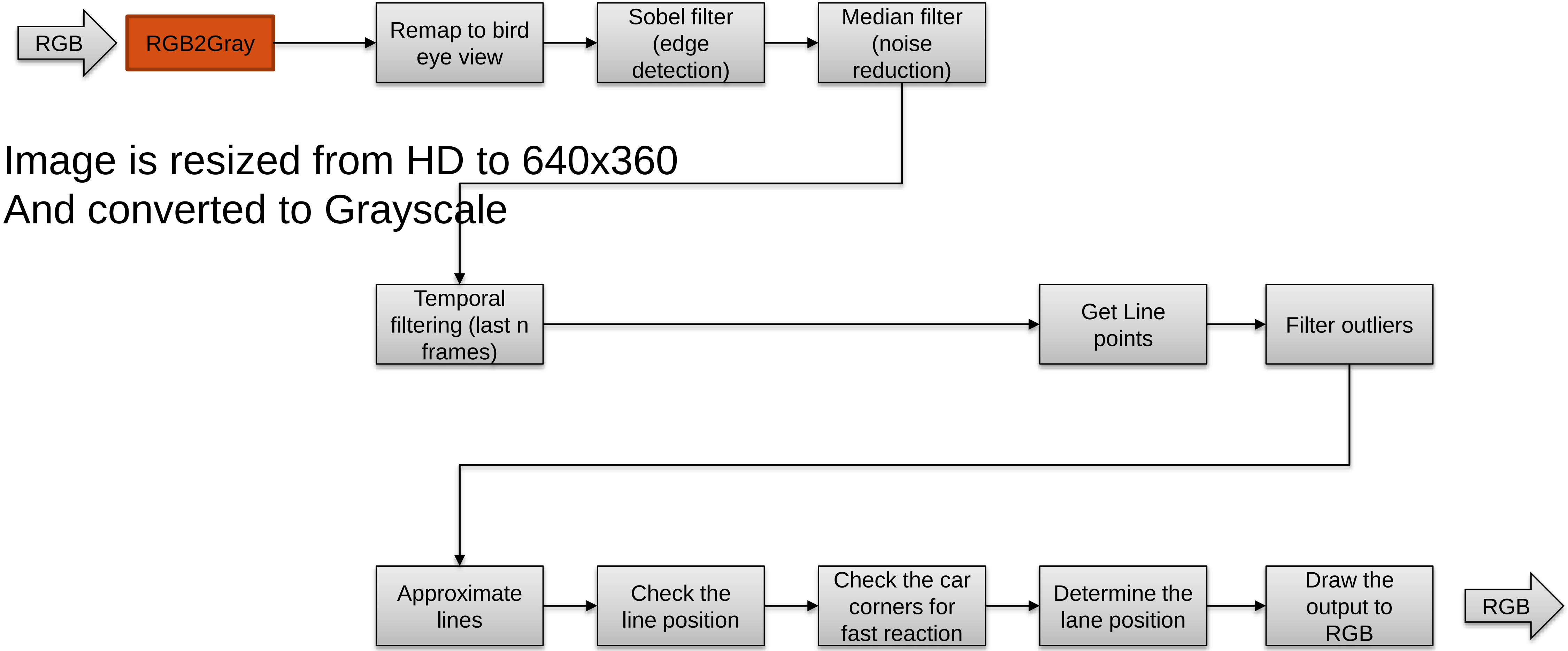
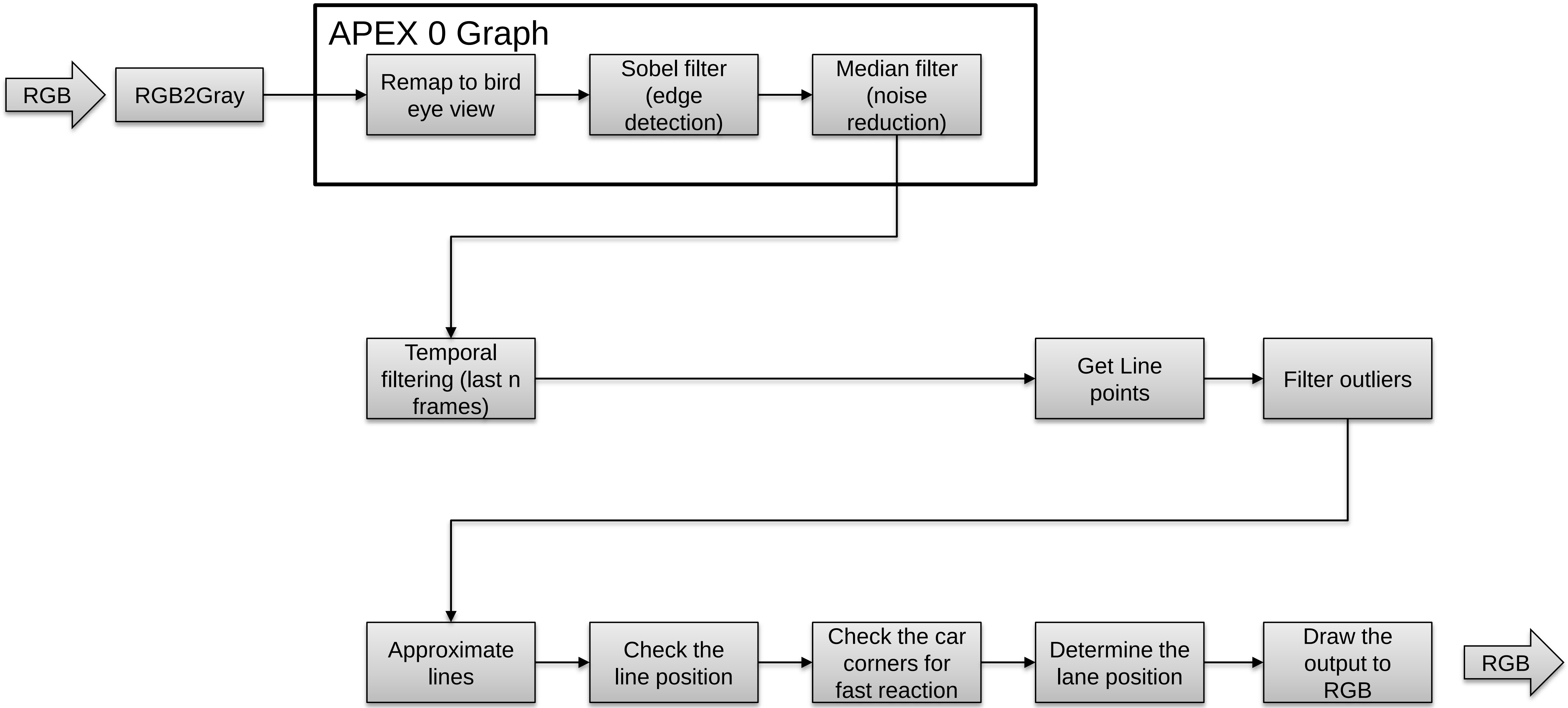
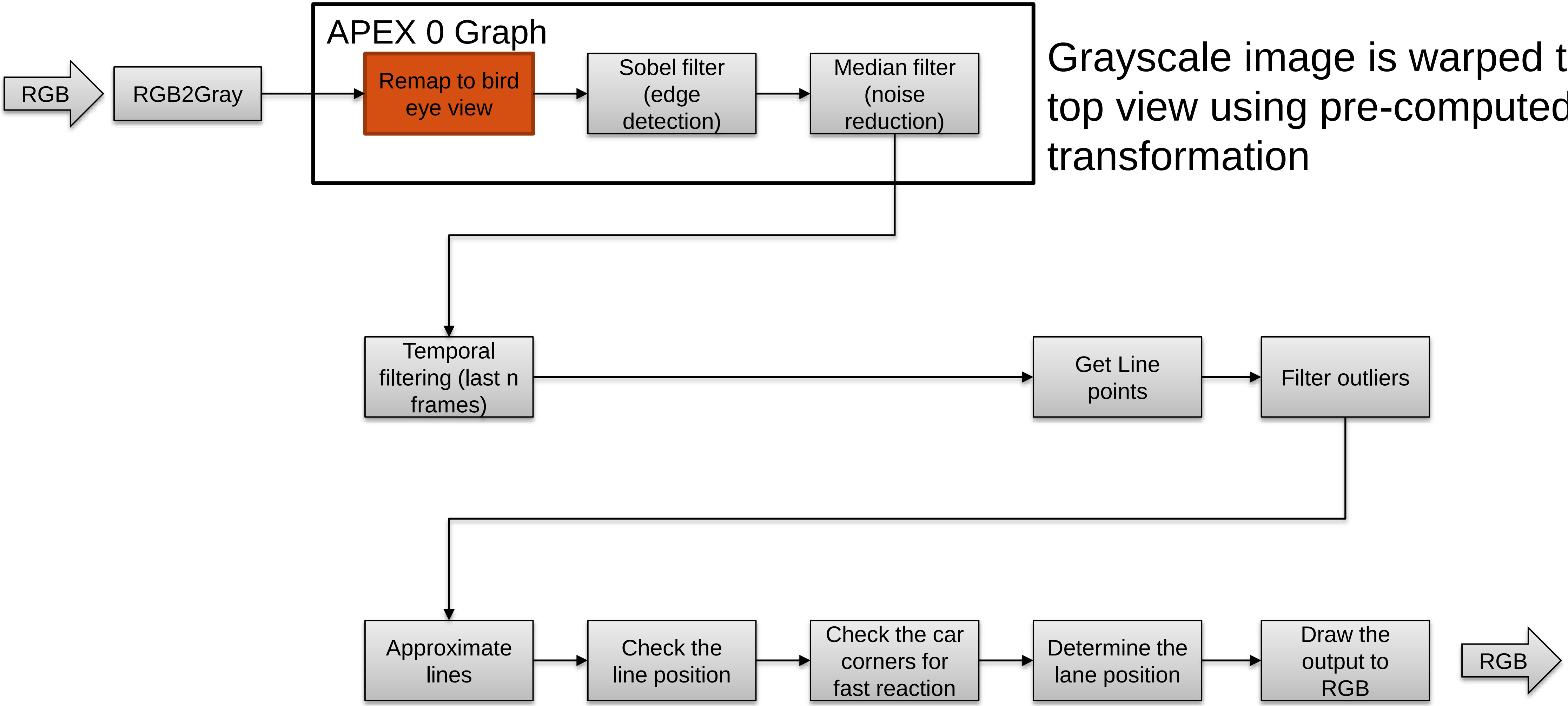


Image is resized from HD to 640x360
And converted to Grayscale

1.2 LDW Algorithm overview

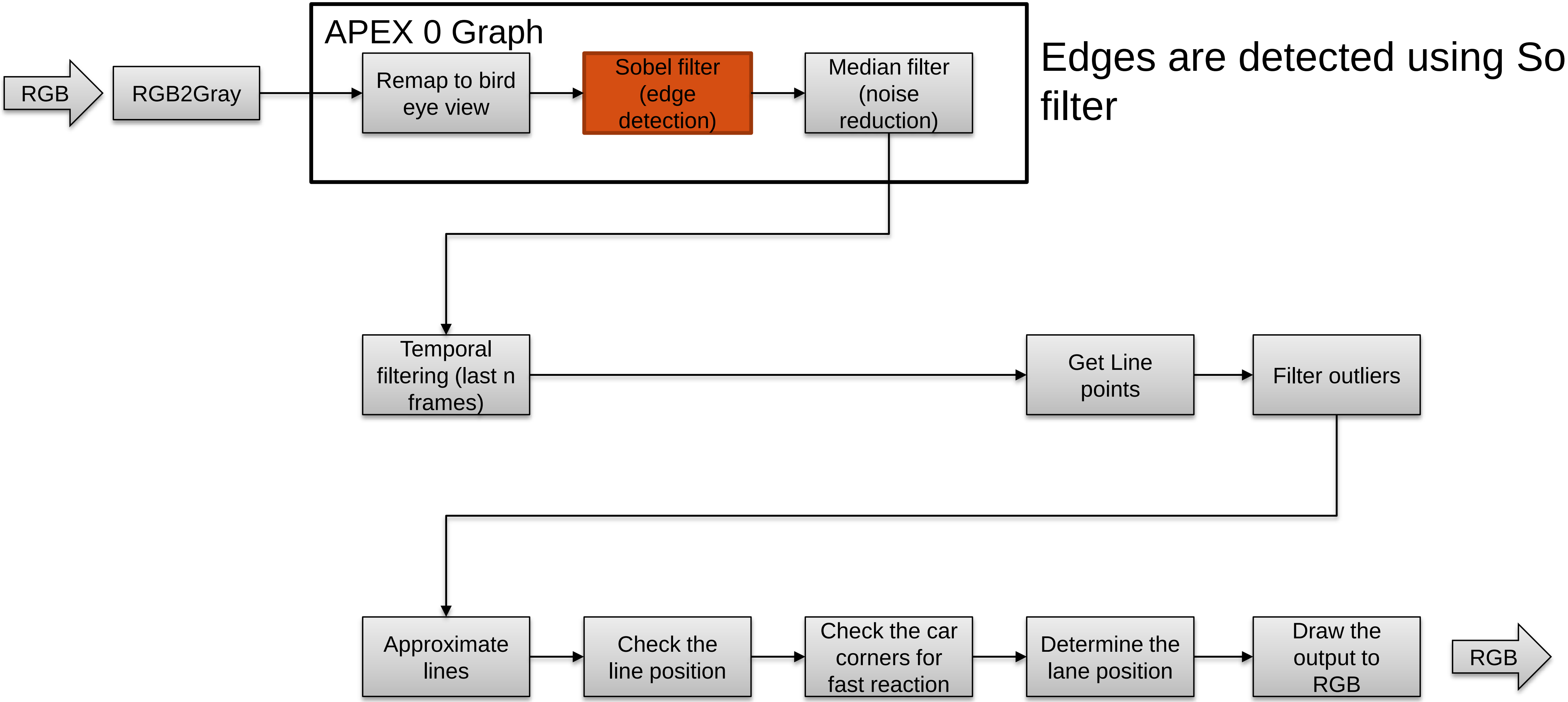


1.2 LDW Algorithm overview



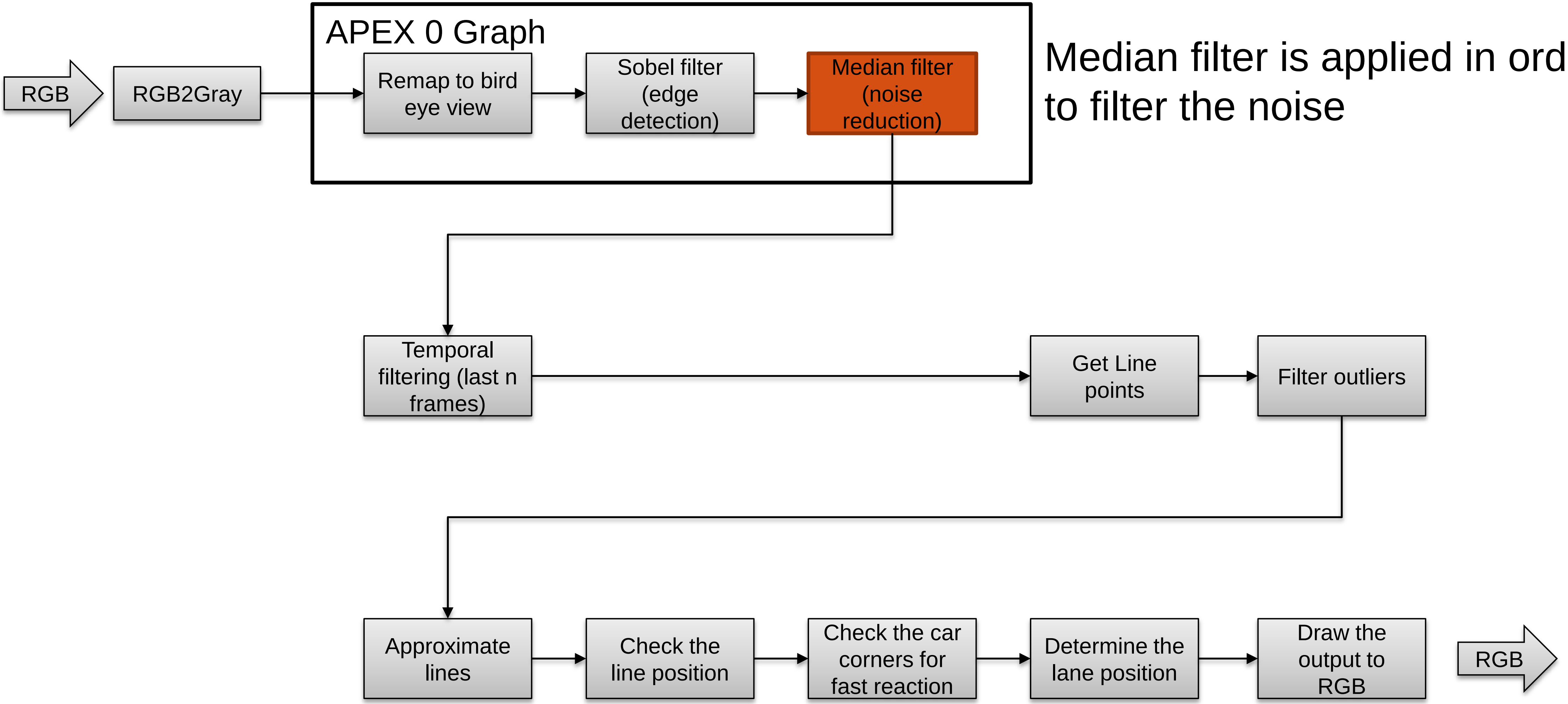
Grayscale image is warped to parallel top view using pre-computed transformation

1.2 LDW Algorithm overview



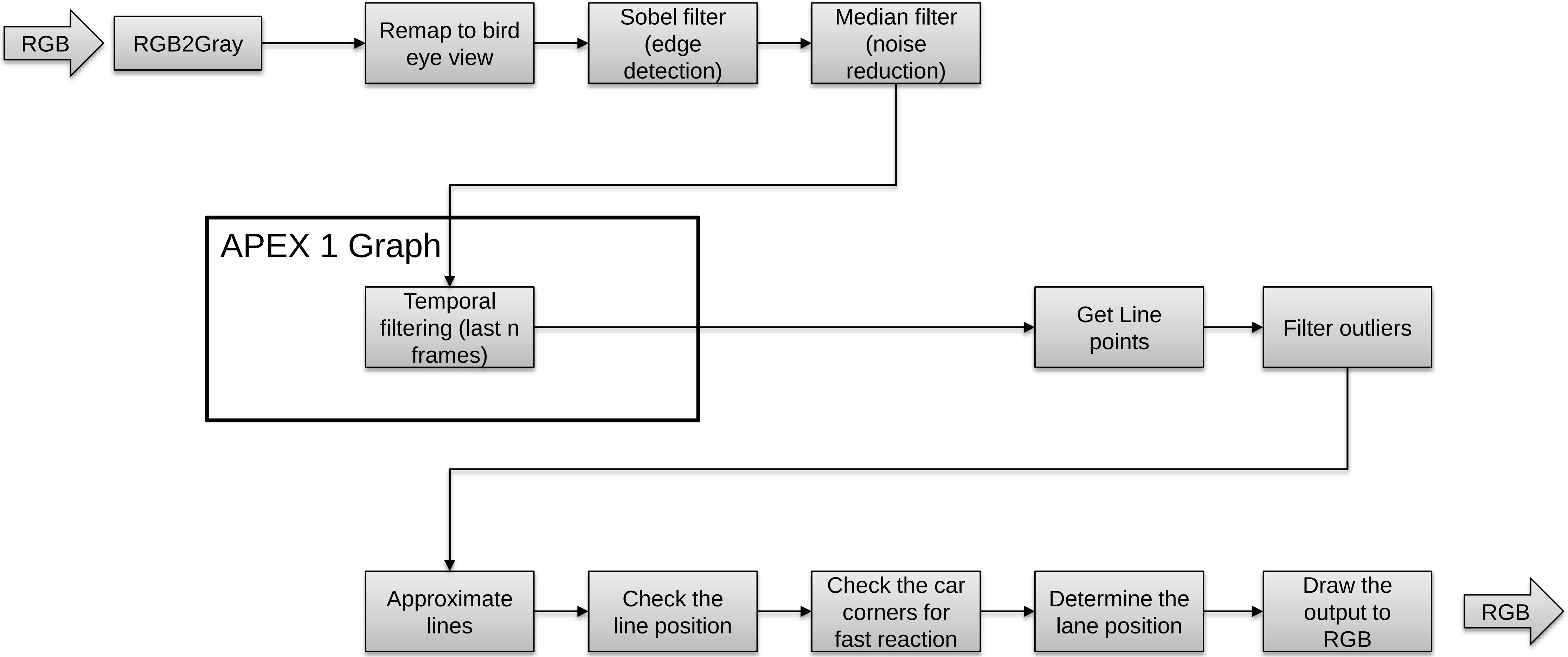
Edges are detected using Sobel filter

1.2 LDW Algorithm overview

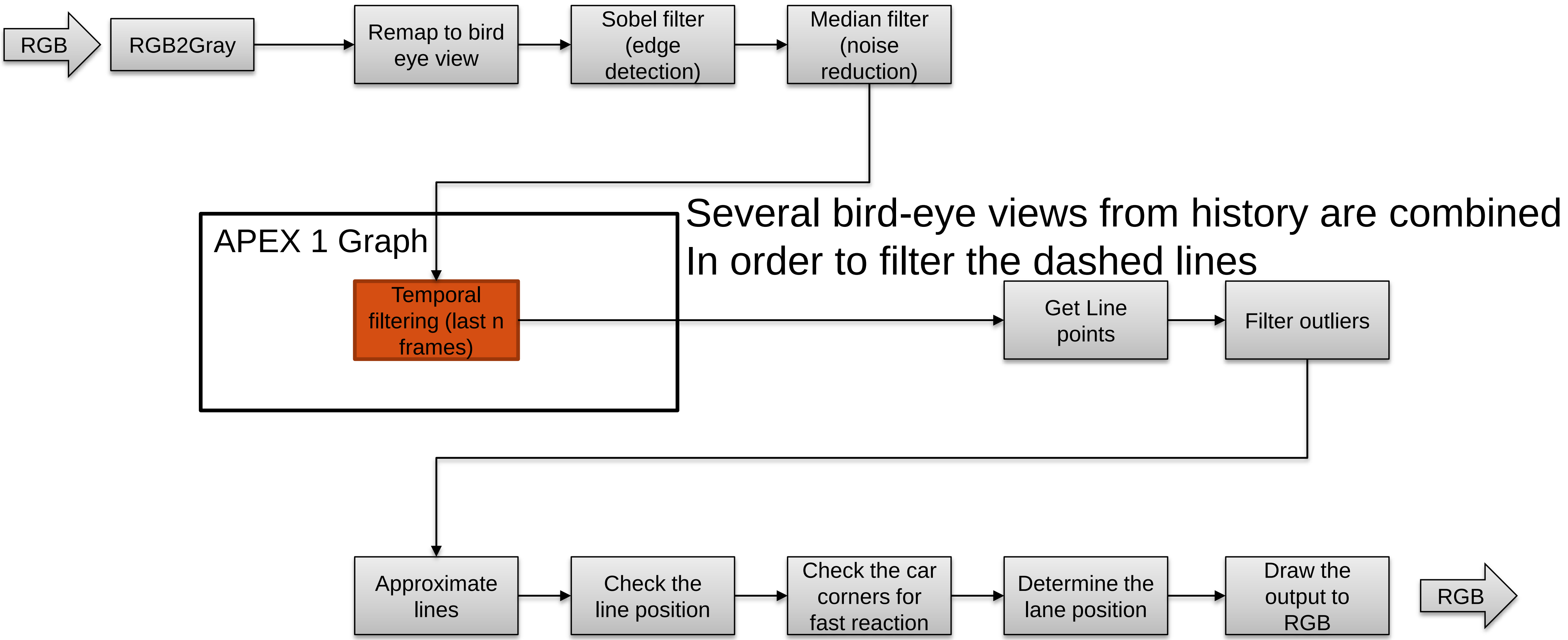


Median filter is applied in order to filter the noise

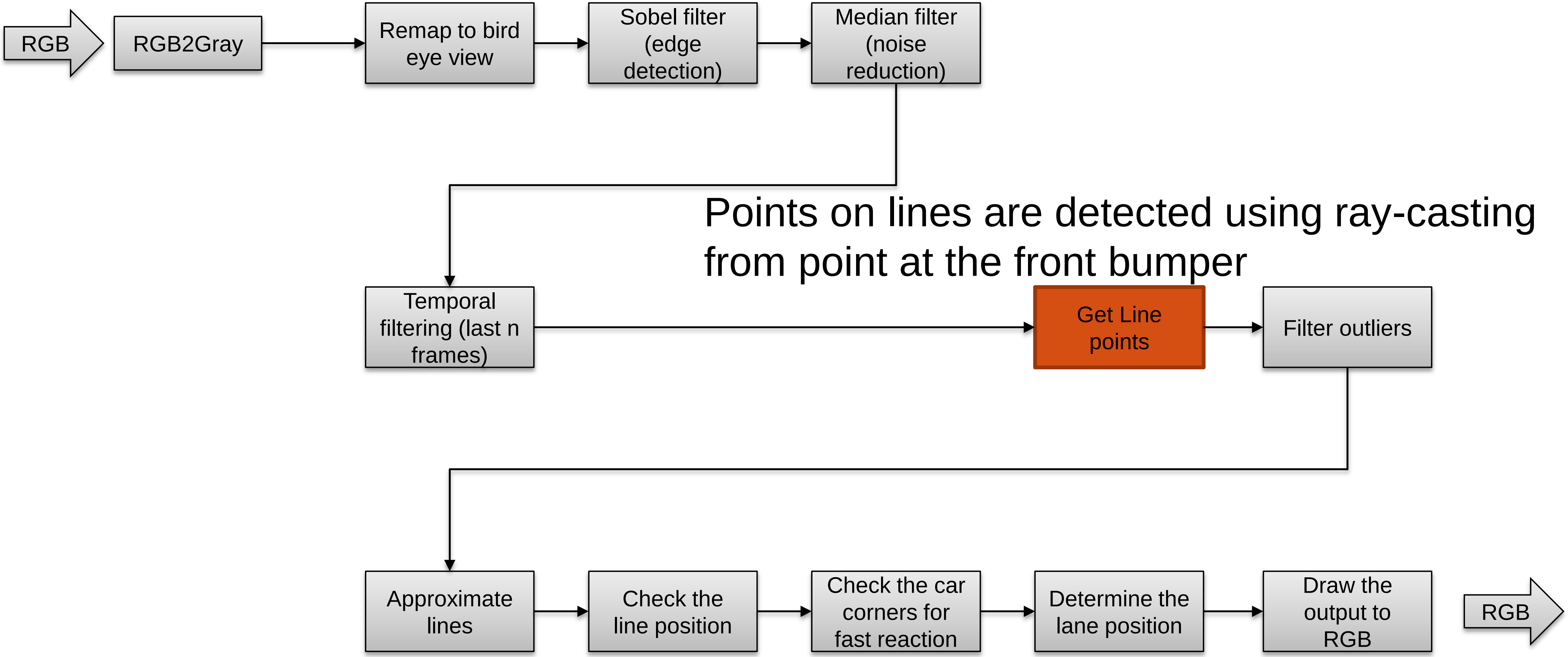
1.2 LDW Algorithm overview



1.2 LDW Algorithm overview

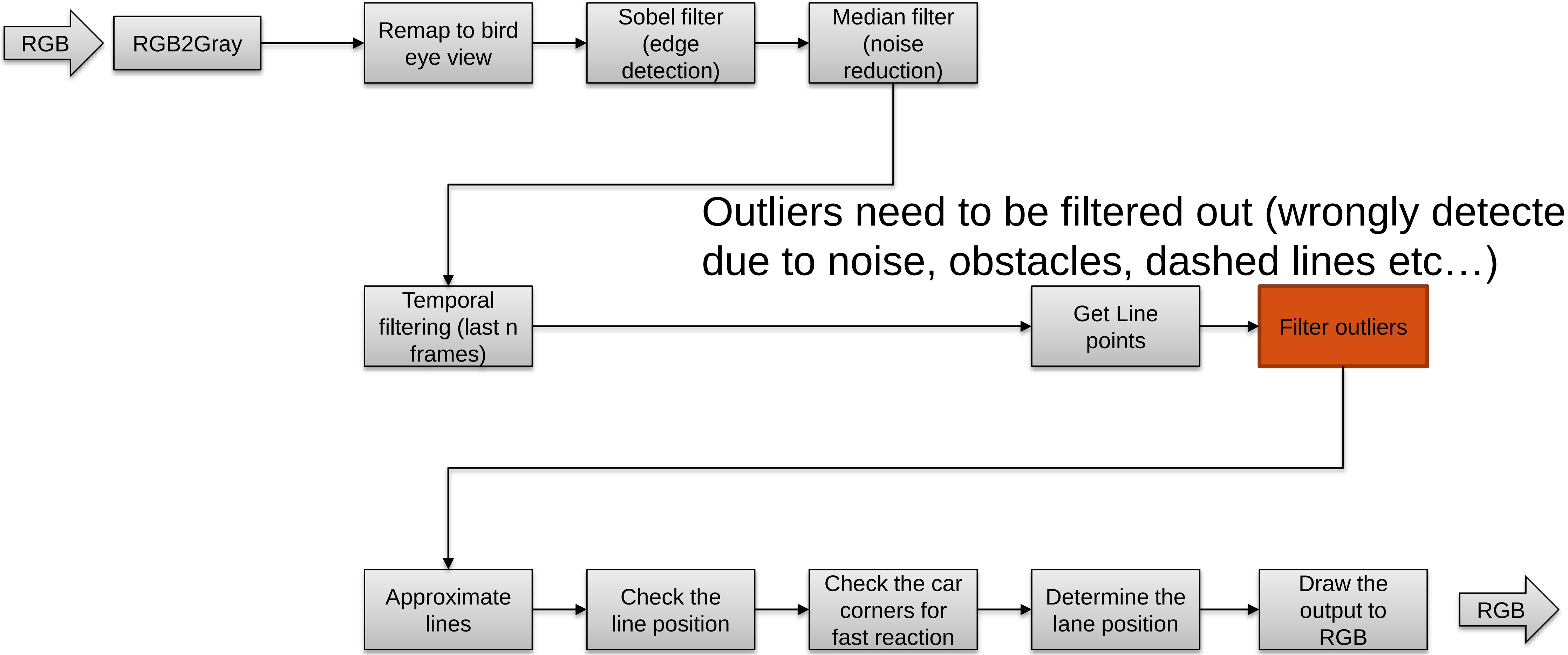


1.2 LDW Algorithm overview



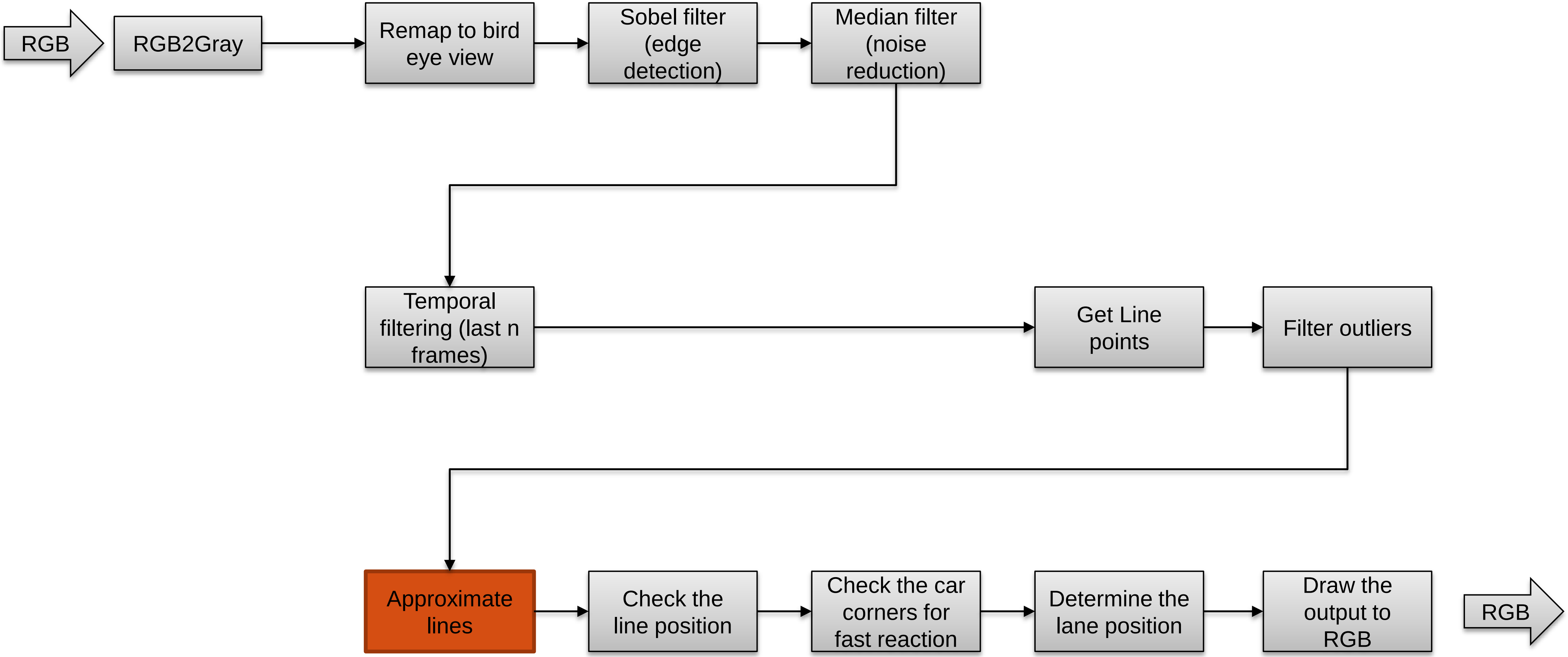
Points on lines are detected using ray-casting from point at the front bumper

1.2 LDW Algorithm overview



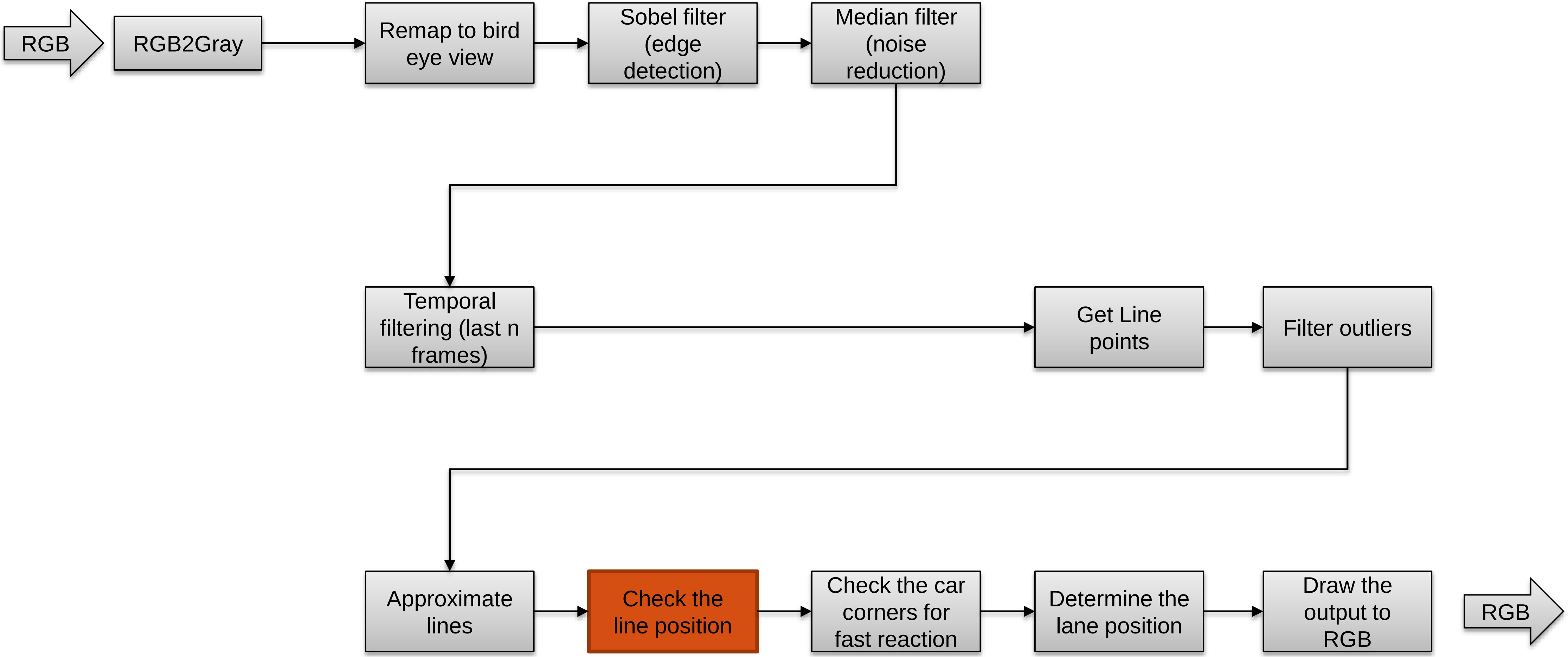
Outliers need to be filtered out (wrongly detected points due to noise, obstacles, dashed lines etc…)

1.2 LDW Algorithm overview



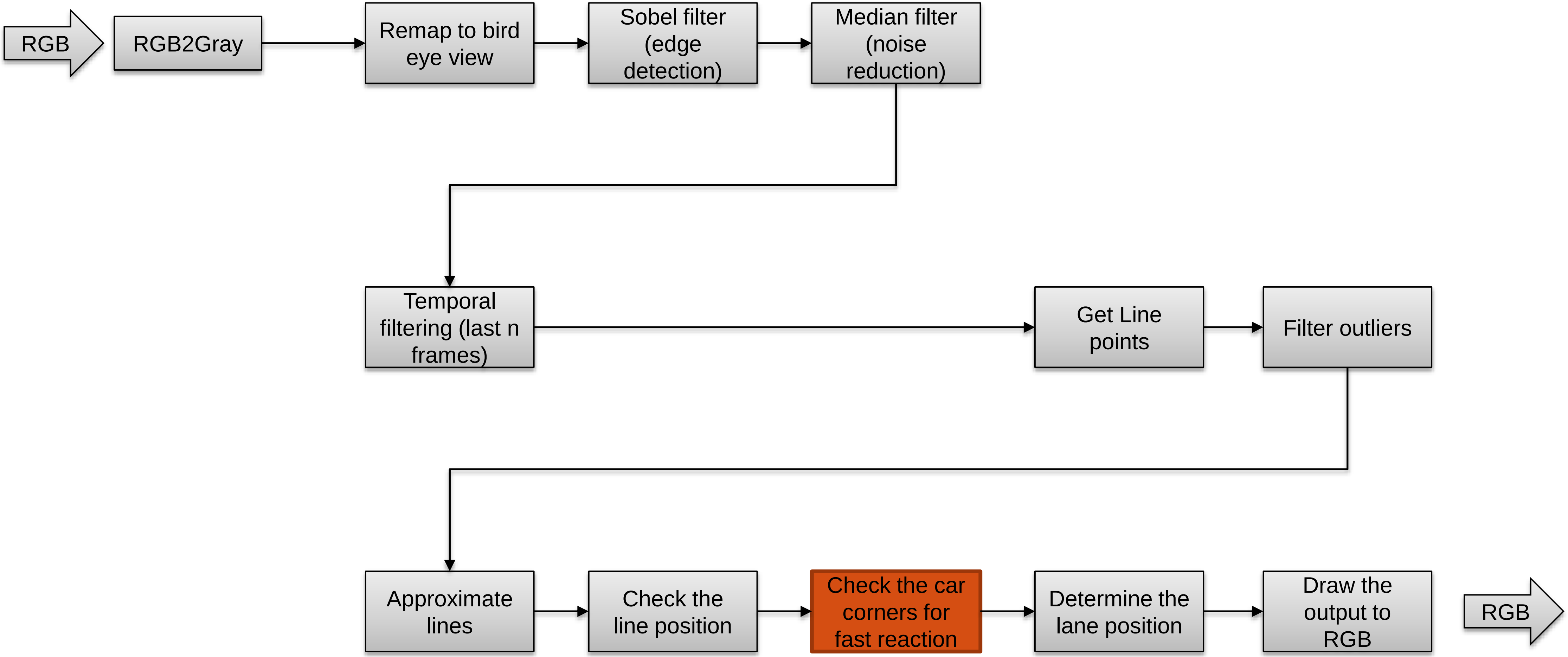
Least squares line approximation through detected and filtered points

1.2 LDW Algorithm overview



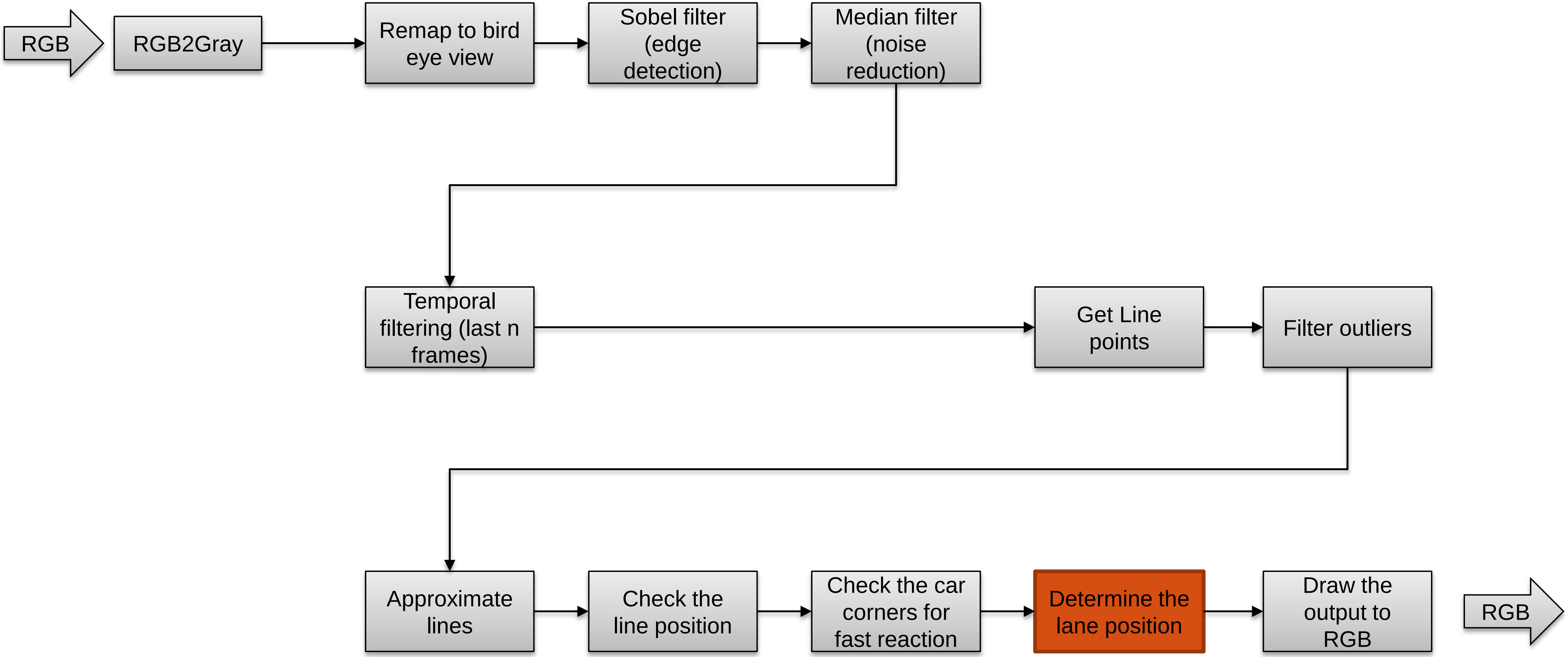
Based on computed lines, compute the position in the lane.

1.2 LDW Algorithm overview



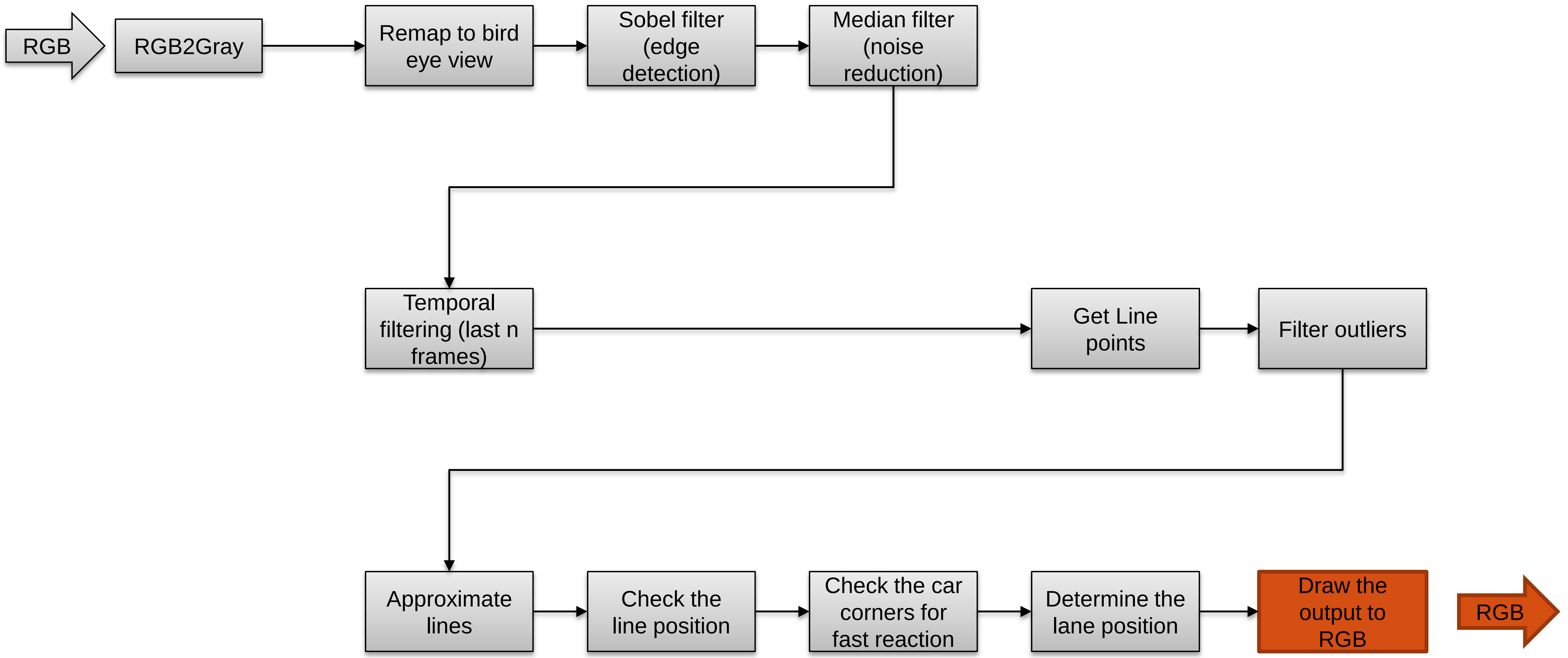
Double-check the detection by checking the white color in front of wheels

1.2 LDW Algorithm overview



Based on computed lines and corner check, decide if the warning should be raised

1.2 LDW Algorithm overview



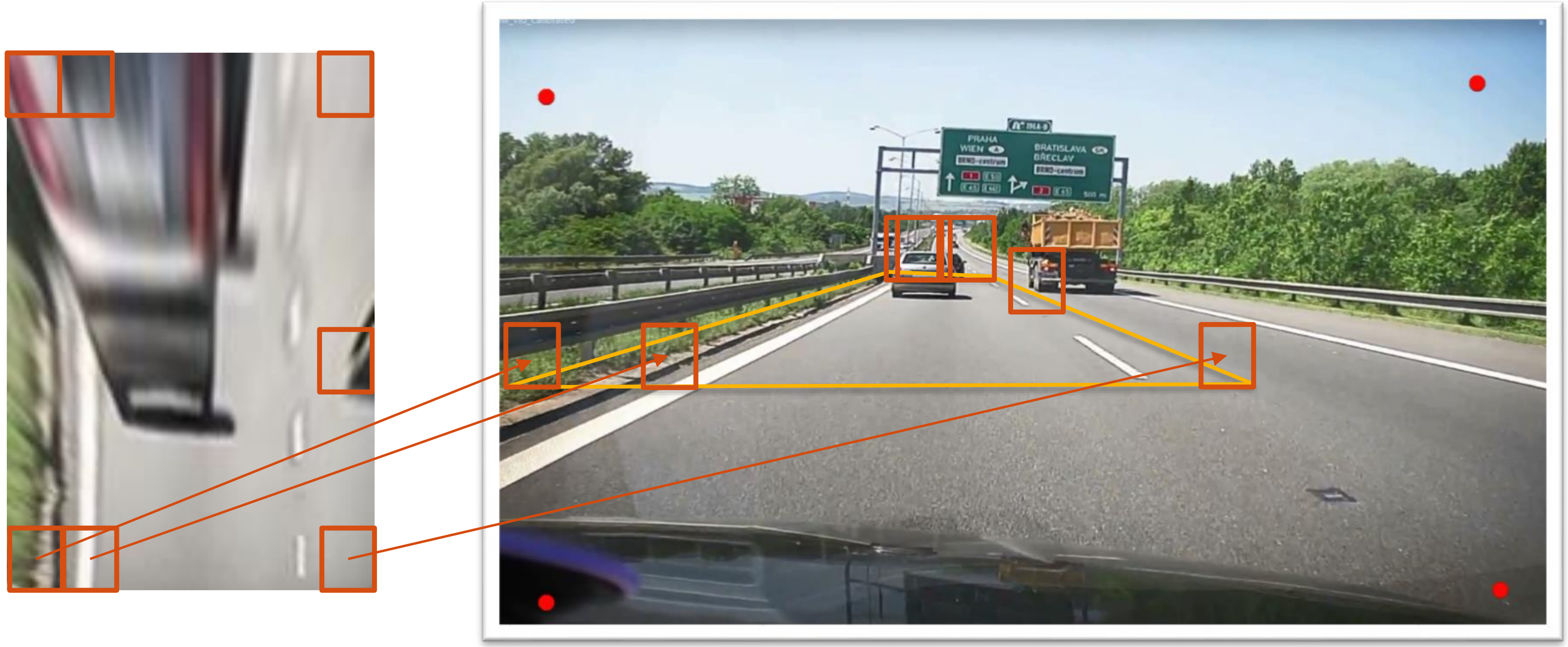
Draw detected lines to the original RGB input and display it

1.3 Bird eye warp – APEX warp

- Every destination image point value needs to be computed from original one.
- On APEX – Indirect access – three tables need to be pre-computed:
  - **m_block**
  - m_local
  - m_delta

Size:
CONFIG_BIRD_SIZE_W / CONFIG_BIRD_CHUNK_SIZE_W
X
CONFIG_BIRD_SIZE_H / CONFIG_BIRD_CHUNK_SIZE_H

Index in original image for every destination chunk

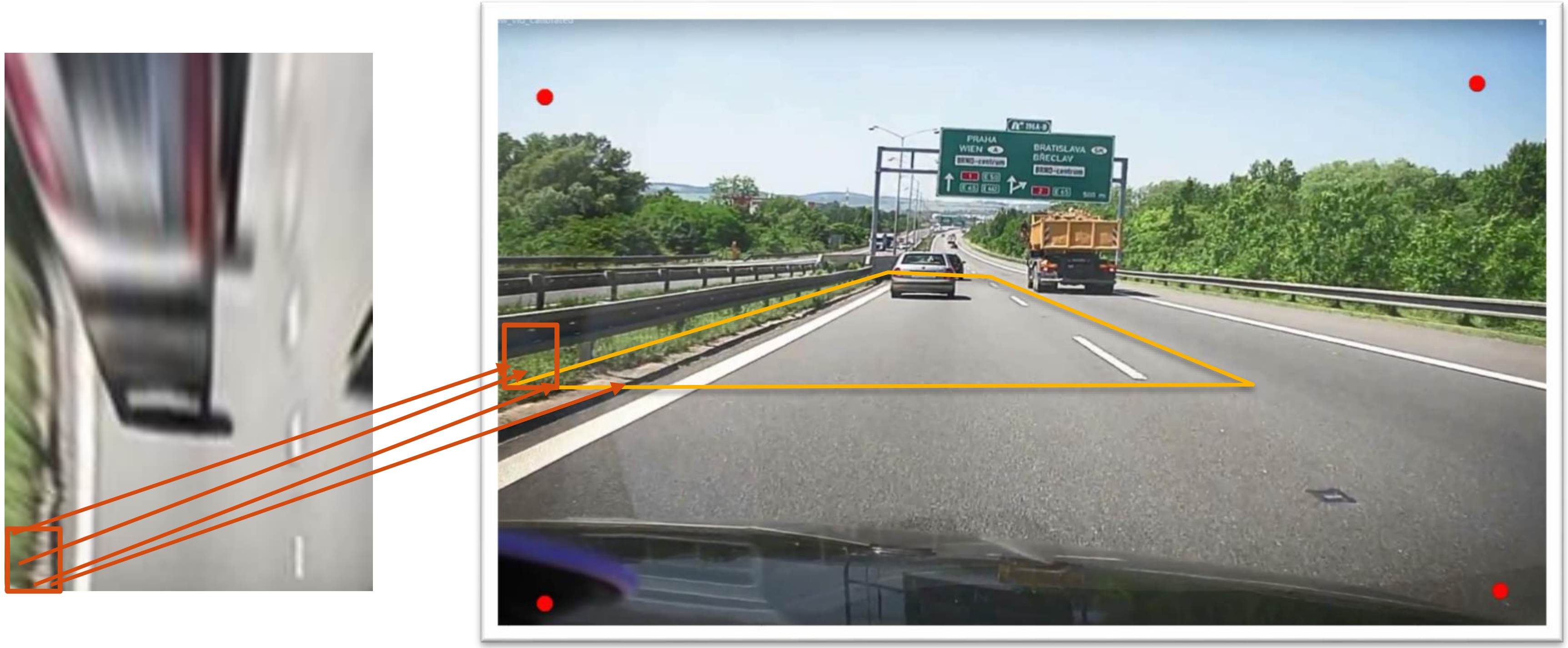


1.3 Bird eye warp – APEX warp

- Every destination image point value needs to be computed from original one.
- On APEX – Indirect access – three tables need to be pre-computed:
  - m_block
  - **m_local**
  - m_delta

Size:
BIRD_EYE_WIDTH X BIRD_EYE_HEIGHT

Index in original image for every destination pixel – local chunk coordinates



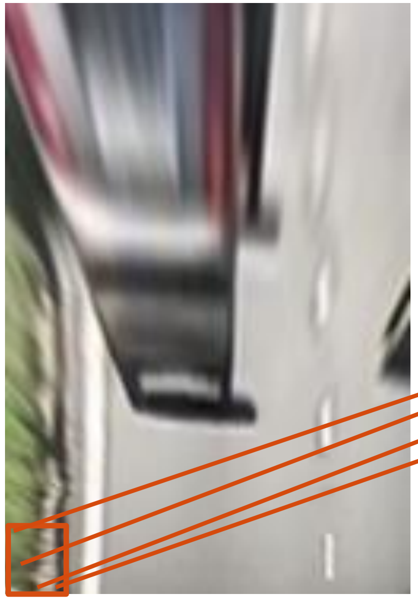
1.3 Bird eye warp – APEX warp

- Every destination image point value needs to be computed from original one.
- On APEX – Indirect access – three tables need to be pre-computed:
  - m_block
  - m_local
  - **m_delta**

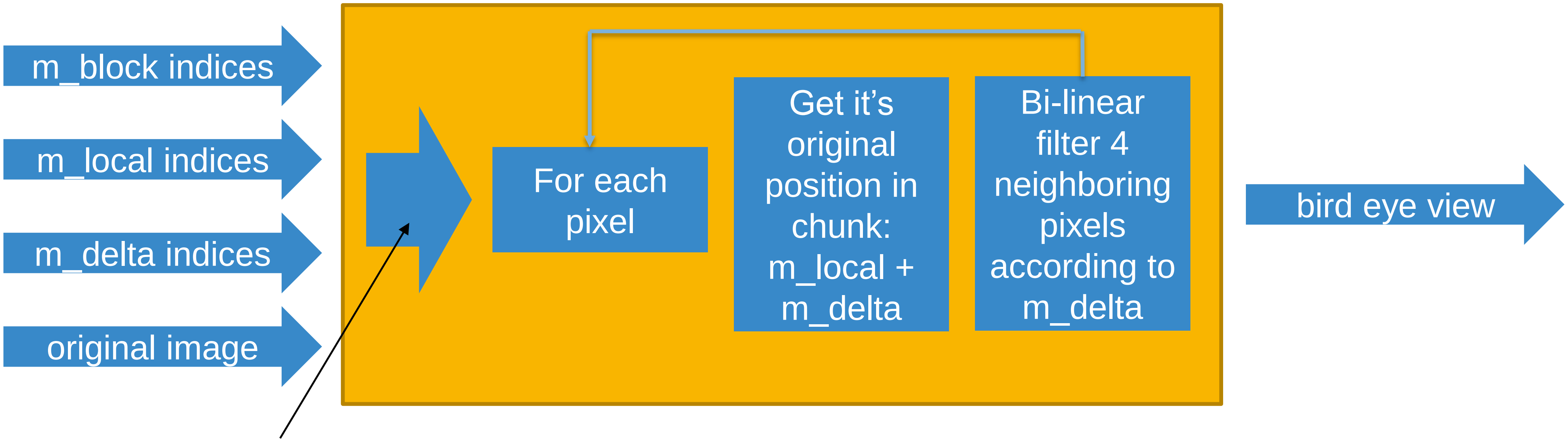
Size:
BIRD_EYE_WIDTH X BIRD_EYE_HEIGHT x 2

Index value after decimal point in original image for every destination pixel –local chunk coordinates

Used for bi-linear map



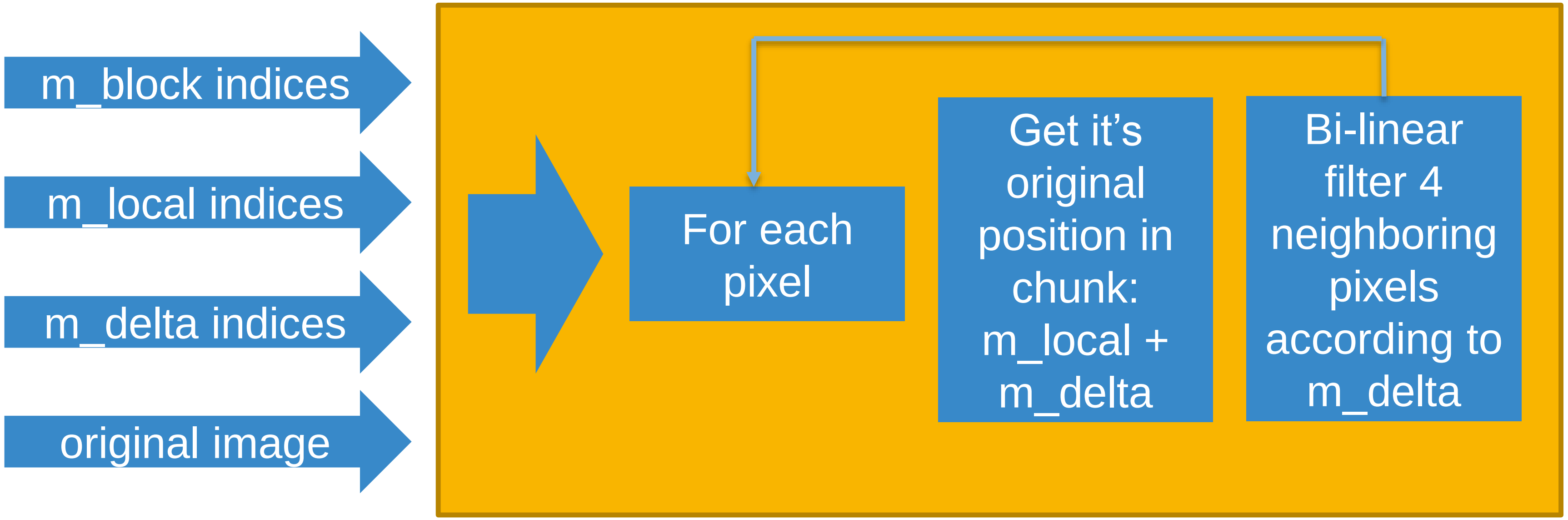
1.3 Bird eye warp – APEX simple kernel graph

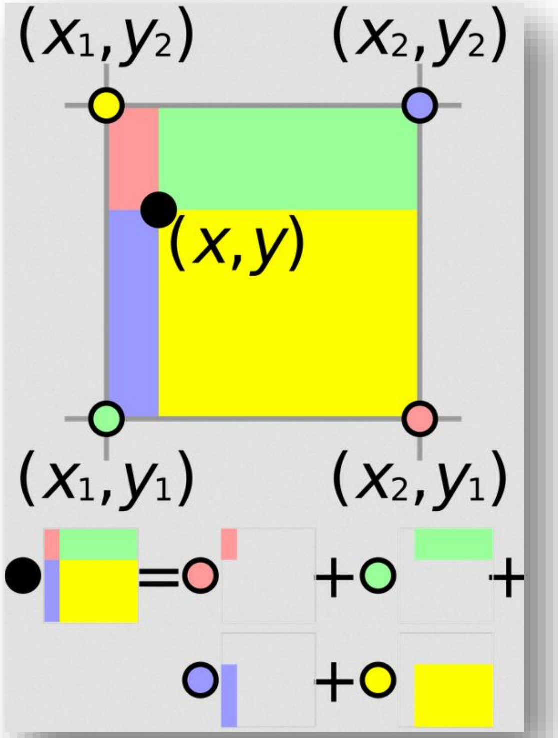


Input image is now same size as bird eye view
– fetched chunks according to m_block indices

1.3 Bird eye warp – APEX simple kernel graph

Bilinear filtering computes the final value as weighted average of neighboring pixels based on m_delta

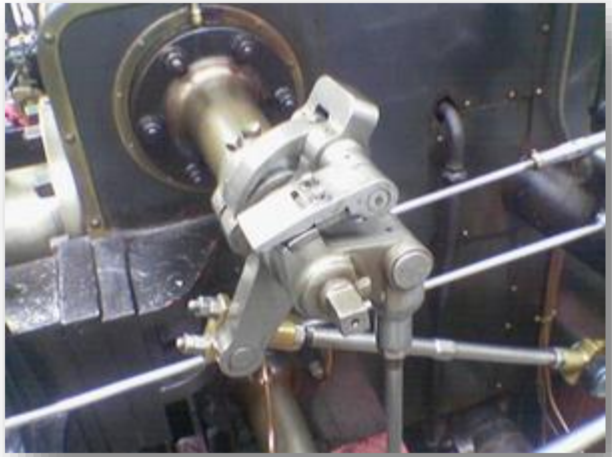


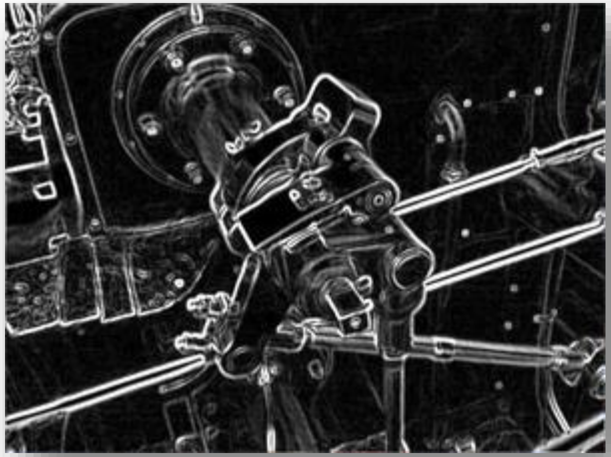


1.4 Sobel filter edge detector

- edge detection algorithm
- **discrete differentiation operator**, computing an approximation of the gradient of the image intensity function
- approximation of the derivatives – two kernels convolved with the image

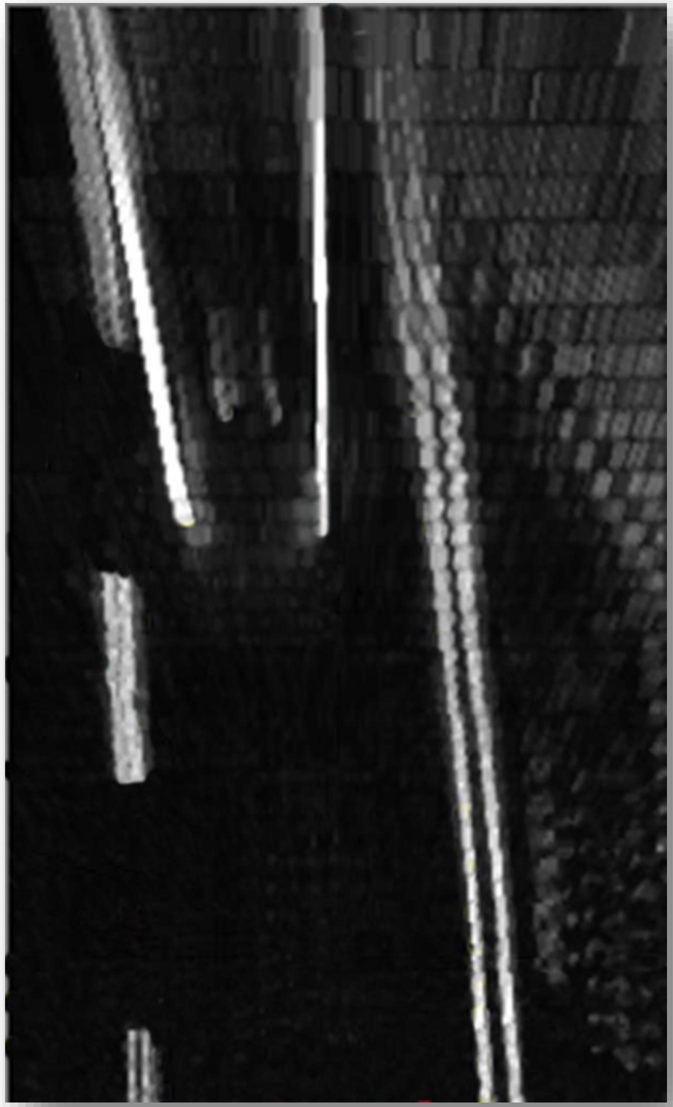
$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$





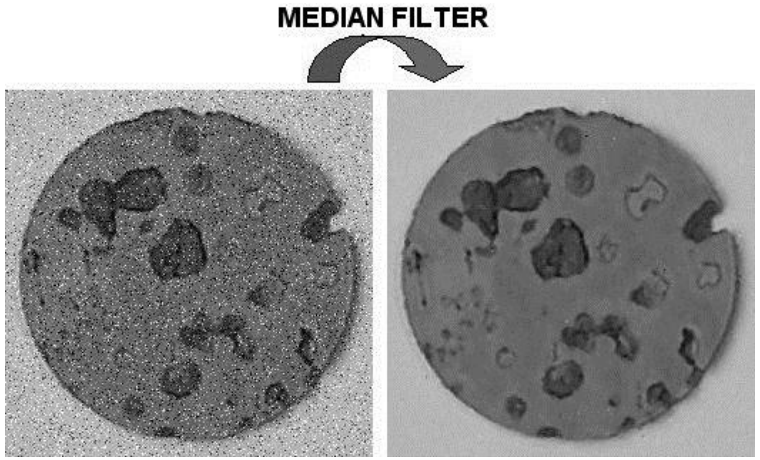
1.4 Sobel filter edge detector

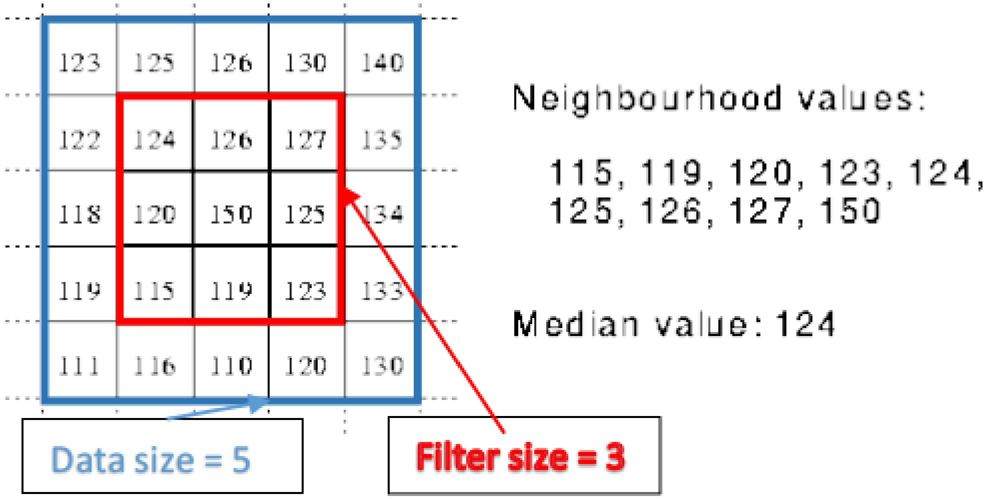
- Second step of the APEX graph
- Computes edges on bird-eye view



1.5 Median filter

- Noise cancelling mechanism
- Non-linear filter
- Computes value of the pixel based
  On median value of it’s neighborhood
- Removes impulsive noise





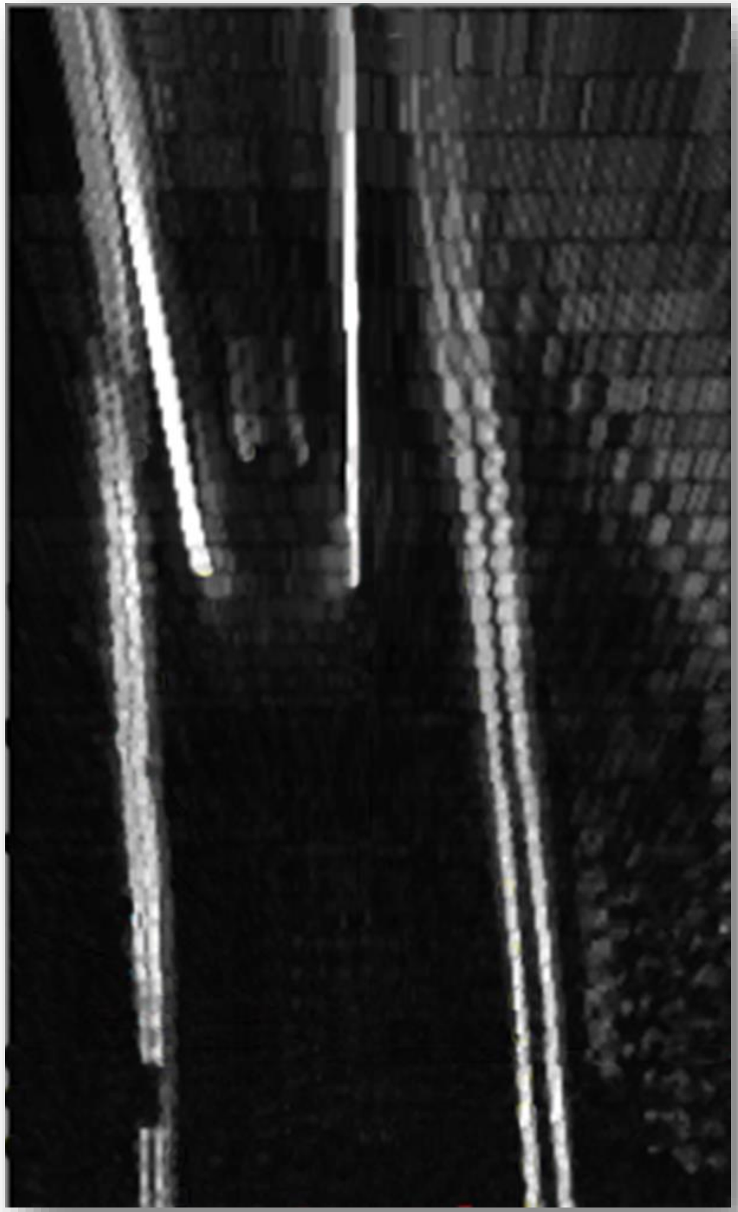
1.6 Temporal filter

- The dashed lines too separated
- Need for making them more straight
  - Addition of the images from history

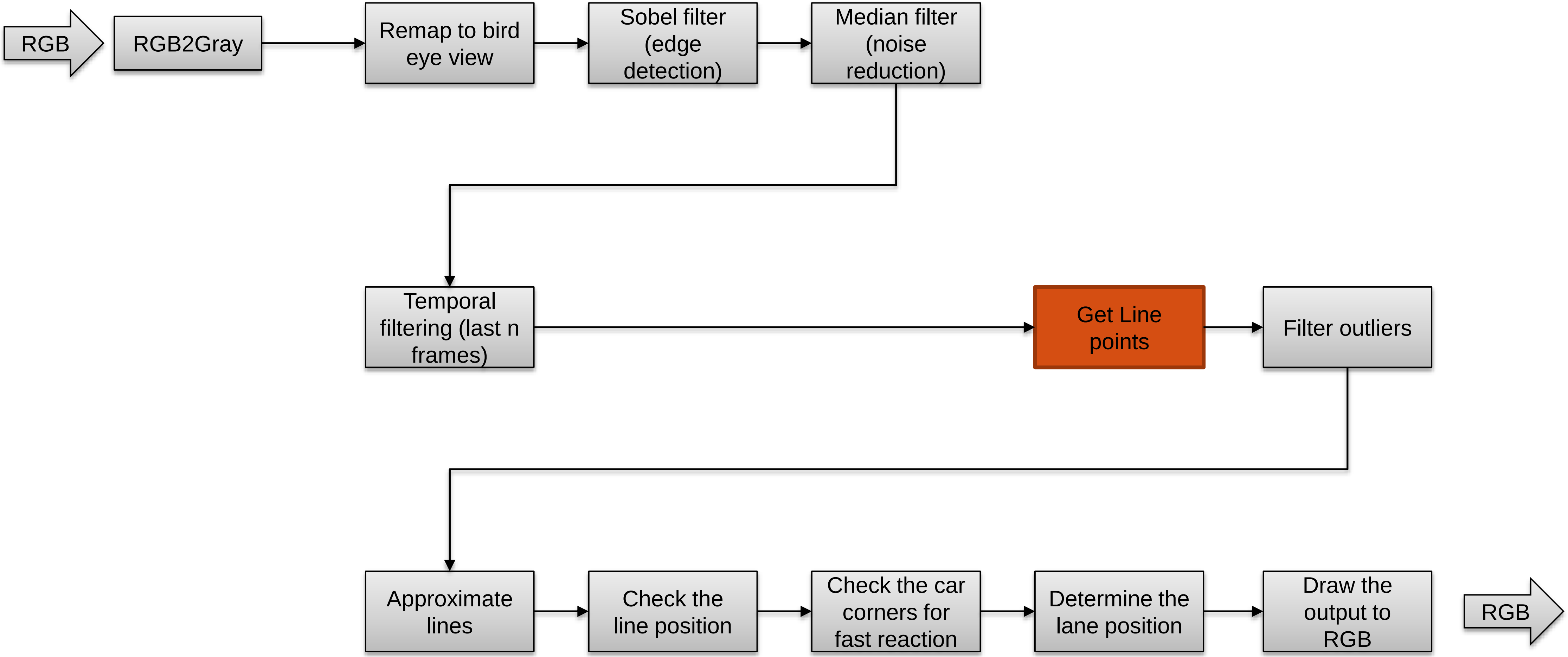
- Implemented on APEX

$$I_{filt}(t) = Max(I(t-n),...,I(t))$$

- 14 last images combined each frame.

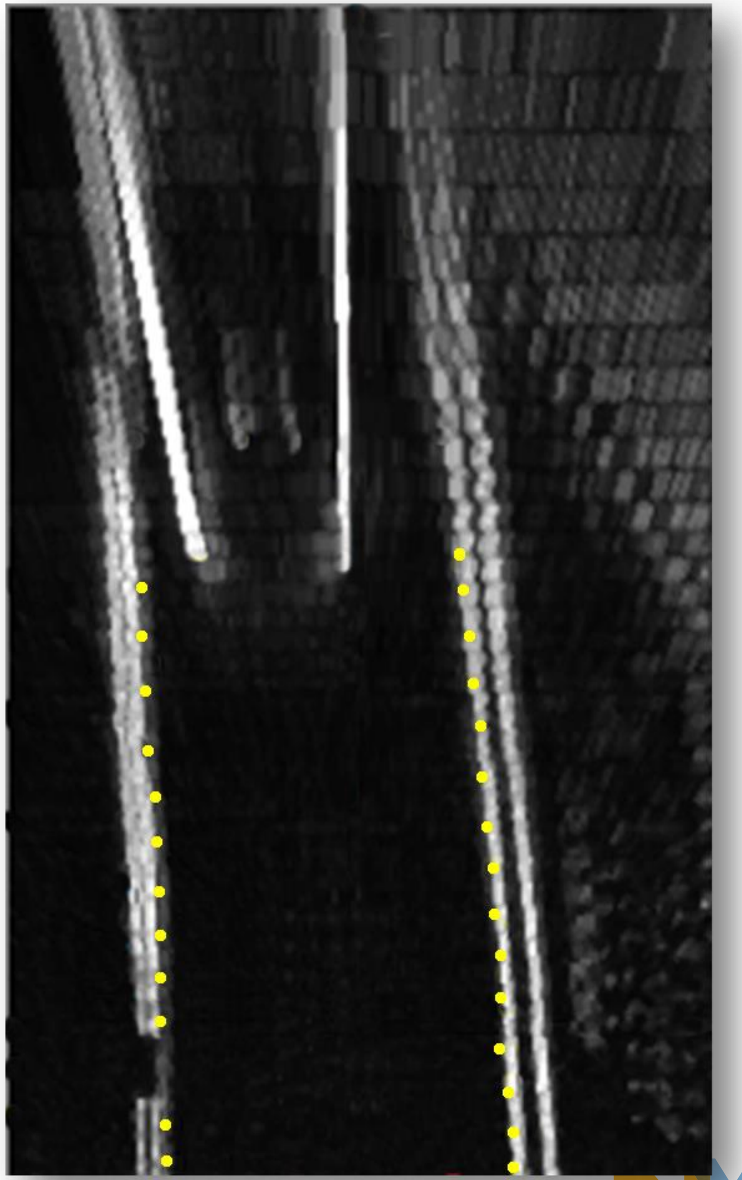


1.7 Lane points detection



1.7 Lane points detection

- Inside points in the lane must be detected
- Several algorithms were investigated
  - Hough Transform – problems on too curvy lanes
  - RANSAC – too slow – big search space
  - Homegrown algorithm inspired with active contours

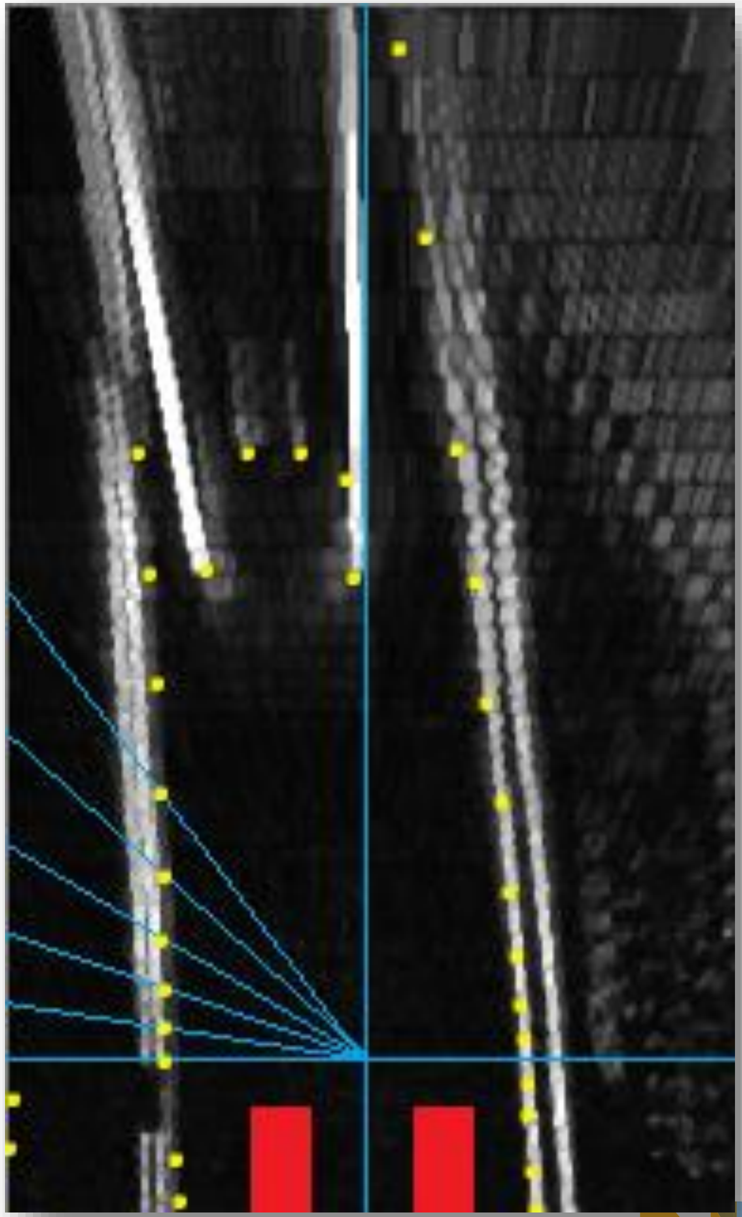


- Why do we need to search anywhere else than inside the lane?

1.7 Lane points detection

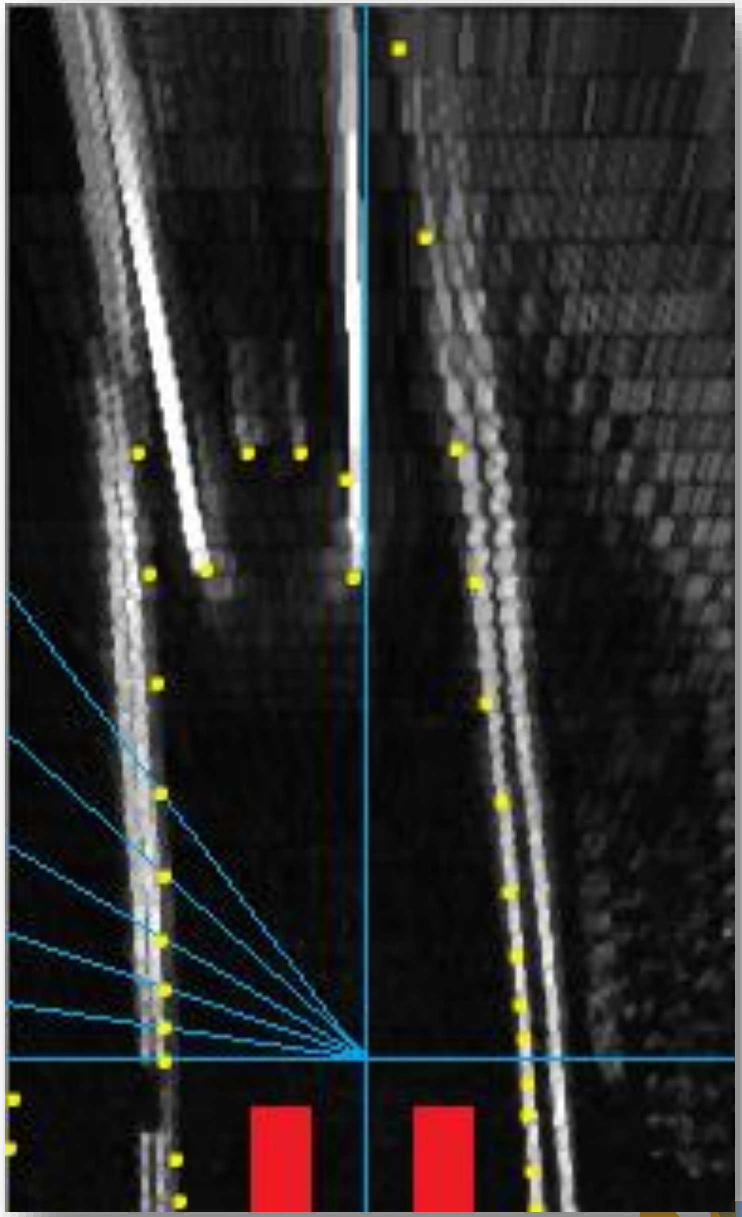
- Approach based on Ray-Casting
- There is no point of searching for points outside the lane

- Rays (precomputed lines) cast from the front bumper – uses OpenCV line iterator
  - Pre-defined angle step
- Thresholding the white values – stops at first
- value exceeding the threshold



1.7 Lane points detection

- Big advantage – creates an ordered list of points
- All computed on ARM due to global image pass
  - Something not possible for APEX
- Disadvantage – points must be post-filtered

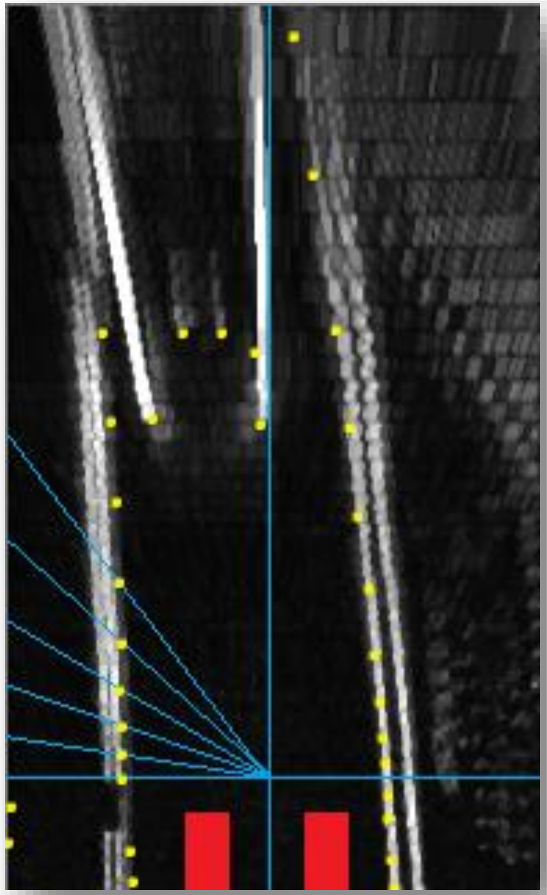


1.8 Outlier filtering

- The outliers
  - points which doesn’t belong to lines but were false detected

- Before line approximation, better to filter them
  - will noise the approximation otherwise

- Algorithm implemented on ARM – works with the detected points only (list of n values according to angle step)

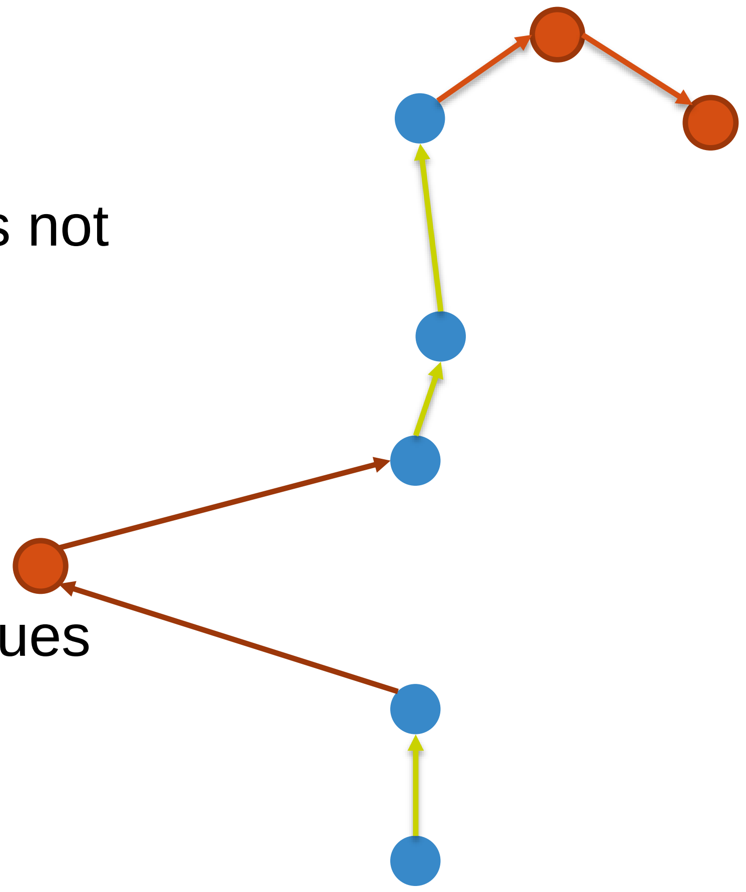


1.8 Outlier filtering

- Filering algorithm
  - Makes use of ordered point list (clockwise)
  - Passes all points and checks the angle between neighbors

  - **Premise:** The angle between two inlier points of the line does not change a lot – even it's the curved line

- All impulse noise is filtered out
- If the defined window of samples changes direction (in front of the car), the algorithm will close the line and continues in second direction (right side line)

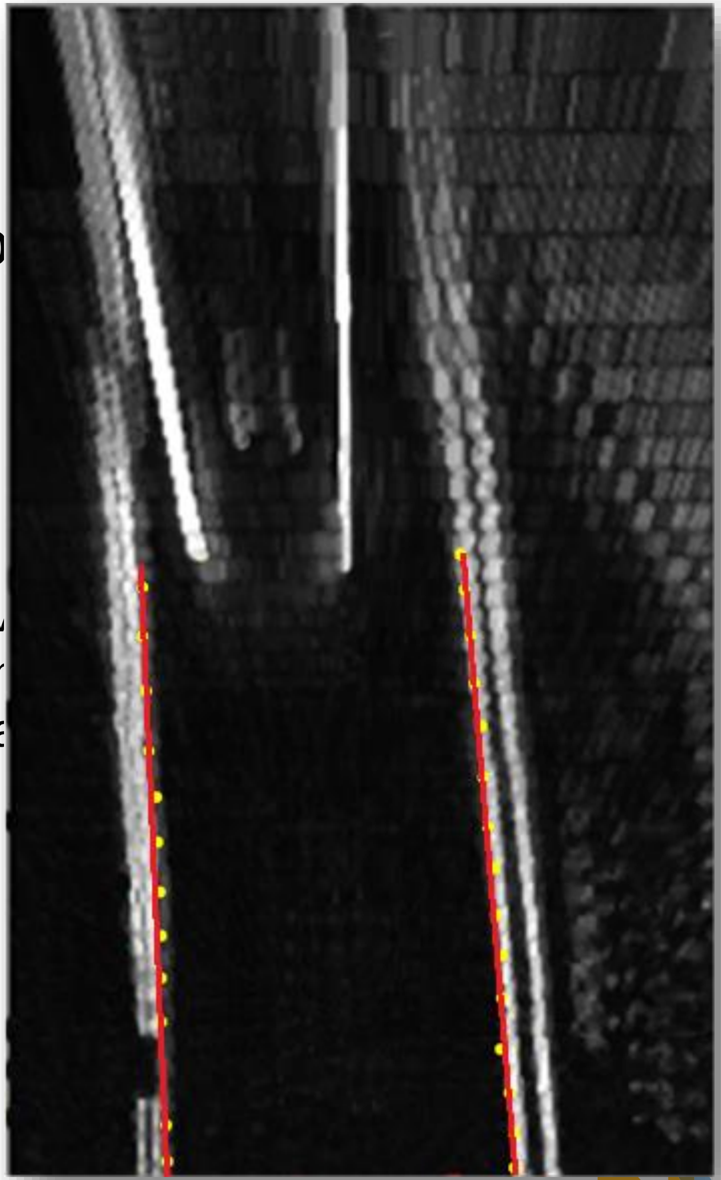


1.8 Line Approximation

- Input – two lists of filtered points – left and right line
- Output – two line equations going through detected po

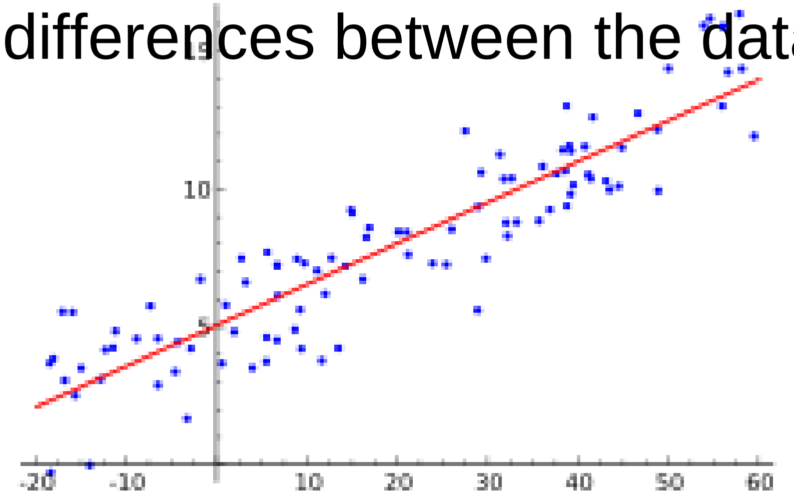
- Least squares method used for line approximation on

- Mathematically, linear least squares is the problem of approximately solv linear equations, where the best approximation is defined as that which r differences between the data values and their corresponding modeled va



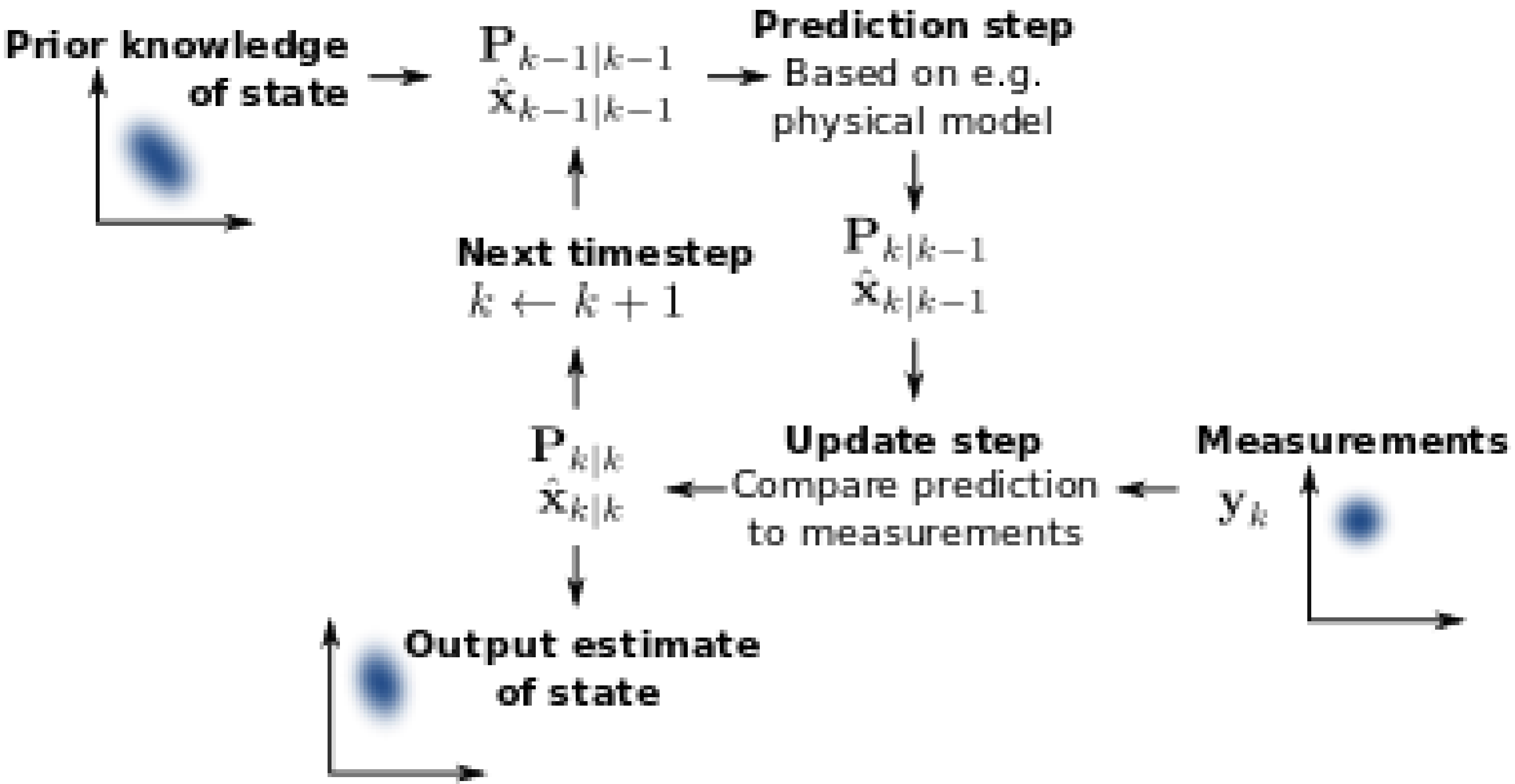
1.8 Line Approximation

- Mathematically, linear least squares is the problem of approximately solving an overdetermined system of linear equations, where the best approximation is defined as that which minimizes the sum of squared differences between the data values and their corresponding modeled values

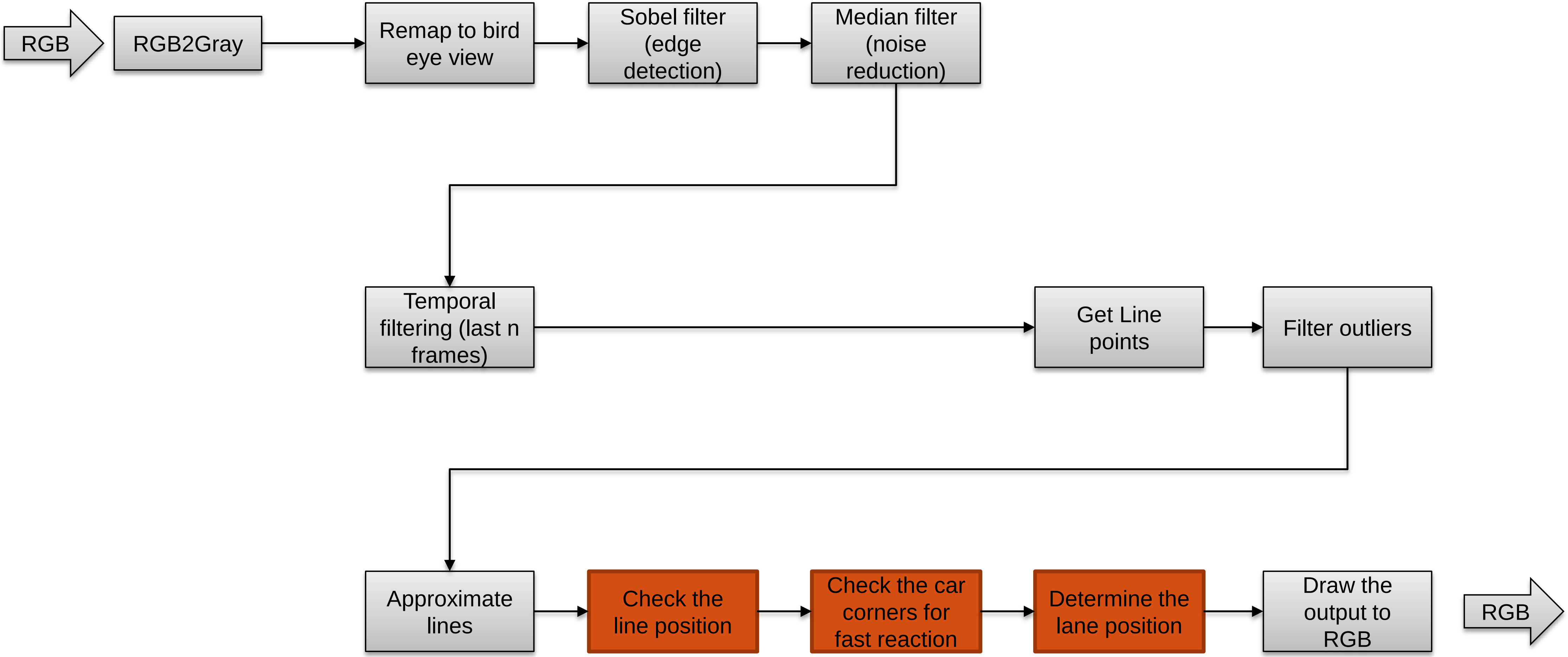


1.8 Kalman Filter

- Detected lines are smoothed by Kalman filter



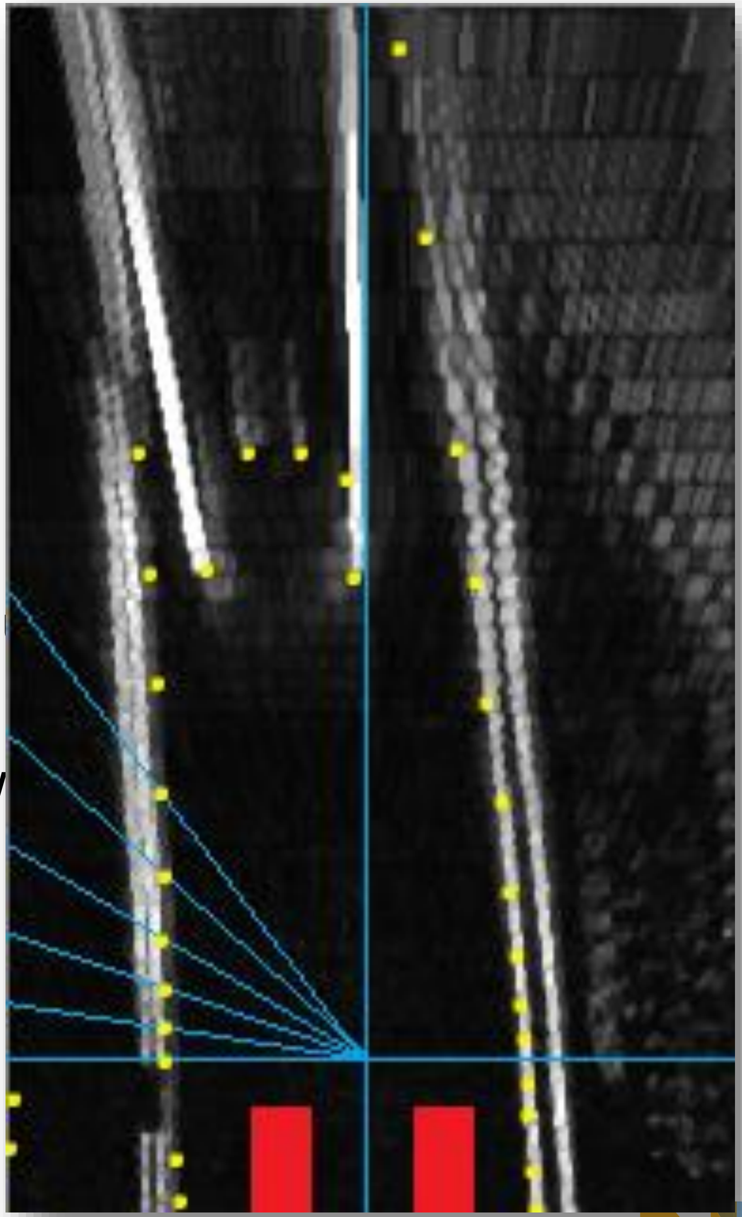
1.9 Lane Departure Warning



1.9 Lane Departure Warning

- At this point, we have lines computed
- We can use the info for Lane Departure Warning
- 2 Way check:
  - Line position towards the car
  - Direct check before each wheel
    - Average value inside the red squares for double checking he res

- If both checks are fine for several frames, we can say w
- If red squares are broken, we immediately signal LD
- If lines are in the wrong position, we signal LD



HANDS-ON: LANE DETECTION

2.5 Hands on – adjusting the parameters

- Kalman Filter Noise
- gedit ~/s32v234_sdk/demos/N1791_LDW/include/config_ldw.h

```
#define CONFIG_KALMAN_MEASUREMENT_NOISE    0.05

// slower response
#define CONFIG_KALMAN_MEASUREMENT_NOISE    0.25

// quicker response
#define CONFIG_KALMAN_MEASUREMENT_NOISE    0.005
```

Lane Detection: Step 2

- On Your Host:
  - Build your application:
    - cd ~/s32v234_sdk/demos/N1791_LDW/build-v234ce-gnu-linux-d/
    - ./build.sh
  - Copy the generated binary to your Network File System:
    - cp apex_isp_ldw_cv.elf ~/rootfs/s32v234/demos/

- On Your Target (Serial Console):
  - Stop the previous demo and run the generated binary:
    - ../s32v234/demos/apex_isp_ldw_cv.elf
  - Observe the results on the screen

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT